Testing a distributed system: generating minimal synchronised test sequences that detect output-shifting faults

R.M. Hierons
Brunel University

March 6, 2001

Abstract

A distributed system may have a number of separate interfaces called ports and in testing it may be necessary to have a separate tester at each port. This introduces a number of issues, including the necessity to use synchronised test sequences and the possibility that output-shifting faults go undetected. This paper considers the problem of generating a minimal synchronised test sequence that detects output-shifting faults when the system is specified using a finite state machine with multiple ports. The set of synchronised test sequences that detect output-shifting faults is represented by a directed graph G and test generation involves finding appropriate tours of G. This approach is illustrated using the test criterion that the test sequence contains a test segment for each transition.

keywords: multiple ports, output-shifting faults, synchronised test sequence, test minimisation.

1 Introduction

The increasing significance of distributed systems has lead to much interest in issues relating to the development of such systems. An important aspect of this is test generation: it is vital to have test techniques that are both effective and efficient.

A distributed system may have several possible sources of input and destinations for output. These sources and destinations are called *ports* and may be spread over a wide area. Thus, for example, when testing a layer of a protocol stack there might be an upper tester and a lower tester [4]. The existence of separate ports may lead to a test architecture in which there is one tester at each interface. This introduces a number of issues in testing [3].

Finite State Machines (FSMs) are used to model a number of classes of system including communications protocols [21] and control circuits [13] and may be used to describe the control structure of a system specified using a language such as SDL, Estelle or Statecharts [18, 15, 12, 11]. While specifications in these languages are usually extended finite state machines (finite state machines with data) rather than FSMs, these may be converted to FSMs by either applying some abstraction or expanding out the data (after making the ranges finite). This has lead to interest in the automatic generation of tests from FSMs (see, for example, [1, 24, 9, 10, 22]). Traditionally, the interest in FSM based testing has largely been limited to protocol conformance testing and the testing of embedded control system. However, the use of Statecharts within the UML has recently widened the interest in this topic.

When testing a system with multiple ports, there is one (local) tester at each port. Suppose testing involves the input of x at port p and then x' at port p' ($p \neq p'$). In order for the tester at p' to know when to input x' it must know when x has been input. If the tester at p' does know when x has been input, because it has received output from the transition triggered by x, these two tests are *synchronised*.

Sometimes there is no synchronised test that satisfies the test criterion [19]. When this is the case, it may be possible to allow the testers to communicate: local testers may send *synchronisation messages* to one another. It is then possible to produce a synchronised test sequence. Naturally, when considering test minimisation, the cost of the synchronisation messages should be included. Thus, approaches that produce test sequences and then add synchronising messages may be suboptimal.

Normally the testing problem is further complicated by the absence of a global clock [16]. In the presence of multiple ports, this reduces the ability of the test system to determine the global order of the input and output. It will be assumed that there is no global clock.

When there are multiple ports, the lack of a global clock may make it difficult to determine which input triggered a particular output value. This makes it difficult to detect output being shifted between adjacent transitions in a test. Faults that shift output between adjacent tests are called *output-shifting faults*. Suppose, for example, that the tester at port p_1 is to input x, this is expected to lead to a_2 and a_3 being output at ports p_2 and p_3 respectively, the tester at port p_2 is then to input x' and this is expected to lead to output of a_4 at port p_4. This test sequence is synchronised as the tester at port p_2 inputs x' after it has received output a_2. However, it is possible that the input values trigger the wrong output but no fault is detected. For example, if the input of x leads to the output of a_2 and a_4 at p_2 and p_4 respectively and the input of x' leads to the output of a_3 at p_3, the correct behaviour is seen at each port. The sequencing of these two transitions has allowed the faults to mask one another. While these faults are not detected in testing they might lead to problems when the system is used.

Other authors have considered related problems. [16] introduce the term output-shifting fault and note that a synchronised test sequence need not detect output-shifting faults. [3] show how a minimal set of messages may be added to a given test sequence to produce a synchronised test sequence that detects output-shifting faults. However, where the initial test sequence has been produced to satisfy some test criterion, the separation of test generation into two phases may lead to a suboptimal test: a short initial sequence may require the addition of many messages. [23] consider the problem of generating minimal synchronised test sequences. In order to do this they produce a directed graph in which every path represents a synchronised test sequence. The problem of generating a minimal synchronised test sequence that satisfies some test criterion is then expressed in terms of this directed graph. Naturally, the resultant test sequences need not detect output-shifting faults. In a similar way, we will introduce a directed graph in which every path represents a synchronised test sequence that detects output-shifting faults. Test generation is then expressed in terms of this directed graph.

The paper is structured as follows. Section 2 provides an overview of some FSM theory and the synchronisation problem is described in Section 3. Output-shifting faults are discussed in Section 4. Section 5 defines a digraph in which every walk, starting at a vertex corresponding to the initial state, represents a synchronised test sequence that detects output-shifting faults. A test generation algorithm, based on this digraph, is then given in Section 6. Finally, conclusions are drawn.

2 Preliminaries

2.1 Directed Graphs

A *directed graph (digraph)* G is defined by a tuple (V, E) in which V is a set of *vertices* and E is a set of *edges*. Each edge is defined by its initial and final vertices and may have a label. Thus, an edge e is defined by a tuple (v_i, v_j, l) in which v_i is the *initial vertex*, v_j is the *final vertex* and l is the *label*.

A *walk* is a sequence of edges $e_1, \ldots, e_m$, $e_i = (v^i, v^{i+1}, l^i)$. The walk $e_1, \ldots,$ e_m is a *path* if no vertex is repeated ($\forall i, j.1 \leq i < j \leq m+1 \Rightarrow v^i \neq v^j$) and a *circuit* if $e_1, \ldots, e_{m-1}$ is a path and $v^1 = v^{m+1}$. A *tour* is a walk that starts and ends at the same vertex.

A digraph $G = (V, E)$ is said to be *strongly connected* if for each ordered pair of vertices (v, v') there is a path from v to v'. G is *weakly connected* if the undirected graph, generated by removing the direction from each edge, is connected.

Given a vertex $v \in V$, $indegree_E(v)$ denotes the number of edges from E that enter v and $outdegree_E(v)$ denotes the number of edges from E that leave v. $G = (V, E)$ is *symmetric* if $\forall v \in V.indegree_E(v) = outdegree_E(v)$. A tour is *Eulerian* if it contains each edge exactly once and it is then said to be an *Euler Tour*. G is *Eulerian* if it has an Euler Tour. It is known that G is Eulerian if and only if G is symmetric and strongly connected and that G is strongly connected if it is weakly connected and symmetric.

It is possible to generalise the problem of generating an Euler Tour to: given a digraph $G = (V, E)$ and some special set of edges, $E_C \subseteq E$, find the shortest tour that contains each edge from E_C. This problem is called the *rural Chinese* *postman problem (RCPP)*. For more on digraphs see, for example, [8].

2.2 Finite State Machines

A deterministic completely specified FSM M is defined by a tuple $(S, s_1, \delta, \lambda, X, Y)$ in which $S = \{s_1, \dots, s_n\}$ is the finite set of *states*, $s_1 \in S$ is the *initial state*, δ is the *state transfer function*, λ is the *output function*, X is the finite *input alphabet* and Y is the *output alphabet*. Only deterministic completely specified FSMs will be considered here.

If M is in state s_i and receives input $x \in X$ it executes a *transition* t, moving to state $s_j = \delta(s_i, x)$ and producing output $y = \lambda(s_i, x)$. Then t is defined by $(s_i, s_j, x/y)$. Let T denote the set of transitions. Thus (S, s_1, T) forms an alternative characterisation of M.

The functions δ and λ may be extended, to take input sequences, producing δ^* and λ^* respectively. Given set A let A^* denote the set of sequences composed of zero or more elements from A and let Λ denote the empty sequence. If $x \in X$ and $x' \in X^*$ the functions δ^* and λ^* are defined by:

- $\delta^*(s, \Lambda) = s$
- $\delta^*(s, xx') = \delta^*(\delta(s, x), x')$
- $\lambda^*(s, \Lambda) = \Lambda$
- $\lambda^*(s, xx') = \lambda(s, x)\lambda^*(\delta(s, x), x')$

An FSM may be represented by a digraph $G = (V, E)$ in which each state s_i is represented by a vertex v_i and a transition $t = (s_i, s_j, x/y)$ is represented by an edge $e = (v_i, v_j, x/y)$. An FSM is said to be *strongly connected* if the corresponding digraph is strongly connected. Only strongly connected FSMs will be considered.

Consider the FSM represented by Figure 1. Here the state set is $\{s_1, s_2, s_3\}$ and the initial state is s_1. The input and output alphabets are $\{a, b\}$ and $\{0, 1\}$ respectively. If this FSM receives input a when in state s_2 it produces output 1 and moves to state s_3. This defines a transition $(s_2, s_3, a/1)$. Clearly this FSM is strongly connected.

Two states s_i and s_j of M are *distinguishable* if there is some $x \in X^*$ such that $\lambda^*(s_i, x) \neq \lambda^*(s_j, x)$. The input sequence x is then said to *distinguish* s_i and s_j. For example, states s_1 and s_3 of M^* are distinguished by input sequence ba: the input of ba in state s_1 leads to output 10 while the input of ba in state s_3 leads to output 11. Two states s_i and s_j are *equivalent* if no input sequence distinguishes them. This may be extended to states from different FSMs M and M' by taking the disjoint union of M and M'. Based on this, FSMs M and M' are *equivalent* if, and only if, their initial states are equivalent. FSM M is *minimal* if there is no equivalent FSM with fewer states. Then a strongly connected FSM M is minimal if and only if no two states of M are equivalent. Any FSM may be rewritten to an equivalent minimal FSM [17] and thus only minimal FSMs will be considered.

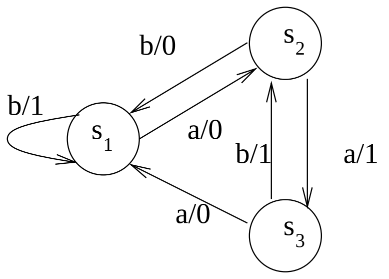

Figure 1: An FSM

When testing *implementation under test (IUT)* I against FSM M, it is normal to assume that I behaves like some unknown FSM M_I. The IUT I is *correct* if and only if M_I is equivalent to M. Testing thus reduces to determining FSM equivalence.

2.3 Testing from an FSM

When testing from an FSM there are many alternative test criteria. Many of these test criteria are based on test purposes that insist that the test sequence used contains certain subsequences. This section will describe one such test criterion.

Many FSM test techniques involve testing individual transitions [20]. There are two types of fault that a transition may have: *output faults*, in which the transition produces the wrong output, and *state transfer faults* in which the transition takes the system to the wrong state. Thus, in order to detect state transfer faults when testing a transition t it is necessary to follow t by one or more sequences that check its final state. Sequences that distinguish, and thus check, the states of M are used.

The sequence u is said to be a *unique input/output sequence (UIO)* for state s_i if $\forall 1 \leq j \leq n . i \neq j \Rightarrow \lambda^*(s_i, u) \neq \lambda^*(s_j, u)$. The sequence u is capable of checking state s_i of M but not necessarily any other state of M. While some FSMs do not have a UIO for each state, many authors have considered the use of UIOs [1, 9, 10, 22]. For information on other approaches to state checking see, for example [20].

Let u_j denote the UIO for state s_j. In order to test a transition $t = (s_i, s_j, x/y)$ it is possible to use a sequence that contains the *test segment* tu_j. This test segment is in the form of a transition t followed by a UIO u_j that checks the final state of the transition. The test criterion, that the test sequence contains such a test segment for each transition, will be called the *U-criterion*. In order to illustrate test generation, this paper will consider the use of the U-criterion when there are multiple ports.

Aho et al. [1] represent the problem of finding a minimal test sequence that satisfies the U-criterion as an instance of the rural Chinese postman problem (RCPP) by modelling the FSM as a digraph $G = (V, E)$ and adding a set E_C of extra edges that represent the test segments. While the RCPP is known to be NP-complete [14] there are polynomial time algorithms that solve it under certain conditions. One approach [1] finds the minimal symmetric augmentation E' of E_C, adding edges from $E \cup E_C$. If E' is strongly connected and contains an edge incident to v_1 then an Euler Tour of E' may be found. This Euler Tour provides the optimal test sequence. Otherwise edges may be added in order to generate some minimal E'' that contains E', is strongly connected and contains an edge incident to v_1. An Euler Tour of E'' is then found. In the latter case, the test sequence generated may be suboptimal.

This algorithm has low order polynomial complexity [1]. The test sequence is optimal whenever E_C is weakly connected and contains an edge incident to v_1. A sufficient condition for this algorithm to generate an optimal solution is that M has a *reset operation*: an input value that takes every state to s_1 [1].

2.4 FSMs with multiple ports

When a system interacts with its environment at a number of ports it is necessary to extend the FSM notation to include information about ports. This may be achieved through having distinct input and output alphabets associated with each port. Then a transition is triggered by input from one port and a transition may send output to one or more ports. Thus the output of a transition may be represented by a set or vector [3]. The model used in this paper will now be described.

Let $P = \{p_1, \ldots, p_r\}$ denote the set of ports of M. Associated with each port $p_i \in P$ there is an input alphabet X_{p_i} and an output alphabet Y_{p_i}. The input and output alphabets are distinct: for all $p_i, p_j \in P$, $X_{p_i} \cap Y_{p_j} = \emptyset$, $i \neq j \Rightarrow X_{p_i} \cap X_{p_j} = \emptyset$ and $i \neq j \Rightarrow Y_{p_i} \cap Y_{p_j} = \emptyset$. Let $\overline{Y}$ denote the set of vectors of length r whose ith component is either the null value ϵ or is from Y_{p_i} $(1 \leq i \leq r)$ and that contain at least one value that is not null. Then $\overline{Y} = (Y_{p_1} \cup \{\epsilon\}) \times \ldots \times (Y_{p_r} \cup \{\epsilon\}) \backslash \{(\epsilon, \ldots, \epsilon)\}$ and $\overline{Y}$ represents the output alphabet of M.

Let $X = X_{p_1} \cup \ldots \cup X_{p_r}$. An FSM with port set P is thus defined by a tuple $(S, s_1, \delta, \lambda, X, \overline{Y}, P)$, where δ is the state transfer function (type $S \times X \rightarrow S$) and λ is the output function (type $S \times X \rightarrow \overline{Y}$). Note that the model is restricted to single concurrent inputs: the FSM cannot received two values at exactly the same time.

Consider the FSM given in Figure 2, in which null output is represented by the - symbol and the transitions are labeled with $t1, \ldots, t6$ as well as the input/output behaviour. This FSM will be denoted M_0 throughout the paper. M_0 has two ports A and B. The input alphabet are $X_A = \{\alpha\}$ and $X_B = \{\beta\}$ and the output alphabets are $Y_A = \{a\}$ and $Y_B = \{b, c\}$. Here, for example, $\delta(s_2, \alpha) = s_2$, $\lambda(s_2, \alpha) = (a, c)$, and $\lambda(s_2, \beta) = (\epsilon, c)$.

For the rest of the paper, an FSM with multiple ports will simply be referred to as an FSM. Throughout this paper it will be assumed that M is a determin-

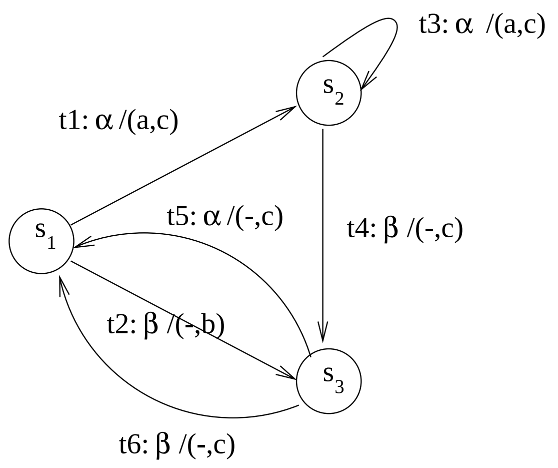

Figure 2: The FSM M_0

istic, strongly connected FSM $(S, s_1, \delta, \lambda, X, Y, P)$ and $S = \{s_1, \ldots, s_n\}$.

When there are multiple ports the test system may be divided into two classes of entity: *testers* and *observers* [6, 7]. Testers are responsible for providing input and receiving output while the observers record the interaction between the testers and the system. Testers are local and thus each port p_i has a separate tester which will be denoted T_i. There may be one observer for each port, a global observer or a combination of these.

The type of error detected by the observers, when a test is executed, depends upon the type of observer used. Ideally, there is a global observer that knows the timings, and thus order, of all input and output. When the system is distributed and there is no global clock this is, however, difficult to implement. In this paper it will be assumed that there are local observers and testers, that any tester may send a message to any other tester and that these messages are observed by the corresponding observers. It will be assumed that the communications medium does not introduce errors.

When there are multiple testers that may communicate, there are a number of possible architectures. One example is to have a ring network connecting the testers [3]. Alternatives include the use of a bus or a mesh network. Naturally, the test minimisation problem may depend upon the test architecture and the form of communications used. This paper will assume that all the testers are connected, that the cost of sending a message between two testers is constant: it does not depend upon the locations of the testers involved. It will also be assumed that multicast communications are not used. Later we will briefly outline how the method might be altered to allow different architectures and to allow multicast communications.

It is important to say what it means for an IUT I to be correct relative to an FSM M with multiple ports. This is defined by: the behaviour exhibited by I in response to test sequence x is correct if, for each port p_i, the sequence of interactions seen at p_i is that expected. Then I is correct relative to M if it behaves correctly on each input sequence of M.

3 The Synchronisation Problem

Given $x \in X$ let $port(x)$ denote the port associated with x and let the tester at port p_i be denoted T_i. Further, given some $\overline{y} = (y_1, \ldots, y_r) \in \overline{Y}$ let $\overline{y}_i$ denote y_i $(1 \leq i \leq r)$ and let $ports(\overline{y})$ denote the set of ports associated with values from $\overline{y}$ that are not null.

Given a test sequence it is vital that the individual testers know when to input values from this. Consider, for example, the subsequence tt', for $t = (s_i, s_j, x/\overline{y})$ and $t' = (s_j, s_k, x'/\overline{y}')$. Since x' should not be input until after x has been input, in order for the tester $T_{port(x')}$ to know when to input the value x' it is necessary for this tester to know when t has been executed. The tester $T_{port(x')}$ receives this information if either $port(x') = port(x)$ or $port(x') \in ports(\overline{y})$. In the first case, the input x' follows x. In the second case, the input of x' follows $T_{port(x)}$ receiving output from t.

Definition 1 *The sequence tt', for transitions $t = (s_i, s_j, x/\overline{y})$ and $t' = (s_j, s_k, x'/\overline{y}')$, has a* synchronisation problem *if $port(x') \neq port(x)$ and $port(x') \notin ports(\overline{y})$.*

If tt' does not have a synchronisation problem then it is said to be *synchronised*. Further, a test sequence is said to be *synchronised* if all subsequences within it are synchronised. Given a test criterion and an FSM M, there may be no synchronised test sequence for M that satisfies the test criterion [2].

In some cases it is possible to include *synchronisation messages* between the testers. These allow one tester to inform another tester of an input or output event. Let $sy_{\Delta\Gamma}$ denotes a synchronisation message from port Δ to port Γ. The synchronisation message $sy_{\Delta\Gamma}$ acts as a transition that involves input from the tester at Δ, output to the tester at Γ and does not affect the IUT.

Consider the example M_0 and the sequence $\beta/(\epsilon, c), \alpha/(\epsilon, c)$ from state s_2. Since the first transition involves interaction at port B only and the second involves input at A, there is a synchronisation problem. However, it is sufficient to place a synchronisation message, from B to A, between the two transitions. While $\beta/(\epsilon, c), \alpha/(\epsilon, c)$ is not synchronised, $\beta/(\epsilon, c), sy_{BA}, \alpha/(\epsilon, c)$ is synchronised.

The cost of synchronisation messages should be considered in any test minimisation procedure. In this paper it will be assumed that synchronisation messages can be sent between any two testers. As noted earlier, the cost of sending messages between testers might depend upon factors relating to the architecture and the communications method used. In this paper it will be assumed that the cost of sending a message between two testers is constant and that multicast communications are not being used.

State	UIO	Transition sequence	Final State
s_1	$\beta/(\epsilon, b)$	$t2$	s_3
s_2	$\beta/(\epsilon, c), \beta/(\epsilon, c)$	$t4t6$	s_1
s_3	$\alpha/(\epsilon, c)$	$t5$	s_1

Figure 3: The UIOs for M_0

The problem of finding the minimal synchronised test sequence in the presence of two ports has been represented as an instance of the RCPP [5]. This has been generalized to multiple ports [23]. These approaches do not, however, detect output-shifting faults [23]. This class of fault will be discussed further in Section 4.

When necessary, the UIOs used in testing contain synchronisation messages. In the example, each state has a UIO that does not require the addition of synchronisation messages within it. These UIOs are given in Figure 3.

4 Output-shifting Faults

Suppose IUT I has multiple ports, receives input sequence $x \in X^*$ and produces output sequence α_i at port p_i ($1 \leq i \leq r$). We will see that while each value in α_i is triggered by some value in x, it is not always possible to match the values in α_i to those in x. Even if the behaviour observed at each port is that expected, some of the transitions may have produced incorrect output, faults masking one another. These faults might produce erroneous output when the transitions are executed in a different sequence.

In the following, given $\overline{y} \in \overline{Y}$ and $y \in Y_{p_i}$ let $\overline{y} \oplus (i, y)$ denote $\overline{y}$ with its ith component replaced by y. Suppose two transitions $t = (s_i, s_j, x/\overline{y})$ and $t' = (s_j, s_k, x'/\overline{y}')$ are sequenced to form tt' and tt' is synchronised. Suppose also that there is some i such that $\overline{y}_i \neq \epsilon$, $\overline{y}'_i = \epsilon$ and $port(x') \neq p_i$. Then while tt' is synchronised it will fail to detect a fault in which x triggers output $\overline{y} \oplus (i, \epsilon)$ and x' triggers output $\overline{y}' \oplus (i, \overline{y}_i)$. This is because the output sequence observed at each port is that expected. Similarly, it may fail to detect an output expected in response to x' being produced in response to x. These cases represent the following two types of output-shifting fault.

Definition 2 *Suppose two transitions $t = (s_i, s_j, x/\overline{y})$ and $t' = (s_j, s_k, x'/\overline{y}')$ are sequenced to form tt'. Suppose also that $\overline{y}_i \neq \epsilon$, $\overline{y}'_i = \epsilon$, $port(x') \neq p_i$, and that the actual outputs in the corresponding transitions in the IUT are $\overline{y} \oplus (i, \epsilon)$ and $\overline{y}' \oplus (i, \overline{y}_i)$ respectively. This combination of faults is called a* forward output-shifting fault *[25].*

Definition 3 *Suppose two transitions $t = (s_i, s_j, x/\overline{y})$ and $t' = (s_j, s_k, x'/\overline{y}')$ are sequenced to form tt'. Suppose also that $\overline{y}_i = \epsilon$, $\overline{y}'_i \neq \epsilon$, $port(x') \neq p_i$, and* *that the actual outputs in the corresponding transitions in the IUT are* $\overline{y} \oplus (i, \overline{y}'_i)$ *and* $\overline{y}' \oplus (i, \epsilon)$ *respectively. This combination of faults is called a* backward output-shifting fault *[25].*

Definition 4 *A fault is an* output-shifting fault *[16] if it is either a forward output-shifting fault or a backwards output-shifting fault.*

Then the *output-shifting problem* is: given a test criterion, generate a test sequence that satisfies this test criterion and that detects output-shifting faults. Naturally, it is desirable to use synchronised test sequences that detects output-shifting faults. The problems of detecting forward and backwards output-shifting faults for tt' ($t = (s_i, s_j, x/\overline{y})$ and $t' = (s_j, s_k, x'/\overline{y}')$) will now be considered.

4.1 Detecting forward output-shifting faults

In order to detect a forward output-shifting fault in tt' it is sufficient for the tester at $port(x')$ to know when all the expected output values have been produced in response to t: x' is not input until all the $\overline{y}_i$ have been received. A fault is detected through the absence of one of these values. Thus, in order to detect forward output-shifting faults, it is sufficient for $T_{port(x')}$ to know when all the $\overline{y}_i$ have been received by the corresponding testers.

In order for $T_{port(x')}$ to know when $\overline{y}_i$ has been received it is sufficient for a message to be sent from p_i to $port(x')$ once T_{p_i} receives $\overline{y}_i$. Clearly, if $p_i = port(x')$ this message is not required. This may be done for each port $p \in ports(\overline{y}) \backslash \{port(x')\}$. These messages help to identify the completion of transition t and will be called *post-transition framing messages*. A post-transition framing message from port Δ to port Γ will be denoted $post_{\Delta\Gamma}$.

When $ports(\overline{y}) = \{p_i\}$, some $p_i \neq port(x')$, the message that follows t, in order for x' to be input, is equivalent to a synchronisation message from p_i to $port(x')$. We will not distinguish between post-transition framing messages following a transition with only one output and synchronisation messages.

4.2 Detecting backwards output-shifting faults

In order to detect a backward output-shifting fault in tt' it is sufficient for the following conditions to be satisfied.

1. Any extra output, in response to x, is received before x' is input.
2. If extra output is received before x' is input, this fault is observed.

In order to guarantee that any extra output, in response to x, is produced before x' is input, $T_{port(x')}$ can wait for some time Δ_1 after all the members of $ports(\overline{y})$ have received the expected values. Then, assuming Δ_1 is sufficiently long, any extra output generated in response to x should appear before x' is input. This approach relies upon the test hypothesis that, if the system fails to produce any output during a time period of length Δ_1 or more then it will not output any more values until it receives further input. The choice of Δ_1 will depend upon properties of the system.

In order to observe output from $\overline{y}'$ being erroneously produced in response to x it is sufficient for each observer at a port from $ports(\overline{y}')$ to know when x' is input. This may be achieved by $T_{port(x')}$ sending a message to each port in $ports(\overline{y}')$ before x' is input. These messages identify the beginning of transition t' and will be called *pre-transition framing messages.* A pre-transition framing message from port Δ to port Γ will be denoted $pre_{\Delta\Gamma}$. $T_{port(x')}$ may then wait some fixed time Δ_2, before it inputs x', in order to guarantee that these messages are received before x' is input. Any values received by a port from $ports(\overline{y}')$, before this message, were triggered by x rather than x'.

4.3 Reducing the number of framing messages

The framing messages described above are sufficient. However

1. the generation of post-transition framing messages for t might take advantage of the knowledge that it is to be followed by t';

2. the generation of pre-transition framing messages for t' might take advantage of the knowledge that it is to be preceded by t.

In general, framing messages are only required for ports when output-shifting faults are possible. These are those ports for which there is some value for exactly one of $\overline{y}$ and $\overline{y}'$. In particular

1. after t a post-transition framing message is only required from a port p_i if $\overline{y}'_i = \epsilon$.

2. before t' a pre-transition framing message is only required from a port p_i if $\overline{y}_i = \epsilon$.

Also, output-shifting faults cannot appear at $port(x')$, as the input of x' separates any output received at this port in response to x and x'. Thus the following guarantees the detection of output-shifting faults in tt'.

1. Each tester at a port from $ports(\overline{y})\backslash(ports(\overline{y}') \cup \{port(x')\})$ sends a post-transition framing message to $port(x')$ upon receiving the expected output in response to x.

2. If $ports(\overline{y})\backslash(ports(\overline{y}')\cup\{port(x')\}) = \emptyset$ and $port(x') \notin ports(\overline{y})\cup\{port(x)\}$ then send a synchronisation message from the tester at $port(x)$ to the tester at $port(x')$. This is required since otherwise post-transition framing messages would not be sent and thus the transitions would not be synchronised.

3. $T_{port(x')}$ waits time Δ_1 after receiving all of these messages.

4. $T_{port(x')}$ sends a pre-transition framing message to each tester at a port in $ports(\overline{y}')\backslash(ports(\overline{y}) \cup \{port(x')\})$.

5. $T_{port(x')}$ waits time Δ_2.

6. $T_{port(x')}$ inputs the value x'.

Each transition of a test sequence, except the first and last, may thus be preceded by pre-transition framing messages and followed by post-transition framing messages. If these steps are followed then tt' is said to be *p-synchronised.* A test sequence is *p-synchronised* if every subsequence within it is p-synchronised. In order to simplify the analysis it will be assumed that $\Delta_2 = 0$.

5 The digraph G

This section will define a digraph $G = (V, E)$, in which every walk represents a p-synchronised sequence. When the test criterion being used involves the test sequence containing certain subsequences, G may be augmented by edges representing these subsequences. The problem of generating a minimal p-synchronised test sequence is then an instance of the RCPP. This approach will be applied to M_0, using the U-criterion, in Section 6.

In order to consider test minimisation, it is necessary to know the costs of transitions and synchronisation/framing messages. Recall that it is to be assumed that all synchronisation/framing messages have the same cost. Then it is sufficient to know the relative costs of transitions and synchronisation/framing messages. This will be normalized to give a transition cost 1 and synchronisation and framing messages cost c_s.

The digraph G contains a number of copies of each state. Let the copy of s_i associated with port p be denoted v_i^p. Vertex v_i^p denotes the condition in which M is in state s_i and the synchronisation information allows input at port p. However, in order to represent the information about the output ports of the transitions, which is required to utilise the reductions described in Section 4.3, it will be necessary to add further vertices. Consider a transition $t = (s_i, s_j, x/\overline{y})$.

1. If $ports(\overline{y}) = \{port(x)\}$ then the initial vertex of t is $v_i^{port(x)}$ and the final vertex of t is $v_j^{port(x)}$.

2. Otherwise, the initial vertex of t is I_i^t and the final vertex of t is F_j^t.

Since each v_i^p, I_i^t and F_i^t represents the same vertex there will be edges between these. These edges will represent the sending of synchronisation or framing messages and will be described below.

In M_0 the transition t_2 has input and output at port B and thus t_2 leads to an edge from v_1^B to v_3^B. The transition t_1 sends output to both ports and thus is represented by an edge from $I_1^{t_1}$ to $F_2^{t_1}$. The set of edges that represent the transitions of M_0 is given in Figure 4

Transition	Initial vertex	Final Vertex
t_1	$I_1^{t_1}$	$F_2^{t_1}$
t_2	v_1^B	v_3^B
t_3	$I_2^{t_3}$	$F_2^{t_3}$
t_4	v_2^B	v_3^B
t_5	$I_3^{t_5}$	$F_1^{t_5}$
t_6	v_3^B	v_1^B

Figure 4: Edges representing transitions

A vertex v_i^p is only included in G if either an edge representing a transition enters it or an edge representing a transition leaves it.

The vertex set V is the union of the sets V_1, V_2, and V_3 defined below:

1. $V_1 = \{v_i^p | s_i \in S \wedge p \in P\}$
2. $V_2 = \{I_i^t | t = (s_i, s_j, x/\overline{y}) \in T \wedge ports(\overline{y}) \neq \{port(x)\}\}$
3. $V_3 = \{F_j^t | t = (s_i, s_j, x/\overline{y}) \in T \wedge ports(\overline{y}) \neq \{port(x)\}\}$

It is now necessary to consider the edges that represent synchronisation and framing messages. For vertices v_i^p and $v_i^{p'}$, $v_i^p \neq v_i^{p'}$, there is an edge from v_i^p to $v_i^{p'}$, with cost c_s, representing a synchronisation message travelling from p to p'.

Given vertices F_j^t and $I_j^{t'}$, $t = (s_i, s_j, x/\overline{y})$, $t' = (s_j, s_k, x'/\overline{y}')$ there is an edge from F_j^t to $I_j^{t'}$. The cost of this is determined by the number of framing messages required between t and t'. This is given by the following two cases, in which the first is where a synchronisation message is required but no post-transition framing messages are required.

1. If $port(x') \notin (ports(\overline{y}) \cup \{port(x)\})$ and $ports(\overline{y}) \setminus ports(\overline{y}') = \emptyset$ then cost $c_s(1 + |ports(\overline{y}') \setminus (ports(\overline{y}) \cup \{port(x')\})|)$.
2. Otherwise cost $c_s(|ports(\overline{y}) \setminus (ports(\overline{y}') \cup \{port(x')\})| + |ports(\overline{y}') \setminus (ports(\overline{y}) \cup \{port(x')\})|)$.

There are also edges between the v_i^p and the I_i^t and F_i^t: these represent the required framing messages without any reductions applied. These are given by the following.

1. Given I_i^t, $t = (s_i, s_j, x/\overline{y})$, there is an edge from $v_i^{port(x)}$ to I_i^t with cost $c_s|ports(\overline{y}) \setminus \{port(x)\}|$. This represents the pre-transition framing messages for t.

2. Given F_j^t, $t = (s_i, s_j, x/\overline{y})$, and port $p \in ports(\overline{y}) \cup \{port(x)\}$ there is an edge from F_j^t to v_j^p with cost $c_s|ports(\overline{y})\backslash\{p\}|$. This represents the post-transition framing messages for t when it is followed by input at port p. These edges are to any vertex representing either $port(x)$ or a port that receives input from t: the ports that can next provide input without the addition of a synchronisation message.

The edge set E is the union of the following sets in which the label sy indicates a synchronisation message and $frame$ represents one or more framing messages.

1. $E_1 = \{(v_i^p, v_j^p, x/\overline{y}) | \exists t = (s_i, s_j, x/\overline{y}) \in T.port(x) = p \wedge ports(\overline{y}) = \{p\}\}$.
2. $E_2 = \{(I_i^t, F_j^t, x/\overline{y}) | t = (s_i, s_j, x/\overline{y}) \in T \wedge ports(\overline{y}) \neq \{port(x)\}\}$.
3. $E_3 = \{(v_i^p, I_i^t, frame) | t = (s_i, s_j, x/\overline{y}) \in T \wedge v_i^p, I_i^t \in V \wedge p = port(x)\}$.
4. $E_4 = \{(F_j^t, v_j^p, frame) | t = (s_i, s_j, x/\overline{y}) \in T \wedge F_j^t, v_j^p \in V \wedge (p \in ports(\overline{y}) \vee p = port(x))\}$.
5. $E_5 = \{F_j^t, I_j^{t'}, frame) | t = (s_i, s_j, x/\overline{y}), t' = (s_j, s_k, x'/\overline{y}') \in T \wedge I_j^{t'}, F_j^t \in V\}$.
6. $E_6 = \{(v_i^p, v_i^{p'}, sy) | v_i^p, v_i^{p'} \in V \wedge p \neq p'\}$.

Here the sets E_1 and E_2 represent the transitions. The sets E_3, E_4, and E_5 involve the addition of framing messages, E_5 being the case where optimisations are included through knowing both transitions involved in a subsequence. Finally, E_6 represents the addition of synchronisation messages.

There is a one-to-one correspondence between walks of G that start at some v_1^p and p-synchronised test sequences. Thus test generation can be represented in terms of finding appropriate walks in G.

Recall that two assumptions, about the communications between testers, were made. The first of these was that multicast communications is not used. When multicast communications are used, single pre and post framing message suffice for each transition. This impacts upon the costs of the edges in E_3, E_4, and E_5 but does not otherwise affect G. It has also been assumed that the cost of messages between testers is fixed. When this is not the case, the costs of the edges in E_3, E_4, E_5, and E_6 will be affected but, again, this does not affect the structure of G.

6 Test Generation

This section will explore the problem of generating a p-synchronised test sequence, that satisfies the U-criterion, in the presence of a known set of UIOs. Given a transition $t = (s_i, s_j, x/\overline{y})$, it is possible to add an edge e_t that represents t followed by UIO u_j, with framing and synchronisation messages included

Transition	Test	Transitions	Cost
$t1$	$t1, post_{AB}, \beta/(\epsilon, c), \beta/(\epsilon, c)$	$t1, post_{AB}, t4, t6$	$3 + c_s$
$t2$	$t2, sy_{BA}, \alpha/(\epsilon, c)$	$t2, sy_{BA}, t5$	$2 + c_s$
$t3$	$t3, post_{AB}, \beta/(\epsilon, c), \beta/(\epsilon, c)$	$t3, post_{AB}, t4, t6$	$3 + c_s$
$t4$	$t4, sy_{BA}, \alpha/(\epsilon, c)$	$t4, sy_{BA}, t5$	$2 + c_s$
$t5$	$t5, \beta/(\epsilon, c)$	$t5, t2$	2
$t6$	$t6, \beta/(\epsilon, c)$	$t6, t2$	2

Figure 5: The subsequences in E^* for M_0.

where necessary. The initial vertex of e_t is the initial vertex of the edge that corresponds to t. Similarly, the final vertex of e_t is the final vertex of the edge that corresponds to the last transition from u_j. Let the set of such edges be denoted E^*. The cost of e_t may be found by taking the sequence of edges of G that comprise e_t and summing the costs of these edges.

Consider, for example, the transition $t1 = (s_1, s_2, \alpha/(a, c))$ from M_0. Then $e_{t1} = t1 post_{AB} t4 t6$ and the final vertex of e_{t1} is the final vertex of the edge that corresponds to $t6$, which is v_1^B. The cost of e_{t1} is given by the sum of the cost of $t1$, the costs of $post_{AB}$, the cost of $t4$ and the cost of $t6$. The test subsequences for M_0 are given in Figure 5.

Let G^* denote G augmented by E^*: $G^* = (V, E \cup E^*)$. The final step involves finding a minimal tour of G^* that contains every element of E^*. This is an instance of the RCPP and can be solved using any of the standard approaches. A test sequence may be generated from the resulting tour by starting the tour at some v_1^p.

Assuming that UIOs for each state are known, the test generation algorithm may thus be summarized as the following.

Algorithm

1. Generate the digraph G.
2. Determine the test segments for each transition.
3. Develop the set E^* of edges that represent the test segments.
4. Augment G with edge set E^* to form digraph G^*.
5. Find a minimal tour of G^* that contains each edge from E^*.

Consider the FSM M_0. In order to aid simplicity c_s will be set to 1. The augmented digraph, in which some of the edges representing synchronisation and framing messages are not shown, is given in Figure 6. Here test segments for transitions $t1, \ldots, t6$ are represented by $T1, \ldots, T6$. Note that here the vertices v_1^A, v_2^A, and v_3^A are not the source or destination of an edge representing a transition and so are not included. The costs of the edges are not shown.

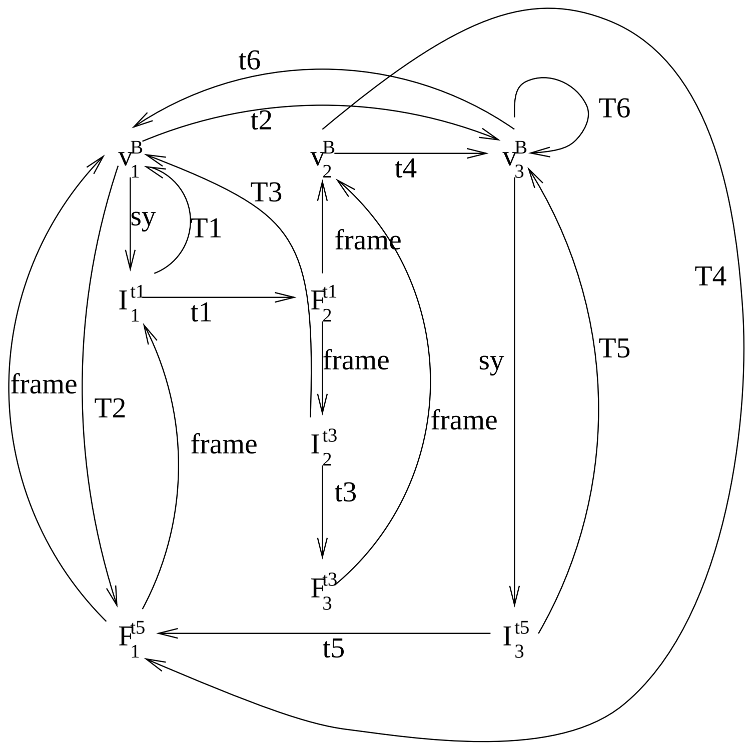

Figure 6: The augmented digraph

However the costs should be clear: for a transition it is 1 and for a test segment it is that given in Figure 5. The only remaining messages are the framing messages. The costs of these are given below.

1. Cost 0 for the edges: $F_2^{t1} \rightarrow I_2^{t3}$, $F_1^{t5} \rightarrow I_1^{t1}$, and $F_1^{t5} \rightarrow v_1^B$.
2. Cost 1 for the edges: $v_1^B \rightarrow I_1^{t1}$, $F_2^{t1} \rightarrow v_2^B$, $v_3^B \rightarrow I_3^{t5}$, and $F_2^{t3} \rightarrow v_2^B$.

The minimal symmetric augmentation of $\{T1, \ldots, T6\}$ is given in Figure 7 in which empty sequences of framing messages are denoted null. Here, an edge name being preceded by an integer n indicates that there are n copies of the edge.

As this minimal symmetric augmentation is not strongly connected, the circuit $t_2 t_6$ is added. This leads to the following tour.

$$v_1^B \rightarrow_{sy} I_1^{t_1} \rightarrow_{T1} v_1^B \rightarrow_{T2} F_1^{t_5} \rightarrow_{null} I_1^{t_1} \rightarrow_{t_1} F_2^{t_1} \rightarrow_{null} I_2^{t_3} \rightarrow_{T3} v_1^B \rightarrow_{sy} I_1^{t_1}$$

$$\rightarrow_{t_1} F_2^{t_1} \rightarrow_{frame} v_2^B \rightarrow_{T4} F_1^{t_5} \rightarrow_{null} v_1^B \rightarrow_{t_2} v_3^B \rightarrow_{T6} v_3^B \rightarrow_{sy} I_3^{t_5} \rightarrow_{T5} v_3^B \rightarrow_{t_6} v_1^B$$

$$\begin{tikzcd}
v_1^B \arrow[d, "2sy"] \arrow[ddd, bend right=20, "T2"'] & v_2^B \arrow[dddl, bend left=90, "T4"] & v_3^B \arrow[loop right, "T6"] \arrow[ddd, "sy"'] \\
I_1^{t1} \arrow[u, bend right=60, "T1"'] \arrow[r, "2t1"] & F_2^{t1} \arrow[u, "frame"'] \arrow[d, "null"] & \\
 & I_2^{t3} \arrow[uul, bend right=40, "T3"'] & \\
F_1^{t5} \arrow[uuu, bend left=40, "null"] \arrow[uu, bend right=40, "null"'] & & I_3^{t5} \arrow[uuu, bend right=30, "T5"']
\end{tikzcd}$$

Figure 7: Minimal Symmetric Augmentation

The resultant input sequence may be produced by taking the input from these edges. This is: $sy_{BA}\alpha post_{AB}\beta\beta\beta sy_{BA}\alpha\alpha\alpha post_{AB}\beta\beta sy_{BA}\alpha sy_{AB}\beta sy_{BA}\alpha$ $sy_{AB}\beta\beta\beta sy_{BA}\alpha\beta\beta$

It is easy to check that this test has cost 27 and contains a test segment for every transition. As the last transition simply returns M_0 to its initial state, and does not contribute to the test, it may be removed. Similarly, the initial synchronisation message may be removed, reducing the test sequence length to 25.

6.1 Properties of the algorithm

Let r denote the number of ports of M and T denote the transition set of M. As every transition from M leads to at most 2 vertices in G^*, G^* has $O(|T|)$ vertices. There are $O(|T|)$ edges that do not represent synchronisation and framing messages. For each I_i^t and F_i^t there are $O(r+|T|)$ edges entering and leaving the vertex. For each $i \in 1 \dots n$, there are $O(r^2)$ edges between the v_i^p. Thus there are $O(nr^2 + r|T|)$ edges that represent synchronisation and framing messages. Thus, G^* has $O(nr^2 + r|T|)$ edges. For fixed r, this gives $O(|T|)$ vertices and $O(|T|)$ edges.

7 Conclusions

While distributed systems are important, they introduce new challenges for the tester. The existence of a number of interfaces, called ports, between the system and its environment complicate testing. When there are multiple ports and distributed testers it is important that any test sequence used is synchronised: otherwise local testers do not know when to send input to the system. The existence of multiple ports also has an impact on the ability of a test sequence to detect faults: some tests fail to detect output-shifting faults. Both the production of synchronised tests and the detection of output-shifting faults may necessitate the addition of messages between the local testers.

This paper has introduced an approach that describes the set of synchronised test sequences that detect output-shifting faults using a directed graph. When test generation can be represented in terms of the inclusion of certain sequences of transitions, test generation can be expresses in terms of this directed graph. The test minimisation problem may then be phrased as an instance of the rural Chinese postman problem. The resultant optimisation problem may be solved using standard algorithms.

One criterion, that the sequence contains a test for each transition, has been considered in this paper. The algorithm given thus generates test sequences that satisfy this criterion.

In this paper it has been assumed that the cost of sending a message between two testers is fixed. It has also been assumed that communication between the testers is not multicast. Alternative assumptions lead to no changes in the structure of the digraph but do affect the costs of edges. Thus, while the assumptions used may influence the result of optimisation it should not be difficult to adapt the approach given here to other architectures and forms of communications.

References

[1] A. V. Aho, A. T. Dahbura, D. Lee, and M. U. Uyar. An optimization technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman Tours. In *Protocol Specification, Testing, and Verification VIII*, pages 75–86, Atlantic City, 1988. Elsevier (North-Holland).

[2] S. Boyd and H. Ural. The synchronization problem in protocol testing and its complexity. *Information Processing Letters*, 40:131–136, 1991.

[3] L. Cacciari and O. Rafiq. Controllability and observability in distributed testing. *Information and Software Technology*, 41:767–780, 1999.

[4] S. T. Chanson, B. P. Lee, N. J. Parakh, and H. X. Zeng. Design and implementation of a Ferry Clip test system. In *Protocol Specification, Testing and Verificaion, IX*, pages 101–118. Elsevier (North-Holland), 1990.

[5] W.-H. Chen and H. Ural. Synchronizable test sequences based on multiple UIO sequence. *IEEE/ACM Transactions on Networking*, 3:152–157, 1995.

[6] 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.

[7] 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.

[8] A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, 1985.

[9] R. M. Hierons. Extending test sequence overlap by invertibility. *The Computer Journal*, 39:325–330, 1996.

[10] R. M. Hierons. Testing from a finite state machine: Extending invertibility to sequences. *The Computer Journal*, 40:220–230, 1997.

[11] R. M. Hierons, S. Sadeghipour, and H. Singh. Testing a system specified using Statecharts and Z. *Information and Software Technology*, 43:137–149, 2001.

[12] Hyoung Seok Hong, Young Gon Kim, Sung Deok Cha, Doo Hwan Bae, and Hasan Ural. A test sequence selection method for statecharts. *Journal of Software Testing, Verification and Reliability*, 10, 2000.

[13] Vikram Iyengar and Krishnendu Chakrabarty. An efficient finite-state machine implementation of Huffman decoders. *Information Processing Letters*, 64:271–275, 1998.

[14] J. L. Lenstra and Rinnoy Khan. On general routing problems. *Networks*, 6:273–280, 1976.

[15] G. Luo, A. Das, and G. von Bochmann. Generating tests for control portion of SDL specifications. In *Protocol Test Systems VI*, pages 51–66. Elsevier (North-Holland), 1994.

[16] G. Luo, R. Dssouli, and G. von 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.

[17] E. P. Moore. Gedanken-Experiments. In C. Shannon and J. McCarthy, editors, *Automata Studies*. Princeton University Press, 1956.

[18] A. Petrenko, G. v. Bochmann, and R. Dssouli. Conformance relations and test derivation. In *Proceedings of Protocol Test Systems VI (C-19)*, pages 157–178, 1994.

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

[20] D. P. Sidhu and T.-K. Leung. Formal methods for protocol testing: A detailed study. *IEEE Transactions on Software Engineering*, 15:413–426, 1989.

[21] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 3 edition, 1996.

[22] H. Ural, X. Wu, and F. Zhang. On minimizing the lengths of checking sequences. *IEEE Transactions on Computers*, 46:93–99, 1997.

[23] W.-J. Wu, W.-H. Chen, and C. Y. Tang. Synchronizable test sequence for multi-party protocol conformance testing. *Computer Communications*, 21:1177–1183, 1998.

[24] B. Yang and H. Ural. Protocol conformance test generation using multiple UIO sequences with overlapping. In *ACM SIGCOMM 90: Communications, Architectures, and Protocols*, pages 118–125, Twente, The Netherlands, September 24-27 1990.

[25] 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.