Checking Experiments for Stream X-Machines

Robert M. Hierons

Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK

Abstract

Stream X-machines are a state based formalism that has associated with it a particular development process in which a system is built from trusted components. Testing thus essentially checks that these components have been combined in a correct manner and that the orders in which they can occur are consistent with the specification. Importantly, there are test generation methods that return a checking experiment: a test that is guaranteed to determine correctness as long as the implementation under test (IUT) is functionally equivalent to an unknown element of a given fault domain Ψ. Previous work has show how three methods for generating checking experiments from a finite state machine (FSM) can be adapted to testing from a stream X-machine. However, there are many other methods for generating checking experiments from an FSM and these have a variety of benefits that correspond to different testing scenarios. This paper shows how any method for generating a checking experiment from an FSM can be adapted to generate a checking experiment for testing an implementation against a stream X-machine. This is the case whether we are testing to check that the IUT is functionally equivalent to a specification or we are testing to check that every trace (input/output sequence) of the IUT is also a trace of a nondeterministic specification. Interestingly, this holds even if the fault domain Ψ used is not that traditionally associated with testing from a stream X-machine. The results also apply for both deterministic and nondeterministic implementations.

Keywords: Stream X-machines, test generation, checking experiment

Email address: rob.hierons@brunel.ac.uk (Robert M. Hierons)

1. Introduction

Many classes of system, such as reactive systems and communications protocols, are state based. This has led to interest in languages for describing state based models and methods for testing from such models. While there has been much interest in testing from finite state machines (FSMs) (see, for example, [1, 2, 3, 4]), FSMs do not model internal data and so are not always appropriate. Where it is important to describe the data, modelling languages/approaches such as stream X-machines [5, 6, 7], SDL [8], and statecharts [9], are often used.

A fault domain is a set Φ of models with the property that the tester believes that the *implementation under test (IUT)* behaves like an unknown element of Φ. Fault domains can be used to capture known (or believed) properties of the IUT and allow us to reason about test effectiveness relative to the fault domain. For example, we might wish to prove that a test suite T determines correctness relative to a fault domain Ψ: all faulty elements of Ψ fail T and all other element of Ψ pass T. Such a test suite has been called a checking experiment in the context of FSMs (see, for example, [10, 2, 11]) and in this paper the term checking experiment is used for both FSMs and stream X-machines. Note that sometimes the term fault domain is also used to denote a set of alternative implementations that have been produced by changing the IUT and it is possible for this set to not contain any implementations that conform to the specification. However, we use the term as described above and in contrast to the notion of a fault model, which is a set of models that do not conform to the specification [12].

Stream X-machines are a state based formalism that has associated with it a particular development methodology. In this methodology, the system is built from trusted components that correspond to relations or functions used in the specification. These components are assumed to be correct, potentially as a result of them having been tested in a previous phase based on stream X-machines for the components. Testing can then be seen as checking that these components interact in an appropriate manner [13]. The testing process can therefore been seen as integration testing. The primary interest in stream X-machines has been software development but they have also been used to model biological agents [14, 15, 16] and NASA has discussed using them in the development of swarm satellite systems [17].

There has been interest in the automatic generation of test suites from a stream X-machine (see, for example, [18, 19, 20, 21, 22, 23, 24, 6, 13, 7, 25,

26]). It has been found that if certain restrictions, called *specify for test conditions*, are placed on the stream X-machine specifications then some checking experiment generation techniques for FSMs can be adapted to stream X-machines with corresponding fault domains. The resulting tests are guaranteed to determine correctness as long as the IUT is functionally equivalent to an element of the fault domain. This has been done for the W-method [1, 27], the Wp method [25, 28], and state counting [23, 29]. However, there are many other methods for automatically generating checking experiments from FSMs (see, for example, [1, 10, 2, 30, 3, 4]) and potentially alternative fault domains and it is natural to ask whether any of these can be used in testing from a stream X-machine. This is the problem solved in this paper: we prove that all checking experiment generation algorithms for FSMs can be used when testing an IUT against a stream X-machine.

The results in this paper make two major contributions to the field of automating testing from a stream X-machine. First, we show how to adapt any checking experiment generation algorithm, for FSMs, when testing from a stream X-machine that satisfies the traditional specify for test conditions. The resultant test is guaranteed to determine whether the IUT is correct as long as it is functionally equivalent to an element of the fault domain used. Second, we show that this is valid even if the fault domain is not the one used in previous work on testing from a stream X-machine, which places an upper bound on the number of states of the IUT. The results thus allow testers to utilise a wider range of checking experiment generation algorithms in addition to those previously used and also to consider alternative fault domains. The results are proved for two notions of correctness: for testing to determine whether the IUT and specification are equivalent and for testing to determine whether the traces (input/output sequences) of the IUT are also traces of the specification. The results apply with both deterministic and nondeterministic implementations.

This paper is structured as follows. Section 2 describes the notation used in the paper and provides relevant definitions and results regarding finite automata, finite state machines, and stream X-machines. Section 3 then shows how problems regarding testing an IUT against a stream X-machine can be transformed into problems of testing from finite automata. We then show how problems of testing from finite automata can be transformed into problems of testing against finite state machines, Sections 4 and 5 considering testing for equivalence and inclusion respectively. Section 6 then makes a number of practical observations while Section 7 draws conclusions.

2. Preliminaries

2.1. Basic notation

Throughout this paper we let ϵ denote the empty sequence and the name of a variable has a bar above it (for example $\bar{z}$) if the variable represents a sequence. Given a set X, $\mathcal{P}(X)$ will denote the powerset of X and X^* will denote the set of finite sequences of elements of X. Given a relation f of type $X \leftrightarrow Y$, *dom* f denotes the set of elements of X for which f is defined: $dom\ f = \{x \in X | \exists y \in Y.(x, y) \in f\}$.

This paper proves that there is a correspondence between tests for stream X-machines and tests for FSMs and does so by considering tests for finite automata. In testing against a stream X-machine or finite state machine we apply an input sequence and observe an input/output sequence, called a *trace*. A test suite is a set of input sequences to be used in testing. We use the following notation for test suites: we use T_A and its variants (priming etc.) to denote test suites used with finite automata, T_F and its variants to denote test suites used with FSMs and T_X and its variants to denote test suites used with stream X-machines.

2.2. Finite Automata

A *finite automaton (FA)* N is defined by a tuple $(S, s_0, Z, \delta, \Gamma)$ in which S is a finite set of states, $s_0 \in S$ is the initial state, Z is the finite alphabet, δ is the state transfer relation of type $S \times Z \leftrightarrow S$, and $\Gamma \subseteq S$ is the set of final states. If $s' \in \delta(s, z)$ then (s, s', z) is a *transition* of N with starting state s and ending state s'. A sequence $(s_0, s_1, z_1) \ldots (s_{k-1}, s_k, z_k)$ of consecutive transitions, whose first transition has starting state s_0, is a *path* that has *ending state* s_k and *label* $z_1, \ldots, z_k$.

If N receives $z \in Z$ when in state $s \in S$ it moves to a state in the set $\delta(s, z)$. The state transfer relation δ can be extended to sequences in Z^* in the usual way: $\delta(s, \epsilon) = \{s\}$ and for $z \in Z, \bar{z} \in Z^*$ we have that $\delta(s, \bar{z}z) = \{s' \in S | \exists s'' \in \delta(s, \bar{z}).s' \in \delta(s'', z)\}$. The FA N defines the language $L(N) = \{\bar{z} \in Z^* | \delta(s_0, \bar{z}) \cap \Gamma \neq \emptyset\}$ of sequences that can take it to a final state. Clearly $L(N)$ is the set of labels of paths of N that have ending state in Γ.

FA $N = (S, s_0, Z, \delta, \Gamma)$ is *deterministic* if for all $s \in S$ and $z \in Z$ there is at most one possible next state and so $|\delta(s, z)| \leq 1$. Two FA are *equivalent* if they define the same language. Given FA N, there is an equivalent deterministic FA [31]. A deterministic FA (DFA) is *minimal* if there is no

equivalent DFA with fewer states. Since any FA can be rewritten to an equivalent minimal DFA [3] we only consider minimal DFA in this paper.

In this paper we will use FA as an intermediate step between finite state machines and stream X-machines, which we define later. Since we are interested in distinguishing models we describe here what it means for a sequence from Z^* to distinguish two FA and also what it means for a set of sequences from $\mathcal{P}(Z^*)$ to distinguish between an FA and a set of FA. The context will be one FA N that represents the specification and another FA N' that represents the implementation and depending on the testing context we will require either that $L(N') = L(N)$ or $L(N') \subseteq L(N)$.

Definition 1. *Let N and N' be FA and Ψ a set of FA. A sequence $\bar{z} \in Z^*$* distinguishes *N from N'* under equivalence *if either $\bar{z} \in L(N) \setminus L(N')$ or $\bar{z} \in L(N') \setminus L(N)$. A set $T_A \in \mathcal{P}(Z^*)$* distinguishes *N from Ψ* under equivalence *if for all $N' \in \Psi$ such that $L(N') \neq L(N)$ there exists $\bar{z} \in T_A$ such that $\bar{z}$ distinguishes N from N' under equivalence.*

Definition 2. *Let N and N' be FA and Ψ a set of FA. A sequence $\bar{z} \in Z^*$* distinguishes *N from N'* under inclusion *if $\bar{z} \in L(N') \setminus L(N)$. A set $T_A \in \mathcal{P}(Z^*)$* distinguishes *N from Ψ* under inclusion *if for all $N' \in \Psi$ such that $L(N') \not\subseteq L(N)$ there exists $\bar{z} \in T_A$ such that $\bar{z}$ distinguishes N from N' under inclusion.*

In practice, the set Ψ of FA will be a *fault domain*: a set of models such that the tester believes that the implementation is equivalent to an unknown element of Ψ. Normally the set Ψ is allowed to contain models that correspond to correct implementations and this is why we allow a set $T \in \mathcal{P}(Z^*)$ to distinguish N from Ψ even if some elements of Ψ are correct implementations. We will discuss fault domains further when describing finite state machines and stream X-machines.

2.3. Finite State Machines

While finite automata are highly appropriate for defining languages, they do not distinguish between input and output. Thus, when defining a reactive system, which responds to input by providing output, it is more usual to use a finite state machine. A (completely specified) *finite state machine (FSM)* R is defined by a tuple (S, s_0, X, Y, δ) in which S is a finite set of states, $s_0 \in S$ is the initial state, X is the finite input alphabet, Y is the finite

output alphabet, and δ is the state transfer relation of type $S \times X \leftrightarrow S \times Y$ such that for all $s \in S$ and $x \in X$, $\delta(s,x) \neq \emptyset$. In this paper we only consider completely specified FSMs.

If R receives input x when in state s it produces an output y and moves to a state s' such that $(s',y) \in \delta(s,x)$ and this defines a transition $(s,s',x/y)$. A *path* of R is a sequence $(s_0,s_1,x_1/y_1),\ldots,(s_{k-1},s_k,x_k/y_k)$ of consecutive transitions that starts at the initial state of R. Such a path has *label* $x_1/y_1,\ldots,x_k/y_k$ and *ending state* s_k.

As with FA, the relation δ can be extended to input sequences: $\delta(s,\epsilon) = \{(s,\epsilon)\}$ and for all $s \in S$, $x \in X$ and $\bar{x} \in X^*$ we have that $\delta(s,\bar{x}x) = \{(s',\bar{y}y) | \exists s'' \in S.(s'',\bar{y}) \in \delta(s,\bar{x}) \wedge (s',y) \in \delta(s'',x)\}$. FSM R also defines a language: the set of input/output sequences that can occur from the initial state. More formally, given FSM $R = (S,s_0,X,Y,\delta)$ we have that $L(R) = \{\bar{x}/\bar{y} | \exists s \in S.(s,\bar{y}) \in \delta(s,\bar{x})\}$. Given an input sequence $\bar{x}$ we let $R(\bar{x}) = \{\bar{x}'/\bar{y}' \in L(R) | \bar{x} = \bar{x}'\}$.

In testing from an FSM we compare the observed behaviour of the IUT with that of the specification. There are two standard notions of an implementation FSM R' being correct relative to a specification FSM R: either we require the implementation to be *equivalent* to the specification ($L(R') = L(R)$) or that every behaviour of the implementation is also a behaviour of the specification ($L(R') \subseteq L(R)$) and this latter notion of correctness is typically called *conformance*. These two different notions of correctness again lead to two notions of distinguishing FSMs.

Definition 3. *Given FSMs R and R' with the same input alphabet, a sequence $\bar{x} \in X^*$* distinguishes *R from R' under equivalence if $R(\bar{x}) \neq R'(\bar{x})$. A set $T_F \in \mathcal{P}(X^*)$* distinguishes *R from a set Ψ of FSMs under equivalence if for all $R' \in \Psi$ either $L(R) = L(R')$ or there exists $\bar{x} \in T_F$ such that $\bar{x}$ distinguishes R from R' under equivalence. The set T_F is then said to be a* checking experiment for *R with Ψ under equivalence.*

Definition 4. *Given specification FSM R and FSM R' with the same input alphabet, a sequence $\bar{x} \in X^*$* distinguishes *R from R' under inclusion if $R'(\bar{x}) \not\subseteq R(\bar{x})$. A set $T_F \in \mathcal{P}(X^*)$* distinguishes *R from a set Ψ of FSMs under inclusion if for all $R' \in \Psi$ either $L(R') \subseteq L(R)$ or there exists $\bar{x} \in T_F$ such that $\bar{x}$ distinguishes R from R' under inclusion. The set T_F is then said to be a* checking experiment for *R with Ψ under inclusion.*

The set Ψ used is usually a *fault domain* that describes the types of faults that the tester believes can happen: the tester believes that the IUT is equivalent to an element of Ψ. Most work on testing from FSMs either uses the fault domain where the number of states of the FSM that represents the IUT is no greater than the number of states of the specification FSM (see, for example, [10, 2, 32, 11]) or places an upper bound on the number of states of the FSM that represents the IUT (see, for example, [1, 30, 28]). However, in principle any fault domain can be used.

2.4. Stream X-Machines

There has been much interest in testing from FSMs since they can be used to model many state based systems. However, they do not model data and guards on transitions (state transfers) and so there has also been interest in more expressive types of models such as stream X-machines. In this section we give standard definitions of stream X-machines and associated notation (see, for example, [23]). A stream X-machine has a set of states, there are transitions between states that are labelled with relations, and there is an internal memory. Formally, a stream X-machine is defined by a tuple $(In, Out, S, Mem, \Phi, F, s_0, m_0, \Gamma)$ [7] in which:

- In is the input alphabet.
- Out is the output alphabet.
- S is the finite set of states.
- Mem is the memory. Mem need not be finite.
- Φ is the finite set of processing relations, of type $Mem \times In \leftrightarrow Out \times Mem$.
- F is the next state relation of type $S \times \Phi \leftrightarrow S$.
- $s_0 \in S$ is the initial state.
- $m_0 \in Mem$ is the initial memory.
- Γ is the set of final states.

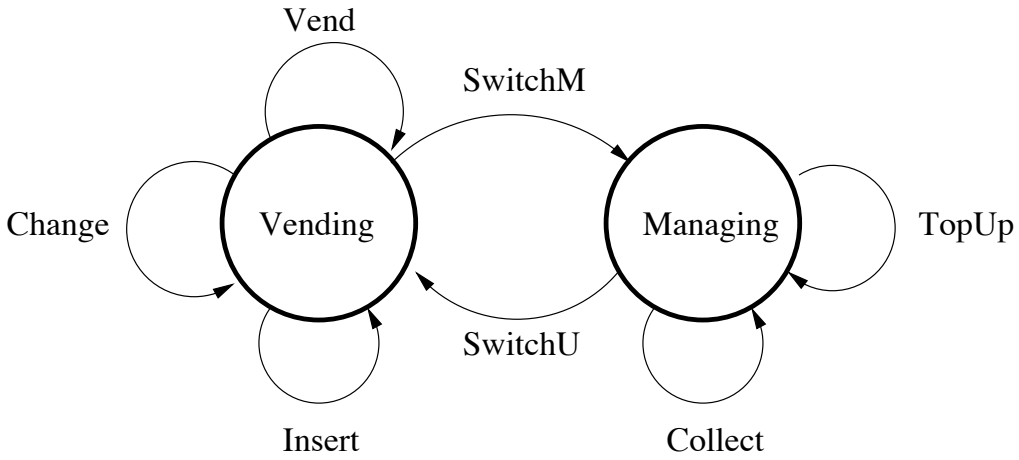

Figure 1: The Vending Stream X-Machine M_V

Consider, for example, the stream X-machine M_V for a simply vending machine shown in Figure 1; this was originally described in [22]. As stated in [22], this stream X-machine operates in the following way. M_V has BUTTONS that are input devices and Lights provide output. The $\mathcal{V}$ button requests the vending of a chocolate: if sufficient payment has been received then this occurs, the Choc light is activated and the current balance is updated. Otherwise the NoVend light is activated. The chocolate costs 20. The other operation that the customer sees is the function that allows the customer to insert money: this is triggered by the input of a coin. The input is represented by the coin value and a label that represents the coin being input at slot UserIn. There are two coin values: 10 and 20. The machine has an LCD display which displays the amount of credit the machine possesses when the user inserts coins. There is a $\mathcal{C}$ button, which requests change (the current balance) to be returned. This simply returns the change to the customer through a particular slot. This relation is nondeterministic since there may be several alternative choices regarding the coins returned: it is sufficient that these coins are in the machine and that their values sum to the correct value. The memory is a tuple that specifies how many of each type of coin is currently in the machine and the current balance. A memory value of (x, y, z) represents there being x coins of value 10, y coins of value 20, and a current balance of z.

There is a special set of operations for the manager. The $\mathcal{M}$ button, which is key operated, triggers the operation SwitchM that moves the system into manager mode. In this mode the $\mathcal{E}$ button can be used to empty the machine of change and the $\mathcal{U}$ button is used to leave management mode and return to user mode. Light ManageOn shows that the machine is switching to management mode and ManageOff shows that the machine is switching back to user mode.

The TopUp function allows the manager to add coins to the machine, at a separate slot to the one used by customers. The input is therefore a coin value and a label indicating the slot ManagerIn is used and the output is a message to the screen that shows the coins currently in the machine.

The complete Stream X-machine is given in [22] and so here we indicate how this can be defined. First, the memory is the set of triples of integers. As noted above, memory value (x, y, z) represents there being x coins of value 10, y coins of value 20, and a current balance of z. Second, there are two states Vending and Managing, the initial state is Vending, and the next state relation is defined by the arcs shown in Figure 1. The processing relations operate as described above. For example, the Insert relation takes as input a coin value (10 or 20) and a label that represents the coin being inserted at slot UserIn. If the memory before this operation was (x, y, z) then there are two cases: if the coin value was 10 then the memory becomes $(x + 1, y, z + 10)$ and the output is $z + 10$ while if the coin value was 20 then the memory becomes $(x, y + 1, z + 20)$ and the output is $z + 20$.

Since stream X-machines are typically used to model reactive systems it is normal to assume that all states are final states ($\Gamma = S$) and we make this assumption throughout this paper. We can abstract away the relations on the transitions of a stream X-machine to define a FA called the associated automaton.

Definition 5. *Given stream X-machine* $M = (In, Out, S, Mem, \Phi, F, s_0, m_0, \Gamma)$, *the* associated automaton $A(M)$ *is the FA* $(S, s_0, \Phi, F, \Gamma)$.

In analysing a stream X-machine M we will need to reason about paths of M and the relations defined by these and we introduce notation in order to assist with this. Since we only consider deterministic, minimal FA in this paper, for any stream X-machine M considered we make the normal assumption that the FA $A(M)$ is minimal and deterministic and thus that F

is a function[1]. Note that M can still be nondeterministic since a relation in Φ need not be a function and there can be a state s and relations f and f' with overlapping input domains such that $(s, f) \in dom\ F$ and $(s, f') \in dom\ F$. We now give definitions based on those in [23].

A sequence $\bar{\sigma}$ of elements from Φ defines a relation $\|\bar{\sigma}\|$ of type $Mem \times In^* \leftrightarrow Out^* \times Mem$ corresponding to the possible results of executing the relations from $\bar{\sigma}$ in the given order.

Definition 6. *Given $\bar{\sigma} \in \Phi^*$, $\|\bar{\sigma}\|$ of type $Mem \times In^* \leftrightarrow Out^* \times Mem$, is defined by the following in which $f \in \Phi$ and $\bar{\sigma}' \in \Phi^*$.*

$$\|\epsilon\| = \{((m, \epsilon), (\epsilon, m)) | m \in Mem\}$$

$$\begin{aligned} \|\bar{\sigma} f\| \quad = \quad & \{((m, \bar{x}x), (\bar{y}y, m')) | \exists m'' \in Mem.((m, \bar{x}), (\bar{y}, m'')) \in \|\bar{\sigma}\| \\ & \wedge ((m'', x), (y, m')) \in f\} \end{aligned}$$

A stream X-machine starts with memory m_0 and so a sequence $\bar{\sigma}$ of processing relations defines a relation $\langle \bar{\sigma} \rangle$ formed by restricting $\|\bar{\sigma}\|$ to the case where the initial memory is m_0.

Definition 7. *Given sequence $\bar{\sigma}$ of processing relations, $\langle \bar{\sigma} \rangle$ is the relation of type $In^* \leftrightarrow Out^*$ defined in the following way:*

$$\langle \bar{\sigma} \rangle = \{(\bar{x}, \bar{y}) | \exists m \in Mem.((m_0, \bar{x}), (\bar{y}, m)) \in \|\bar{\sigma}\|\}$$

Consider, for example, M_V and the sequence (Insert Vend) that involves inserting a coin and then pressing vend. There are two possible inputs for the first operation: either input of 10 at UserIn or input of 20 at UserIn. As a result $\|$ Insert Vend $\|$ is the set $\{((0,0,0), (10, \text{UserIn})\ \mathcal{V}, 10\ \mathsf{NoVend}, (1,0,10)), ((0,0,0), (20, \text{UserIn})\ \mathcal{V}, 20\ \mathsf{Choc}, (0,0,0))\}$. Further, $\langle$ Insert Vend $\rangle$ is the set $\{((10, \text{UserIn})\ \mathcal{V}, 10\ \mathsf{NoVend}), ((20, \text{UserIn})\ \mathcal{V}, 20\ \mathsf{Choc})\}$.

Since M defines a set of paths, those that lead to final states, and each path defines a relation of type $In^* \leftrightarrow Out^*$, M defines a relation $\lfloor M \rfloor$, of type $In^* \leftrightarrow Out^*$.

[1]This is not a significant restriction since any FA can be converted to an equivalent minimal deterministic FA.

Definition 8. *Given Stream X-machine M, $\lfloor M \rfloor$ is the relation defined by the following:*

$$\lfloor M \rfloor = \bigcup_{\bar{\sigma} \in L(A(M))} \langle \bar{\sigma} \rangle$$

Given a stream X-machine M and input sequence $\bar{x}$, $\lfloor M \rfloor(\bar{x})$ thus denotes the set of output sequences that M can produce in response to $\bar{x}$. The stream X-machine M has an input domain: that of $\lfloor M \rfloor$. Given a stream X-machine M, the *input domain* of M, *dom* M, is defined by: $dom\ M = \{\overline{x} \in In^* | \exists \overline{y}.\overline{y} \in Out^* \wedge (\overline{x}, \overline{y}) \in \lfloor M \rfloor\}$. Stream X-machine M is *completely specified* if $dom\ M = In^*$.

In this paper we only consider completely specified stream X-machines, however, if a stream X-machine M is not completely specified then it is possible to complete M by adding an error state (see, for example, [23]).

2.5. Conformance relations

It is necessary to say what we mean by an implementation conforming to a specification stream X-machine and again we use two notions of conformance. One is that the IUT and specification are *equivalent* and the other is that every behaviour (input/output sequence) of the implementation is also a behaviour of the specification. We call the latter conformance under inclusion in order to distinguish it from conformance under equivalence.

Definition 9. *Given completely specified stream X-machines M and M' with the same input and output alphabets, M'* conforms to *M* under equivalence *if and only if $\lfloor M' \rfloor = \lfloor M \rfloor$. Further, an input sequence $\bar{x}$* distinguishes *M* from *M'* under equivalence *if $\lfloor M' \rfloor(\bar{x}) \neq \lfloor M \rfloor(\bar{x})$.*

Definition 10. *Given completely specified stream X-machines M and M' with the same input and output alphabets, M'* conforms to *M* under inclusion *if and only if $\lfloor M' \rfloor \subseteq \lfloor M \rfloor$. Further, an input sequence $\bar{x}$* distinguishes *M* from *M'* under inclusion *if $\lfloor M' \rfloor(\bar{x}) \not\subseteq \lfloor M \rfloor(\bar{x})$.*

We can extend the notion of a checking experiment for an FSM to a checking experiment for a stream X-machine.

Definition 11. *Let M be a stream X-machine and let Ψ be a set of stream X-machines with the same input and output alphabets that denotes a fault domain. Then a set T_X of input sequences is a* checking experiment for *M* given Ψ under equivalence *if for all $M' \in \Psi$ with $\lfloor M' \rfloor \neq \lfloor M \rfloor$ there exists some $\bar{x} \in T_X$ such that $\lfloor M' \rfloor(\bar{x}) \neq \lfloor M \rfloor(\bar{x})$.*

Definition 12. *Let M be a stream X-machine and let Ψ be a set of stream X-machines with the same input and output alphabets that denotes a fault domain. Then a set T_X of input sequences is a* checking experiment for M given Ψ under inclusion *if for all $M' \in \Psi$ with $\lfloor M' \rfloor \not\subseteq \lfloor M \rfloor$ there exists some $\bar{x} \in T_X$ such that $\lfloor M' \rfloor(\bar{x}) \not\subseteq \lfloor M \rfloor(\bar{x})$.*

2.6. Specify for test conditions and test hypotheses

The work on testing from stream X-machines traditionally places two restrictions on the stream X-machines considered: the *specify for test conditions* place restrictions on the specification stream X-machine while the *test hypotheses* place restrictions on the stream X-machine that models the IUT. Originally these were all grouped together under the name *design for test conditions*. The idea is that the specify for test conditions can be 'designed into' a system in order to assist testing while the test hypotheses represent beliefs about the IUT that allow us to restrict the set of models that could represent the IUT and to reason about test effectiveness. The notion of test hypothesis here is similar to that used in other areas of testing from a formal specification [33, 34, 35, 36] and relates strongly to the concept of a fault domain. Naturally, the test hypotheses are much less restrictive than the specify for test conditions. We first describe the traditional specify for test conditions for testing from a stream X-machine (see, for example [22, 7, 37, 25]).

The set Φ of processing relations in M is output distinguishable if from observing an input/output pair from a known memory we can determine which relation was applied. This condition holds if, given any two different $f_1, f_2 \in \Phi$, memory value $m \in \mathcal{M}$, and input $x \in I$, the two relations cannot lead to the same output value in response to x when the memory is m. This property allows the tester to determine which relation from Φ has been executed based on observed input/output behaviour (see, for example, [22, 7]).

Definition 13. *Φ is* output distinguishable *if for all $f_1, f_2 \in \Phi$ with $f_1 \neq f_2$, all $x \in In$, $y \in Out$, and $m, m' \in \mathcal{M}$ such that $((m, x), (y, m')) \in f_1$, there does not exist $m'' \in \mathcal{M}$ such that $((m, x), (y, m'')) \in f_2$.*

Consider the stream X-machine M_V. Its set of processing relations is output distinguishable since the different processing relations send output to different devices such as lights, the screen, and the slots that output coins.

The set Φ of processing relations in M is observable if for every $f \in \Phi$, from the memory value before f is applied, the input used and the output produced we can determine the new memory value after f has been applied. This allows the tester to determine the expected memory based on the input and the output observed (as long as Φ is output distinguishable) [22]. If this property does not hold then it is difficult for the tester to determine the next input to be applied in order to trigger a given relation f' since this can depend on the unknown memory value.

Definition 14. Φ *is* observable *if for all* $f \in \Phi, x \in In, m \in \mathcal{M}$ *we have that*

$$(y_1, m_1), (y_2, m_2) \in f(m, x) \Rightarrow ((y_1 = y_2) \Rightarrow (m_1 = m_2)).$$

Consider again M_V. All but one of the processing relations are deterministic and so are trivially observable. The exception is the button that leads to change being returned. This is observable since the memory after the operation is fully defined by the coins in the machine (specified by the memory) before the operation and the coins output.

The set Φ of processing relations in M is complete if for each $f \in \Phi$ and memory m, the tester can always apply an input that is capable of triggering f. Naturally, this does not require there to be a transition with label f from every state, just that if there is such a transition then we can always choose an input to trigger it.

Definition 15. Φ *is* complete *if for all* $m \in \mathcal{M}$ *and* $f \in \Phi$ *there exists* $x \in In$ *such that* $(m, x) \in dom\ f$.

It is straightforward to show that the set of relations of M_V is complete. The following are the specify for test conditions.

Definition 16. *Stream X-machine* $M = (In, Out, S, Mem, \Phi, F, s_0, m_0, S)$ *has the* specify for test conditions *if the following hold:*

1. Φ *is output distinguishable;*
2. Φ *is observable; and*
3. Φ *is complete.*

While these specify for test conditions place restrictions on the stream X-machine M used to specify or design the required behaviour, a stream X-machine that does not satisfy these conditions can be rewritten to one that does [7]. This process of rewriting M can involve adding new inputs and outputs but these could either be removed or hidden in the final system.

Some work has looked at weakening the specify for test conditions [24, 25, 26] but has focussed on the use of particular FSM test generation techniques. It would be interesting to generalise the results in this paper to weaker specify for test conditions but this is a problem for future work.

We now describe the test hypotheses, which assume that the IUT I behaves like an unknown stream X-machine $M_I = (In, Out, S', Mem, \Phi', F', s'_0, m_0, S')$. The approaches to testing from stream X-machines essentially assume that the IUT has been built out of 'correct' components, possibly as a result of the IUT having been built out of components that are known to be correct or through these components having previously been tested. As a result faults can only occur through an incorrect state transition structure [7]. If we are testing for equivalence then this assumption that the IUT is built out of trusted components corresponds to saying that M and M_I have the same sets of processing relations ($\Phi' = \Phi$). When testing for conformance, we make the weaker assumption that each element of the set Φ' of relations of M_I conforms to a relation in M [23].

Definition 17. *Given $f' \in \Phi'$ and $f \in \Phi$, we say that f'* conforms to *f, written $f' \leq f$, if dom f' = dom f and $f' \subseteq f$. Further, we write $\Phi' \leq \Phi$ if for all $f' \in \Phi'$ there exists $f \in \Phi$ such that $f' \leq f$.*

It is possible to extend $\leq$ to take sequences of relations in the natural way [22].

Based on Φ being output distinguishable, it is straightforward to show[2] that under the specify for test conditions, for every $f' \in \Phi'$ there is exactly one $f \in \Phi$ such that $f' \leq f$ and the relation f will be denoted $abs_\Phi(f')$. Reasoning about $A(M_I)$ and $A(M)$ is simplified if we use the same alphabet Φ rather than separate alphabets Φ and Φ' and this leads to the following definition of the abstraction $Abs_\Phi(M_I)$ of M_I produced by replacing each relation f' in $A(M_I)$ by the unique relation $f \in \Phi$ such that $f' \leq f$.

[2]This is proved in [23].

Definition 18. *Given stream X-machine* $M_I = (In, Out, S', Mem, \Phi', F', s'_0, m_0, S')$ *and relation set* Φ *such that* $\Phi' \leq \Phi$, $Abs_\Phi(M_I)$ *is the automaton* $(S', s'_0, \Phi, F'', S')$ *such that* F'' *is defined by the following.*

$$F'' = \{((s'_i, abs_\Phi(f')), s'_j) | ((s'_i, f'), s'_j) \in F'\}$$

The following result, which is Proposition 2 in [23], shows that if $\Phi' \leq \Phi$ and Φ satisfies the specify for test conditions then Φ' also satisfies some of these conditions.

Proposition 1. *Let us suppose that stream X-machine* $M = (In, Out, S, Mem, \Phi, F, s_0, m_0, S)$ *satisfies the specify for test conditions. If* $\Phi' \leq \Phi$ *then* Φ' *is complete and observable.*

It is now possible to formally state the test hypotheses for the case where we have a fault domain Ψ.

Definition 19. *Let* $M = (In, Out, S, Mem, \Phi, F, s_0, m_0, S)$ *be a stream X-machine, I the IUT, and* Ψ *the fault domain. The IUT is said to satisfy the* test hypotheses *if the following hold:*

1. *I behaves like an unknown stream X-machine* $M_I = (In, Out, S', Mem, \Phi', F', s'_0, m_0, S')$ *such that* $A(M_I)$ *is deterministic and minimal;*
2. $\Phi' \leq \Phi$*;*
3. $M_I \in \Psi$.

The first two test hypotheses correspond to the IUT being built out of correct components while the last one says that the IUT is equivalent to a member of the given fault domain. Note that previous work on testing from a stream X-machine has considered one particular fault domain, in which there is a known upper bound on the number of states of M_I. However, in this paper we show that it is possible to use any fault domain Ψ: we can utilise checking experiment generation algorithms for the corresponding FSM problem where such algorithms exist.

If the IUT is nondeterministic then we have the usual problem that we have to repeat tests in order to observe the alternative responses of the IUT to these tests. If the nondeterminism is due to concurrency then we might be able to use methods such as deterministic testing [38, 39] or reachability testing [40] in order to ensure that all relevant interleavings are tested. This

can be generalised by making an assumption, called the *complete testing assumption* (see, for example, [41]), that it is sufficient to repeat a test k times for some prior k. Naturally, the value of k used may increase for longer test sequences and this may favour checking experiments that consist of many short sequences rather than a few long sequences. Throughout this paper we make the complete testing assumption. Note that this assumption holds trivially in one important case: when the IUT is deterministic.

2.7. Test functions

In the literature on testing from a stream X-machine M it is normal to identify a set of sequences in $L(M)$ and to then test to determine which of these is implemented in the IUT. The testing problem then reduces to finding an appropriate set of sequences from Φ^*. An implementation M' is said to *implement* $\bar{\sigma}$ if $\bar{\sigma} \in L(Abs_\Phi(M'))$. A test function is used in order to test whether the IUT implements a sequence $\bar{\sigma} \in \Phi^*$.

The test function has as input an element $\bar{\sigma}$ of Φ^* and returns an input sequence $\bar{x}$. If $\bar{\sigma} \in L(M)$ then the input sequence $\bar{x}$ returned is in the input domain of $\bar{\sigma}$. Otherwise, $\bar{x}$ has the property that if $\bar{\sigma}'$ is the longest prefix of $\bar{\sigma}$ that is not in $L(M)$ then $\bar{x}$ is in the input domain of $\bar{\sigma}'$. In the first case, by applying $\bar{x}$ to the IUT we are checking that a sequence that should be implemented is actually implemented. In the second case, we are checking that a sequence that should not be implemented indeed is not.

The following definition is based on one in [7].

Definition 20. *A test function for a stream X-machine M is a function t of type $\Phi^* \rightarrow In^*$ that satisfies the following conditions:*

1. $t(\epsilon) = \epsilon$.
2. *Let us suppose that $\bar{\sigma}' \in L(M)$, $t(\bar{\sigma}') = \bar{x}_1$, and there exists $\bar{y}_1 \in Out^*$ and $m' \in Mem$ such that $((m_0, \bar{x}_1), (\bar{y}_1, m')) \in \|\bar{\sigma}'\|$ and $\bar{x}_1/\bar{y}_1$ is a behaviour of the IUT. Then choose some such $\bar{y}_1$ and m' and let $t(\bar{\sigma}' f) = \bar{x}_1 x$ for some $x \in In$ such that $(m', x) \in dom\ f$.*
3. *Let us suppose that $\bar{\sigma}' \in L(M)$, $t(\bar{\sigma}') = \bar{x}_1$, and there exists $\bar{y}_1 \in Out^*$ such that $\bar{x}_1/\bar{y}_1 \notin \lfloor M \rfloor$ is a behaviour of the IUT. Then $t(\bar{\sigma}' f) = \bar{x}_1$.*
4. *Let us suppose that $\bar{\sigma}' \notin L(M)$ and $t(\bar{\sigma}') = \bar{x}_1$. Then $t(\bar{\sigma}' f) = \bar{x}_1$.*

We can make the following observations regarding these rules.

1. The first rule is the base case since testing terminates when there are no more relations in the sequence $\bar{\sigma}$ being considered.
2. The second rule is the recursive case where the sequence $\bar{\sigma}'$ is contained in $L(M)$ and so we are looking for an input that can follow $\bar{x}_1$ and trigger f. The test function chooses some such input. The memory after $\bar{x}_1/\bar{y}_1$ is known because Φ is observable.
3. The third rule is the recursive case where the sequence $\bar{\sigma}'$ is contained in $L(M)$ but we have already observed a failure $\bar{x}_1/\bar{y}_1$ in applying the test function and so there is no need to continue.
4. In the last (recursive) rule it is sufficient to determine whether $\bar{\sigma}'$ has been implemented and we can then stop testing.

Observe that the IUT is an implicit parameter of the test function.

Consider the stream X-machine M_V and a test function t for this. If we were to apply t to the sequence $\bar{\sigma} =$ Insert Insert Vend then since this sequence is in $L(A(M_V))$ the test function could return any input sequence in the input domain of $\bar{\sigma}$. For example, it could return input sequence $(10, \textsc{UserIn})(20, \textsc{UserIn})\mathcal{V}$. In contrast, if we apply t using the sequence $\bar{\sigma}' =$ Vend TopUp TopUp then we find that the shortest prefix of this that is not in $L(A(M_V))$ is the sequence $\bar{\sigma}'_1 =$ Vend TopUp and so the test function returns an input sequence such as $\mathcal{V}(10, \textsc{ManagerIn})$ in the input domain of $\bar{\sigma}'_1$.

There are many possible test functions for a given stream X-machine and we assume that some such function has been defined for the specification M. It is possible to generalise the notion of a test function to a test process, which is adaptive, and in practice this will make testing more efficient (see, for example, [23]). However, the use of a test function simplifies the description and the results in this paper do not depend on whether we use a test process or a test function.

Recall that we make the complete testing assumption, which is that in testing an input sequence will be applied sufficiently often to observe all possible output sequences. We thus introduce the following notation to represent the set of input/output sequences that can be observed when applying a test function.

Definition 21. *Given a stream X-machines M_1, a test function t of type $\Phi^* \to In^*$ and a sequence $\bar{\sigma} \in \Phi^*$ such that $t(\bar{\sigma}) = \bar{x}$ we let the set of test*

runs of M_1 with t and $\bar{\sigma}$ be:

$$\mathcal{R}(t, \bar{\sigma}, M_1) = \lfloor M_1 \rfloor(\bar{x})$$

Here $\mathcal{R}(t, \bar{\sigma}, M_1)$ is exactly the set of input/output sequences that can occur when we are applying $\bar{x} = t(\bar{\sigma})$ to M_1, which could be a stream X-machine that models the actual behaviour of the IUT. Since we make the complete testing assumption, we will assume that all of these will be observed in testing if we are testing an IUT equivalent to M_1 with t and $\bar{\sigma}$. Recall that the complete testing assumption always holds in the important case where the IUT is deterministic.

We now prove a result that shows that under the specify for test conditions two sequences of relations can have a common input/output sequence if and only if they are the same sequences.

Proposition 2. *Let us suppose that Φ is a observable, output-distinguishable and complete set of processing relations and $\bar{\sigma}$ and $\bar{\sigma}'$ are in Φ^*. Then $\langle \bar{\sigma} \rangle \cap \langle \bar{\sigma}' \rangle \neq \emptyset$ if and only if $\bar{\sigma} = \bar{\sigma}'$.*

Proof

First assume that $\bar{\sigma} = \bar{\sigma}'$. Since Φ is complete we have that $\bar{\sigma}$ and $\bar{\sigma}'$ are feasible and so $\langle \bar{\sigma} \rangle \neq \emptyset$ and $\langle \bar{\sigma}' \rangle \neq \emptyset$. Thus, since $\bar{\sigma} = \bar{\sigma}'$, we have that $\langle \bar{\sigma} \rangle \cap \langle \bar{\sigma}' \rangle \neq \emptyset$ as required.

Now assume that $\langle \bar{\sigma} \rangle \cap \langle \bar{\sigma}' \rangle \neq \emptyset$. If $\bar{\sigma}$ and $\bar{\sigma}'$ have different lengths then we immediately have that $\langle \bar{\sigma} \rangle \cap \langle \bar{\sigma}' \rangle = \emptyset$ and so we can assume that they have the same length.

We will use proof by induction on the length of $\bar{\sigma}$. The base case, of sequences of length 0, follows immediately. Inductive hypothesis: the result holds for all sequences of length less than k and we let $\bar{\sigma} = f_1, \ldots, f_k$ and $\bar{\sigma}' = f'_1, \ldots, f'_k$ denote sequences of length k. Let $\bar{\sigma}_1 = f_1, \ldots, f_{k-1}$ and $\bar{\sigma}'_1 = f'_1, \ldots, f'_{k-1}$. Clearly if $\langle \bar{\sigma}_1 \rangle \cap \langle \bar{\sigma}'_1 \rangle = \emptyset$ then $\langle \bar{\sigma} \rangle \cap \langle \bar{\sigma}' \rangle = \emptyset$ and so we only need to consider the case where $\langle \bar{\sigma}_1 \rangle \cap \langle \bar{\sigma}'_1 \rangle \neq \emptyset$. By the inductive hypothesis we know that $\bar{\sigma}_1 = \bar{\sigma}'_1$. Now consider some input/output sequence $\bar{x}_1 x / \bar{y}_1 y \in \langle \bar{\sigma}_1 f_k \rangle \cap \langle \bar{\sigma}_1 f'_k \rangle$, $x \in In$, $y \in Out$. Since Φ is observable we must have that the memory value m after input/output $\bar{x}_1/\bar{y}_1$ when applying $\bar{\sigma}_1$ is uniquely defined. Further, both f_k and f'_k can produce input/output x/y from memory m and so, since Φ is output-distinguishable, we have that $f'_k = f_k$. The result thus follows. □

The following result, which generalises Lemmas 9 and 10 of [23] (for test processes) to the case where the IUT can be nondeterministic, is the crucial property of a test function.

Proposition 3. *Let us suppose that* $M = (In, Out, S, Mem, \Phi, F, s_0, m_0, S)$ *is a specification that satisfies the specify for test conditions,* $M_1 = (In, Out, S', Mem, \Phi', F', s'_0, m_0, S')$ *is a stream X-machine and* $\Phi' \leq \Phi$. *Further, let us suppose that* t *is a test function and* $\bar{\sigma} \in \Phi^*$ *such that either* $\bar{\sigma} \in L(M)$ *or* $\bar{\sigma} = \bar{\sigma}' f$ *for some* $\bar{\sigma}' \in L(M)$ *and* $f \in \Phi$. *We have that* $\mathcal{R}(t, \bar{\sigma}, M_1) \cap \langle \bar{\sigma} \rangle \neq \emptyset$ *if and only if* $\bar{\sigma} \in L(Abs_\Phi(M_1))$.

Proof
First assume that we have that $\mathcal{R}(t, \bar{\sigma}, M_1) \cap \langle \bar{\sigma} \rangle \neq \emptyset$ and so we are required to prove that $\bar{\sigma} \in L(Abs_\Phi(M_1))$.

Assume that $(\bar{x}, \bar{y}) \in \mathcal{R}(t, \bar{\sigma}, M_1) \cap \langle \bar{\sigma} \rangle$. Thus, $(\bar{x}, \bar{y}) \in \lfloor M_1 \rfloor$ and so $(\bar{x}, \bar{y}) \in \langle \bar{\sigma}_1 \rangle$ for some $\bar{\sigma}_1 \in L(Abs_\Phi(M_1))$. Thus, $\langle \bar{\sigma} \rangle \cap \langle \bar{\sigma}_1 \rangle \neq \emptyset$ and so, by Proposition 2, we have that $\bar{\sigma} = \bar{\sigma}_1$ as required.

Now assume that $\bar{\sigma} \in L(Abs_\Phi(M_1))$ and so we need to prove that $\mathcal{R}(t, \bar{\sigma}, M_1) \cap \langle \bar{\sigma} \rangle \neq \emptyset$. This follows immediately from the definition of the test function since we must have that $t(\bar{\sigma})$ is an input sequence $\bar{x}$ such that $\bar{x} \in dom\ \langle \bar{\sigma} \rangle$. □

This essentially says that if we apply a test function t to the M_1 with $\bar{\sigma} \in \Phi^*$ and one possible resultant input/output sequence is consistent with $\bar{\sigma}$ ($(\bar{x}, \bar{y}) \in \langle \bar{\sigma} \rangle$) then $L(Abs_\Phi(M_1))$ must contain $\bar{\sigma}$ and so $\bar{\sigma}$ must have been implemented.

3. Transforming Stream X-machine problems into FA problems

We have already seen that given a stream X-machine M there is a corresponding FA $A(M)$ produced by abstracting out the memory, input, and output. We also assume that the IUT behaves like an unknown stream X-machine M_I in a fault domain Ψ and thus that the testing problem is one of deciding whether M_I is one of the elements of Ψ that conform to M. If we let $A(\Psi) = \{A(M') | M' \in \Psi\}$ be the set of abstractions of elements of the fault domain then it seems natural to ask whether the problem of deciding whether M_I conforms to M is equivalent to a problem of deciding whether $A(M_I)$ is related to $A(M)$ in some way and whether this can be done for both testing for equivalence and testing for inclusion. In this section we prove that

there is such a correspondence; results in Sections 4 and 5 then show that each FA problem, under equivalence and inclusion, corresponds to an FSM problem that can be solved using checking experiments.

We now give results regarding how $L(A(M))$ and $L(A(M'))$ must relate for stream X-machines M' and M such that either $\lfloor M' \rfloor = \lfloor M \rfloor$ or $\lfloor M' \rfloor \subseteq \lfloor M \rfloor$. These will be used in the proofs of the main results in this section.

Proposition 4. *Let us suppose that M and M' are stream X-machines with sets Φ and Φ' of processing relations with $\Phi' \leq \Phi$ and let $N = A(M)$ and $N' = Abs_{\Phi}(M')$. Then $\lfloor M' \rfloor \subseteq \lfloor M \rfloor$ if and only if $L(N') \subseteq L(N)$.*

Proof
First let us suppose that $\lfloor M' \rfloor \subseteq \lfloor M \rfloor$. We need to prove that $L(N') \subseteq L(N)$ and so it is sufficient to prove that for all $\bar{\sigma}' \in L(N')$ we have that $\bar{\sigma}' \in L(N)$.

Let $\bar{\sigma}'$ be some element of $L(N')$. Since Φ is complete we have that $\langle \bar{\sigma}' \rangle \neq \emptyset$ and thus, since $\lfloor M' \rfloor \subseteq \lfloor M \rfloor$ and $\langle \bar{\sigma}' \rangle \subseteq \lfloor M' \rfloor$ there is some sequence $\bar{\sigma} \in L(N)$ such that $\langle \bar{\sigma} \rangle \cap \langle \bar{\sigma}' \rangle \neq \emptyset$. But, by Proposition 2 we must have that $\bar{\sigma} = \bar{\sigma}'$ and so the result follows.

Now let us suppose that $L(N') \subseteq L(N)$. We have that $\lfloor M \rfloor = \bigcup_{\bar{\sigma} \in L(N)} \langle \bar{\sigma} \rangle$ and $\lfloor M' \rfloor = \bigcup_{\bar{\sigma} \in L(A(M'))} \langle \bar{\sigma} \rangle$. Further, $\bigcup_{\bar{\sigma} \in L(A(M'))} \langle \bar{\sigma} \rangle \subseteq \bigcup_{\bar{\sigma} \in L(N')} \langle \bar{\sigma} \rangle$ and so the result follows. □

Proposition 5. *Let us suppose that M and M' are stream X-machines with the same set of processing relations Φ and let $N = A(M)$ and $N' = A(M')$. Then $\lfloor M' \rfloor = \lfloor M \rfloor$ if and only if $L(N') = L(N)$.*

Proof
This follows immediately from Proposition 4 by noting that $\lfloor M' \rfloor = \lfloor M \rfloor$ if and only if $\lfloor M' \rfloor \subseteq \lfloor M \rfloor \wedge \lfloor M \rfloor \subseteq \lfloor M' \rfloor$ and that $L(N') = L(N)$ if and only if $L(N') \subseteq L(N) \wedge L(N) \subseteq L(N')$. □

We are now in the position to relate the testing problem for a Stream X-machine M under equivalence to sequences for the FA $A(M)$.

Theorem 1. *Let us suppose that M is a stream X-machine with set Φ of processing relations that satisfies the specify for test conditions and let $N = A(M)$. Let Ψ be a fault domain for M such that every element of Ψ has set Φ of processing relations. If a set $T_A \in \mathcal{P}(\Phi^*)$ distinguishes N from fault domain $A(\Psi)$ under equivalence then the test function t, when applied to each element of T_A and $M' \in \Psi$, distinguishes M from M' under equivalence.*

Proof
Let M' be some element of Ψ such that $\lfloor M' \rfloor \neq \lfloor M \rfloor$ and assume that the test function t is being applied to an implementation that is equivalent to M'. Let $N' = A(M')$. Since for all $\bar{\sigma} \in \Phi^*$ we have that $\mathcal{R}(t, \bar{\sigma}, M') = \lfloor M' \rfloor(\bar{x})$ for $\bar{x} = t(\bar{\sigma})$ it is sufficient to prove that for some $\bar{\sigma} \in T_A$ we have that $\mathcal{R}(t, \bar{\sigma}, M') \neq \mathcal{R}(t, \bar{\sigma}, M)$. By Proposition 5 we know that, since $\lfloor M \rfloor \neq \lfloor M' \rfloor$, we have that $L(N) \neq L(N')$. Thus set $T_A \in \mathcal{P}(\Psi^*)$ distinguishes N from N' under equivalence and so there is some $\bar{\sigma} \in T_A$ such that either $\bar{\sigma} \in L(N) \setminus L(N')$ or $\bar{\sigma} \in L(N') \setminus L(N)$.

Let $\bar{\sigma}'$ denote a shortest prefix of $\bar{\sigma}$ such that $\bar{\sigma}' \in L(N) \setminus L(N')$ or $\bar{\sigma}' \in L(N') \setminus L(N)$. By the minimality of $\bar{\sigma}'$ either $\bar{\sigma}' \in L(N)$ or $\bar{\sigma}' = \bar{\sigma}''f$ for some $\bar{\sigma}'' \in L(N)$ and $f \in \Phi$.

Case 1: $\bar{\sigma}' \in L(N) \setminus L(N')$. By Proposition 3, $\mathcal{R}(t, \bar{\sigma}', M) \cap \langle \bar{\sigma}' \rangle \neq \emptyset$. Further, if $\mathcal{R}(t, \bar{\sigma}', M') \cap \langle \bar{\sigma}' \rangle \neq \emptyset$ then, by Proposition 3, $\bar{\sigma}' \in L(N')$ and so we must have that $\mathcal{R}(t, \bar{\sigma}', M') \cap \langle \bar{\sigma}' \rangle = \emptyset$. Thus, $\mathcal{R}(t, \bar{\sigma}', M') \neq \mathcal{R}(t, \bar{\sigma}', M)$ and so applying t with $\bar{\sigma}'$ distinguishes M and M'. Clearly, this also holds if we apply t with $\bar{\sigma}$ as required.

Case 2: $\bar{\sigma}' \in L(N') \setminus L(N)$. By Proposition 3 we know that $\mathcal{R}(t, \bar{\sigma}', M') \cap \langle \bar{\sigma}' \rangle \neq \emptyset$. Proof by contradiction: assuming that $\mathcal{R}(t, \bar{\sigma}', M') = \mathcal{R}(t, \bar{\sigma}', M)$ and so that $\mathcal{R}(t, \bar{\sigma}', M) \cap \langle \bar{\sigma}' \rangle \neq \emptyset$. By Proposition 3, we must have that $\bar{\sigma}' \in L(N)$, providing a contradiction as required. □

The following result is a generalisation of one in [23][3] and will be used for reasoning about testing under inclusion.

Proposition 6. *Let us suppose that M is a stream X-machine with relation set Φ that satisfies the specify for test conditions and the IUT behaves like a stream X-machine M_I with relation set Φ' such that $\Phi' \leq \Phi$. Then M_I conforms to M under inclusion if and only if $L(Abs_\Phi(M_I)) \subseteq L(A(M))$.*

Proof
First let us suppose that M_I conforms to M under inclusion and consider some $\bar{\sigma} \in L(Abs_\Phi(M_I))$ and so there is some $\bar{\sigma}' \in L(A(M_I))$ such that $\bar{\sigma}' \leq \bar{\sigma}$. Since Φ is complete, and so Φ' is complete, there is some $(\bar{x}, \bar{y}) \in \langle \bar{\sigma}' \rangle \subseteq \lfloor M_I \rfloor$ and clearly we have that $(\bar{x}, \bar{y}) \in \langle \bar{\sigma} \rangle$. Since M_I conforms to M under inclusion, $\lfloor M_I \rfloor \subseteq \lfloor M \rfloor$ and so there is some $\bar{\sigma}_1 \in L(A(M))$ with $(\bar{x}, \bar{y}) \in \langle \bar{\sigma}_1 \rangle$.

[3]The result in [23] required the IUT to be deterministic

Thus $\langle \bar{\sigma}_1 \rangle \cap \langle \bar{\sigma} \rangle \neq \emptyset$ and so, by Proposition 2, $\bar{\sigma}_1 = \bar{\sigma}$. Thus $\bar{\sigma} \in L(A(M))$, as required.

We now assume that $L(Abs_\Phi(M_I)) \subseteq L(A(M))$ and are required to prove that M_I conforms to M under inclusion. Consider some $(\bar{x}, \bar{y}) \in \lfloor M_I \rfloor$: it is sufficient to prove that $(\bar{x}, \bar{y}) \in \lfloor M \rfloor$. Since $(\bar{x}, \bar{y}) \in \lfloor M_I \rfloor$ there exists some $\bar{\sigma}' \in L(A(M_I))$ such that $(\bar{x}, \bar{y}) \in \langle \bar{\sigma}' \rangle$. But, since $L(Abs_\Phi(M_I)) \subseteq L(A(M))$ there is some $\bar{\sigma} \in L(A(M))$ such that $\bar{\sigma}' \leq \bar{\sigma}$ and so $(\bar{x}, \bar{y}) \in \langle \bar{\sigma} \rangle \subseteq \lfloor M \rfloor$. The result thus follows. □

We can now relate the notion of a checking experiment for a stream X-machine M under inclusion and test sequences that distinguish the FA $A(M)$ from $Abs_\Phi(\Psi)$ under inclusion.

Theorem 2. *Let us suppose that M is a stream X-machine with set Φ of processing relations that satisfies the specify for test conditions and let $N = A(M)$. Let Ψ be a fault domain for M and assume that for each element $M' \in \Psi$ we have that M' has a set Φ' of processing relations such that $\Phi' \leq \Phi$. If a set $T_A \subseteq L(N)$ distinguishes N from fault domain $Abs_\Phi(\Psi)$ under inclusion then the test function t, when applied to each element of T_A and $M' \in \Psi$ with $\lfloor M' \rfloor \not\subseteq \lfloor M \rfloor$, distinguishes M from M' under inclusion.*

Proof

By Proposition 6 we know that M' conforms to M under inclusion if and only if $L(Abs_\Phi(M')) \subseteq L(A(M))$. We require to prove that for all $M' \in \Psi$ with $\lfloor M' \rfloor \not\subseteq \lfloor M \rfloor$ there is some $\bar{\sigma} \in T_A$ such that $\mathcal{R}(t, \bar{\sigma}, M') \not\subseteq \mathcal{R}(t, \bar{\sigma}, M)$. Consider some such M' and $N' = Abs_\Phi(M')$ and assume that the test function t is being applied to an implementation that is equivalent to M'. By Proposition 4 we know that $L(N') \not\subseteq L(N)$. Since T_A distinguishes N from fault domain $Abs_\Phi(\Psi)$ under inclusion there is some $\bar{\sigma} \in T_A$ such that $\bar{\sigma} \in L(N') \setminus L(N)$.

Let $\bar{\sigma}'$ denote a shortest prefix of $\bar{\sigma}$ such that with $\bar{\sigma}' \in L(N') \setminus L(N)$. By the minimality of $\bar{\sigma}'$, $\bar{\sigma}' = \bar{\sigma}''f$ for some $\bar{\sigma}'' \in L(N)$ and $f \in \Phi$. By Proposition 3, $\mathcal{R}(t, \bar{\sigma}', M') \cap \langle \bar{\sigma}' \rangle \neq \emptyset$. Further, by Proposition 3, since $\bar{\sigma}' \notin L(N)$ we have that $\mathcal{R}(t, \bar{\sigma}', M) \cap \langle \bar{\sigma}' \rangle = \emptyset$. We therefore have that $\mathcal{R}(t, \bar{\sigma}', M') \not\subseteq \mathcal{R}(t, \bar{\sigma}', M)$ and so $\mathcal{R}(t, \bar{\sigma}, M') \not\subseteq \mathcal{R}(t, \bar{\sigma}, M)$ as required. □

Note that Theorems 1 and 2 operate in one direction but not the other. That is, we have not proved that if we find a set T_A of sequences from Φ^* such that t applied to the elements of T_A distinguishes M from the elements of Ψ then T_A also distinguished $A(M)$ from the elements of $Abs_\Phi(\Psi)$. In fact, results need not hold in the opposite direction since if we apply t to an IUT

equivalent to M' with sequence $\bar{\sigma}$, it is possible for the output to distinguish M' from M even if $\bar{\sigma} \notin L(A(M))$ and $\bar{\sigma} \notin L(Abs_\Phi(M'))$: the application of the test function to the IUT with $\bar{\sigma}$ might lead to the application of an input sequence $\bar{x}$ that triggers a sequence $\bar{\sigma}' \neq \bar{\sigma}$ such that $\bar{\sigma}' \notin L(A(M))$.

4. Testing for equivalence

We have shown that the problem of finding a checking sequence for a stream X-machine M with a given fault domain can be solved by considering the corresponding FA problem and that this can be done both when testing for equivalence and when testing for inclusion. In this section we show how the FA problem when testing for equivalence can be converted into a problem of producing a checking sequence for a particular FSM. As a result, the problem of producing a checking experiment for a stream X-machine M under equivalence can be solved by producing a checking experiment for an FSM generated from M.

If we are interested in testing for equivalence then we need to slightly adapt what we mean by building the IUT from trusted components. This is because under this notion of correctness, a component of the IUT conforms to a relation f in the specification if and only if it is equivalent to f. Thus, rather than saying that the stream X-machine M_I that models the IUT has a relation set Φ' such that $\Phi' \leq \Phi$, we require that M_I has relation set Φ.

We now show how checking experiments for FSMs under equivalence can be used, first relating the problem of distinguishing FA under equivalence to the problem of distinguishing FSMs under equivalence.

Given FA N with alphabet Z it is possible to define a corresponding FSM $\mathcal{F}_\mathcal{E}(N)$. Since we are concerned with reactive systems all states are final states and so we are only interested in such FA.

Definition 22. *Given FA $N = (S, s_0, Z, \delta, S)$ we define the FSM $\mathcal{F}_\mathcal{E}(N) = (S \cup \{s_e\}, s_0, Z, \{0,1\}, \delta')$ in which $s_e \notin S$, for all $z \in Z$ we have that $\delta(s_e, z) = \{(s_e, 0)\}$ and for all $s \in S$ and $z \in Z$ we have that*

1. *If $(s, z) \in dom\ \delta$ and $\delta(s, z) = s'$ then $\delta'(s, z) = \{(s', 1)\}$*
2. *If $(s, z) \notin dom\ \delta$ then $\delta'(s, z) = \{(s_e, 0)\}$*

The basic idea is that as long as the sequence of inputs being applied to $\mathcal{F}_\mathcal{E}(N)$ corresponds to an element of $L(N)$ the response to an input is 1 but once this is no longer the case all future outputs are 0.

Note that since we only consider deterministic FA, any FSM $\mathcal{F}_{\mathcal{E}}(N)$ will also be deterministic. Given a set Ψ of FA we let $\mathcal{F}_{\mathcal{E}}(\Psi)$ denote the corresponding set of FSMs: $\mathcal{F}_{\mathcal{E}}(\Psi) = \{R|\exists N \in \Psi.R = \mathcal{F}_{\mathcal{E}}(N)\}$. The following shows how $L(N)$ and $L(\mathcal{F}_{\mathcal{E}}(N))$ relate.

Proposition 7. *Given FA N and $R = \mathcal{F}_{\mathcal{E}}(N)$ we have that $z_1, \ldots, z_k \in L(N)$ if and only if $z_1/1, \ldots, z_k/1 \in L(R)$.*

Proof
We will prove a slightly stronger result, this being that for FA N and $R = \mathcal{F}_{\mathcal{E}}(N)$, $\bar{z} = z_1, \ldots, z_k$ is the label of a path of N with ending state $s \in S$ if and only if $z_1/1, \ldots, z_k/1$ is the label of a path of R with ending state s.

We first prove that if $\bar{z} = z_1, \ldots, z_k$ is the label of a path of N with ending state s then $z_1/1, \ldots, z_k/1$ is the label of a path of R with ending state s. We use proof by induction on the length of the sequence. The result follows immediately for the base case, which is the empty sequence. Now let us assume that it holds for all sequences of length less than $k > 0$ and consider some $\bar{z} = z_1, \ldots, z_k$ and state $s \in S$ such that $\bar{z}$ is the label of a path of N with ending state s. Clearly $z_1, \ldots, z_{k-1}$ is the label of a path of N with ending state s' for some $s' \in S$. By the inductive hypothesis, $z_1/1, \ldots, z_{k-1}/1$ is the label of a path of M with ending state s'. It is now sufficient to observe that in N there is a transition from s' to s with label z_k and thus in R there is a transition from s' to s with label $z_k/1$.

We now prove that if $z_1/1, \ldots, z_k/1$ is the label of a path of R with ending state $s \in S$ then $\bar{z} = z_1, \ldots, z_k$ is the label of a path of N with ending state s. Again we use proof by induction on the length of the sequence and the result follows immediately for the base case, which is the empty sequence. Now let us assume that it holds for all sequences of length less than $k > 0$ and consider some $\bar{z} = z_1, \ldots, z_k$ and state $s \in S$ such that $z_1/1, \ldots, z_k/1$ is the label of a path of R with ending state s. Clearly $z_1/1, \ldots, z_{k-1}/1$ is the label of a path of R with ending state s' for some $s' \in S$. By the inductive hypothesis, $z_1, \ldots, z_{k-1}$ is the label of a path of N with ending state s'. It is now sufficient to observe that in R there is a transition from s' to s with label $z_k/1$ and thus in N there is a transition from s' to s with label z_k. □

It therefore appears that there should be some relationship between sets of sequences that distinguish an FA N from elements of a fault domain and checking experiments for $\mathcal{F}_{\mathcal{E}}(N)$. However, $\mathcal{F}_{\mathcal{E}}(N)$ is completed through the addition of a state s_e and so not all paths in $\mathcal{F}_{\mathcal{E}}(N)$ correspond to paths of

N and so we have to be careful in defining such a relationship. Specifically, we have to make sure that we do not use a sequence $\bar{\sigma}$ to distinguish $\mathcal{F}_{\mathcal{E}}(N)$ from some $\mathcal{F}_{\mathcal{E}}(N')$ when $\bar{\sigma}$ is not in $L(N)$ and also is not in $L(N')$. Given a checking experiment T_F for $\mathcal{F}_{\mathcal{E}}(N)$, in this paper we overcome this by using the prefixes of sequences in T_F in order to distinguish N from elements of its fault domain. These prefixes are only required for reasoning about the FA that corresponds to a stream X-machine: in testing from a stream X-machine we gain nothing by using prefixes of a test sequence and so can eliminate them.

We are now in a position to show that for FA N, checking experiments of $\mathcal{F}_{\mathcal{E}}(N)$ under equivalence correspond to sequences that distinguish N from the corresponding fault domain.

Theorem 3. *Let us suppose that we have FA $N = (S, s_0, Z, \delta, S)$ with fault domain Ψ, $R = \mathcal{F}_{\mathcal{E}}(N)$ and $T_F \in \mathcal{P}(Z^*)$ is a test suite for R. Let T_A denote the set of prefixes of T_F. T_F is a checking experiment for R with $\mathcal{F}_{\mathcal{E}}(\Psi)$ under equivalence if and only if T_A distinguishes N from Ψ under equivalence.*

Proof

First assume that T_A distinguishes N from Ψ under equivalence and let R' be an element of $\mathcal{F}_{\mathcal{E}}(\Psi)$ such that $L(R') \neq L(R)$. Then we are required to prove that T_F contains a sequence that distinguishes between R' and R. Let N' denote the FA in Ψ such that $R' = \mathcal{F}_{\mathcal{E}}(N')$.

By Proposition 7 we know that $L(N) \neq L(N')$ and so, since T_A distinguishes N from Ψ under equivalence we must have that T_A distinguishes N from N' under equivalence. There are two cases:

1. There is some $\bar{z} = z_1, \ldots, z_k \in T_A$ such that $\bar{z} \in L(N) \backslash L(N')$. Thus, by Proposition 7 we have that $z_1/1, \ldots, z_k/1 \in L(R)$ and $z_1/1, \ldots, z_k/1 \notin L(R')$ and so $\bar{z}$ distinguishes R' and R as required.
2. There is some $\bar{z} \in T_A$ such that $\bar{z} \in L(N') \setminus L(N)$. By Proposition 7 we have that $z_1/1, \ldots, z_k/1 \in L(R')$ and $z_1/1, \ldots, z_k/1 \notin L(R)$ and so $\bar{z}$ distinguishes R' and R as required.

Now assume that T_F is a checking experiment for R in $\mathcal{F}_{\mathcal{E}}(\Psi)$ and let N' be an element of Ψ such that $L(N') \neq L(N)$. We are required to prove that T_A contains a sequence that distinguishes N from N' under equivalence and we let $R' = \mathcal{F}_{\mathcal{E}}(N')$.

By Proposition 7 we have that $L(R) \neq L(R')$ and so, since T_F is a checking experiment for R given $\mathcal{F}_{\mathcal{E}}(\Psi)$ we must have that T_F distinguishes R and R'.

Choose some shortest $\bar{z} = z_1, \ldots, z_k \in T_A$ that distinguishes R and R'. By the minimality of $\bar{z}$ we must have that exactly one of R and R' responds to $\bar{z}$ through a sequence of 1s and so we have two cases:

1. $z_1/1, \ldots, z_k/1 \in L(R) \setminus L(R')$. By Proposition 7 we have that $\bar{z} \in L(N)$ and $\bar{z} \notin L(N')$ and so $\bar{z}$ distinguishes N from N' under equivalence as required.
2. $z_1/1, \ldots, z_k/1 \in L(R') \setminus L(R)$. By Proposition 7 we have that $\bar{z} \in L(N')$ and $\bar{z} \notin L(N)$ and so $\bar{z}$ distinguishes N from N' under equivalence as required.

□

We can now put results together to relate checking experiments for FSMs and stream X-machines under equivalence.

Theorem 4. *Let us suppose that $M = (In, Out, S, Mem, \Phi, F, s_0, m_0, S)$ is a stream X-machine specification with relation set Φ that satisfies the specify for test conditions. Let us suppose that Ψ is the fault domain used with M and all elements of Ψ have relation set Φ. If a set $T_F \subseteq \Phi^*$ is a checking experiment for $\mathcal{F}_{\mathcal{E}}(A(M))$ with $\mathcal{F}_{\mathcal{E}}(\Psi)$ under equivalence then the test function t, when applied to each element of T_F and $M' \in \Psi$ with $\lfloor M' \rfloor \neq \lfloor M \rfloor$, distinguishes M from M' under equivalence.*

Proof
Let T_A denote the set of prefixes of sequences in T_F. Clearly t, when applied to each element of T_F and $M' \in \Psi$, distinguishes M from M' under equivalence if and only if t, when applied to each element of T_A and $M' \in \Psi$, distinguishes M from M' under equivalence. The result therefore follows from Theorems 1 and 3. □

It is clear that the processes of producing $\mathcal{F}_{\mathcal{E}}(M)$ and converting the resultant checking experiment T_F into a checking experiment for M can both be performed in polynomial time.

5. Testing for inclusion

In this section we show how checking experiments for FSMs can be used when testing for inclusion. We first relate the problem of distinguishing FA under inclusion to the problem of distinguishing FSMs under inclusion. Given FA N with alphabet Z it is possible to define a corresponding FSM $\mathcal{F}_{\mathcal{I}}(N)$ for testing for inclusion.

Definition 23. *Given FA* $N = (S, s_0, Z, \delta, S)$ *we define the FSM* $\mathcal{F}_{\mathcal{I}}(N) = (S \cup \{s_e\}, s_0, Z, \{0,1\}, \delta')$ *in which* $s_e \notin S$*, for all* $z \in Z$ *we have that* $\delta(s_e, z) = \{(s_e, 0)\}$ *and for all* $s \in S$ *and* $z \in Z$ *we have that*

1. *If* $(s, z) \in dom\ \delta$ *and* $\delta(s, z) = s'$ *then* $\delta'(s, z) = \{(s', 1), (s', 0)\}$
2. *If* $(s, z) \notin dom\ \delta$ *then* $\delta'(s, z) = \{(s_e, 0)\}$

The idea here is that while an input sequence is in $L(N)$, the FSM $\mathcal{F}_{\mathcal{I}}(N)$ follows the corresponding path and can produce either 0 or 1 at each stage. If an implementation FA N' does not have this sequence then the behaviour of $\mathcal{F}_{\mathcal{I}}(N')$ is restricted to some subset of this but this is acceptable under inclusion. However, if the implementation FA N' has a sequence $\bar{\sigma} \in L(N')$ that is not in $L(N)$ then $\mathcal{F}_{\mathcal{I}}(N')$ can produce an output sequence consisting only of 1s in response to $\bar{\sigma}$ but the specification $\mathcal{F}_{\mathcal{I}}(N)$ cannot.

Note that even if N is a deterministic FA, FSM $\mathcal{F}_{\mathcal{I}}(N)$ can be nondeterministic. Given a set Ψ of FA we let $\mathcal{F}_{\mathcal{I}}(\Psi)$ denote the corresponding set of FSMs: $\mathcal{F}_{\mathcal{I}}(\Psi) = \{R' | \exists N \in \Psi . R' = \mathcal{F}_{\mathcal{I}}(N)\}$.

The following shows how $L(N)$ and $L(\mathcal{F}_{\mathcal{I}}(N))$ relate.

Proposition 8. *Given FA* N *and* $M = \mathcal{F}_{\mathcal{I}}(N)$*,* $\bar{z} = z_1, \ldots, z_k \in L(N)$ *if and only if* $z_1/y_1, \ldots, z_k/y_k \in L(R)$ *for all* $y_1, \ldots, y_k \in \{0,1\}$*.*

Proof

We will prove that for FA N and $R = \mathcal{F}_{\mathcal{I}}(N)$, $\bar{z} = z_1, \ldots, z_k$ is the label of a path of N with ending state $s \in S$ if and only if $z_1/y_1, \ldots, z_k/y_k$ is the label of a path of R with ending state s for all $y_1, \ldots, y_k \in \{0,1\}$.

We first prove that if $\bar{z} = z_1, \ldots, z_k$ is the label of a path of N with ending state s then $z_1/y_1, \ldots, z_k/y_k$ is the label of a path of R with ending state s for all $y_1, \ldots, y_k \in \{0,1\}$. We use proof by induction on the length of the sequence being considered. The result follows immediately for the base case ϵ. Now let us assume that it holds for sequences of length less than k and consider some $\bar{z} = z_1, \ldots, z_k$ and state $s \in S$ such that $\bar{z}$ is the label of a path of N with ending state s. Clearly $z_1, \ldots, z_{k-1}$ is the label of a path of N with ending state s' for some $s' \in S$. By the inductive hypothesis, $z_1/y_1, \ldots, z_{k-1}/y_{k-1}$ is the label of a path of R with ending state s' for all $y_1, \ldots, y_{k-1} \in \{0,1\}$. It is now sufficient to observe that in N there is a transition from s' to s with label z_k and thus in M there is a transition from s' to s with label $z_k/0$ and there is a transition from s' to s with label $z_k/1$.

We now prove that if $z_1/y_1, \ldots, z_k/y_k$ is the label of a path of R with ending state $s \in S$ for all $y_1, \ldots, y_k \in \{0,1\}$ then $\bar{z} = z_1, \ldots, z_k$ is the label of a path of N with ending state s. Again we use proof by induction on the length of the sequence and the result follows immediately for the base case ϵ. Now let us assume that it holds for all sequences of length less than k and consider some $\bar{z} = z_1, \ldots, z_k$ and state $s \in S$ such that $z_1/y_1, \ldots, z_k/y_k$ is the label of a path of R with ending state s for all $y_1, \ldots, y_k \in \{0,1\}$. Clearly $z_1/y_1, \ldots, z_{k-1}/y_{k-1}$ is the label of a path of R with ending state s' for all $y_1, \ldots, y_{k-1} \in \{0,1\}$ for some $s' \in S$. By the inductive hypothesis, $z_1, \ldots, z_{k-1}$ is the label of a path of N with ending state s'. It is now sufficient to observe that in R there is a transition from s' to s with label $z_k/0$ and a transition from s' to s with label $z_k/1$ and thus in N there is a transition from s' to s with label z_k. □

Again, we have the issue that a sequence $\bar{z}$ may distinguish between FSMs $\mathcal{F}_{\mathcal{I}}(N)$ and $\mathcal{F}_{\mathcal{I}}(N')$ but we may have to use a prefix of this to distinguish between N and N'.

Theorem 5. *Let us suppose that we have FA $N = (S, s_0, Z, \delta, S)$ with fault domain Ψ, $R = \mathcal{F}_{\mathcal{I}}(N)$ and $T_F \in \mathcal{P}(Z^*)$ is a test suite for R. T_F is a checking experiment for R with $\mathcal{F}_{\mathcal{I}}(\Psi)$ under inclusion if and only if the set T_A of prefixes of T_F distinguishes N from Ψ under inclusion.*

Proof

First assume that T_A distinguishes N from Ψ under inclusion and let R' be an element of $\mathcal{F}_{\mathcal{I}}(\Psi)$ such that $L(R') \not\subseteq L(R)$. Then we are required to prove that T_F contains a sequence that distinguishes between R' and R under inclusion. Let N' denote the FA in Ψ such that $R' = \mathcal{F}_{\mathcal{I}}(N')$.

By Proposition 8, $L(N') \not\subseteq L(N)$ and so, since T_A distinguishes N from Ψ under inclusion we must have that T_A distinguishes N from N' under inclusion. Consider an element $\bar{z}$ of T_A such that $\bar{z} = z_1, \ldots, z_k \in L(N') \setminus L(N)$. By Proposition 8 we have that $z_1/y_1, \ldots, z_k/y_k \in L(R')$ for all $y_1, \ldots, y_k \in \{0,1\}$ and $z_1/1, \ldots, z_k/1 \notin L(R)$ and so $\bar{z}$ distinguishes R from R' under inclusion. Since a prefix of a test sequence in T_F distinguishes R from R' under inclusion we have that T_F distinguishes R from R' under inclusion as required.

Now assume that T_F is a checking experiment for R in $\mathcal{F}_{\mathcal{I}}(\Psi)$ under inclusion and let N' be an element of Ψ such that $L(N') \not\subseteq L(N)$. We are required to prove that T_A contains a sequence that distinguishes N from N' under inclusion. Let $R' = \mathcal{F}_{\mathcal{I}}(N')$.

By Proposition 8, $L(R') \not\subseteq L(R)$ and so, since T_F is a checking experiment for R given $\mathcal{F}_{\mathcal{I}}(\Psi)$ under inclusion we must have that T_F distinguishes R and R' under inclusion. Consider some shortest prefix $\bar{z} = z_1, \ldots, z_k$ of a sequence from T_F that distinguishes R and R' under inclusion. Clearly $\bar{z} \in T_A$. By the definition of $R = \mathcal{F}_{\mathcal{I}}(N)$ and $R' = \mathcal{F}_{\mathcal{I}}(N')$ and the minimality of $\bar{z}$ we must have that $z_1/1, \ldots, z_k/1 \in L(R')$ and $z_1/1, \ldots, z_k/1 \notin L(R)$. By Proposition 8 we have that $\bar{z} \in L(N')$ and $\bar{z} \notin L(N)$ and so $\bar{z}$ distinguishes N from N' under inclusion as required. □

We can now put these results together to relate checking experiments for FSMs and stream X-machines under inclusion.

Theorem 6. *Let us suppose that $M = (In, Out, S, Mem, \Phi, F, s_0, m_0, S)$ is a stream X-machine specification with relation set Φ that satisfies the specify for test conditions. Let us suppose that Ψ is the fault domain used with M and all elements of Ψ have relation sets of the form Φ' such that $\Phi' \leq \Phi$. If a set $T_F \subseteq \Phi^*$ is a checking experiment for $\mathcal{F}_{\mathcal{I}}(A(M))$ with $\mathcal{F}_{\mathcal{I}}(\Psi)$ under inclusion then for all $M' \in \Psi$ we have that the test function t, when applied to each element of T_F, distinguishes M from Ψ under inclusion.*

Proof
Let T_A denote the set of prefixes of sequences in T_F. Clearly t, when applied to each element of T_F and $M' \in \Psi$, distinguishes M from M' under inclusion if and only if t, when applied to each element of T_A and $M' \in \Psi$, distinguishes M from M' under inclusion. The result thus follows from Theorems 2 and 5. □

Finally, note that the processes of producing $\mathcal{F}_{\mathcal{I}}(M)$ and converting the resultant checking experiment T_F into a checking experiment for M can both be performed in polynomial time.

6. Observations regarding test generation

In this section we make some general observations and describe ways in which test generation can be made more efficient. The first observation relates to the class of algorithms, for generating a checking experiment, that can be used. The proposed method operates by converting a stream X-machine M into an FSM R and then producing a checking experiment from R. The FSM R need not be strongly connected and it might therefore appear that we cannot use methods for generating checking experiments that require

strongly connected FSMs (see, for example, [10, 2, 42, 32, 11]). However, R can be converted into a strongly connected FSM R_c by adding a reset: an input r that takes R_c back to its initial state (with fixed output) irrespective of the current state. Checking experiments can then be generated from R_c, any test sequence that contains resets essentially representing a set of test sequences.

6.1. Testing deterministic implementations

In some situations it is known that the IUT is deterministic and it may then be possible to make use of this knowledge. Let us suppose that we are testing for equivalence and the checking experiment T_F produced from the FSM R, that corresponds to M, contains a sequence of the form $\bar{\sigma}f/0\bar{\sigma}'$ for some $\bar{\sigma}$ and $\bar{\sigma}'$ in which $\bar{\sigma}$ does not contain 0. It is possible to remove $\bar{\sigma}'$ since we know that all outputs after the first 0 must be 0 and this makes the test more efficient. If $\bar{\sigma}$ has input portion $f_1, \ldots, f_k$ then testing from M based on $\bar{\sigma}f/0$ is effectively checking that $f_1, \ldots, f_k$ is implemented in the IUT and that it cannot be followed by f. Since M is completely specified, there is some $f' \in \Phi$ such that $f_1, \ldots, f_k, f' \in L(A(M))$ and the domains of f and f' intersect. Thus, since the IUT is deterministic, it is sufficient to check that $f_1, \ldots, f_k, f'$ is implemented: a deterministic implementation cannot implement both $f_1, \ldots, f_k, f$ and $f_1, \ldots, f_k, f'$. Thus we can replace $\bar{\sigma}f/0$ by $\bar{\sigma}f'/1$ in T_F. If T_F already contains a sequence that starts with $\bar{\sigma}f'/1$ then we can simply remove $\bar{\sigma}f/0$. This provides opportunities for further optimisation and a similar observation can be made when testing for inclusion rather than equivalence.

6.2. Issues raised by nondeterminisim

If the specification is nondeterministic then there is scope for making testing more efficient by applying an adaptive approach: rather than repeatedly trying to execute a sequence $\bar{\sigma}$ of processing relations from a checking experiment, we use an adaptive process that allows us to try to execute an alternative sequence from the checking experiment if testing diverges from $\bar{\sigma}$ without producing a failure. In order to do this we have to define an adaptive test process and it should then be possible to utilise adaptive methods for testing from FSMs (see, for example, [43, 30, 44, 45]).

7. Conclusions

When testing against a formal model or specification, a fault domain describes a set of possible behaviours for the implementation under test (IUT). The presence of a fault domain Ψ allows the tester to reason about test effectiveness and potentially to produce a test that determines correctness as long as the IUT is functionally equivalent to an unknown element of Ψ. Such a test is called a checking experiment.

Stream X-machines are a state based formalism that have associated with them a particular approach to development. Under this approach, the IUT is built from trusted components and testing constitutes determining whether these components have been put together correctly. Previous work has shown that some methods for generating checking experiments from finite state machines (FSMs) can be adapted to produce checking sequences for stream X-machines but only a few of the many methods for generating checking sequences from FSMs have been considered.

This paper has shown that any method for generating a checking sequence from an FSM can be applied to produce a checking sequence for a stream X-machine. This holds whether we are testing that the IUT is equivalent to the specification or that every trace (input/output sequence) of the IUT is also a trace of the specification. It is possible to convert the stream X-machine M into an FSM R from which a checking sequence T_F can be generated and also to convert T_F into a checking experiment for M. In addition, this conversion process can be computed in polynomial time. We considered two cases: testing for equivalence and testing for inclusion. Interestingly, the approach to be used does not depend on whether the specification or IUT are deterministic but on the conformance relation used.

The results in this paper have two main consequences. First, they show how any checking experiment method, for FSMs, can be adapted to testing from a stream X-machine. The second benefit is that they show that we can use methods for producing checking experiments from FSMs that have fault domains other than the one traditionally used when testing from a stream X-machine. Thus the results make many more methods for generating checking experiments available to the tester and allow the tester to test for a wider range of sets of faults.

There are several lines of future work. First, recent work has shown how the traditional specify for test conditions used with stream X-machines can be weakened [24, 26] and it seems likely that the results in this paper can

be generalised along similar lines. Second, we have shown how a checking experiment might be further reduced and there may be additional scope for such optimisation.

References

[1] T. S. Chow, Testing software design modelled by finite state machines, IEEE Transactions on Software Engineering 4 (1978) 178–187.

[2] F. C. Hennie, Fault-detecting experiments for sequential circuits, in: Proceedings of Fifth Annual Symposium on Switching Circuit Theory and Logical Design, Princeton, New Jersey, 1964, pp. 95–110.

[3] E. P. Moore, Gedanken-experiments, in: C. Shannon, J. McCarthy (Eds.), Automata Studies, Princeton University Press, 1956.

[4] A. Petrenko, N. Yevtushenko, Testing from partial deterministic FSM specifications, IEEE Transaxtions on Computers 54 (9) (2005) 1154–1165.

[5] S. Eilenberg, Automata, languages and machines, Vol. A, Academic Press, 1974.

[6] M. Holcombe, X-machines as a basis for dynamic system specification, Software Engineering Journal 3 (2) (1988) 69–76.

[7] M. Holcombe, F. Ipate, Correct Systems: Building a Business Process Solution, Springer-Verlag, 1998.

[8] ITU-T, Recommendation Z.100 Specification and description language (SDL), International Telecommunications Union, Geneva, Switzerland, 1999.

[9] D. Harel, M. Politi, Modeling reactive systems with statecharts: the STATEMATE approach, McGraw-Hill, New York, 1998.

[10] G. Gonenc, A method for the design of fault detection experiments, IEEE Transactions on Computers 19 (1970) 551–558.

[11] H. Ural, X. Wu, F. Zhang, On minimizing the lengths of checking sequences, IEEE Transactions on Computers 46 (1) (1997) 93–99.

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

[13] M. Holcombe, An integrated methodology for the specification, verification and testing of systems, The Journal of Software Testing, Verification and Reliability 3 (3/4) (1993) 149–163.

[14] D. Jackson, M. Holcombe, F. Ratnieks, Trail geometry gives polarity to ant foraging networks, Nature 432 (2004) 907–909.

[15] R. H. Smallwood, M. Holcombe, The epitheliome project: multiscale agent-based modeling of epithelial cells, in: IEEE International Symposium on Biomedical Imaging: From Nano to Macro (ISBI 2006), 2006, pp. 816–819.

[16] I. Stamatopoulou, M. Gheorghe, P. Kefalas, Modelling dynamic organization of biology-inspired multi-agent systems with communicating X-machines and population p systems, in: Workshop on Membrane Computing, 2004, pp. 389–403.

[17] C. A. Rouff, M. G. Hinchey, J. L. Rash, W. F. Truszkowski, Towards a hybrid formal method for swarm-based exploration missions, in: SEW, 2005, pp. 253–264.

[18] J. Aguado, T. Balanescu, A. J. Cowling, M. Gheorghe, M. Holcombe, F. Ipate, P Systems with Replicated Rewriting and Stream X-Machines (Eilenberg Machines), Fundam. Inform. 49 (1–3) (2002) 17–33.

[19] T. Balanescu, H. Georgescu, M. Gheorghe, C. Vertan, Communicating stream X-machines systems are no more than X-machines, Journal of Universal Computing 5 (9) (1999) 494–507.

[20] F. Bernardini, M. Gheorghe, M. Holcombe, P X systems = P systems + X machines, Natural Computing 2 (3) (2003) 201–213.

[21] K. Bogdanov, M. Holcombe, Statechart testing method for aircraft control systems, The Journal of Software Testing, Verification and Reliability 11 (1) (2001) 39–54.

[22] R. M. Hierons, M. Harman, Testing conformance to a quasi-non-determinstic stream X-machine, Formal Aspects of Computing 12 (6) (2000) 423–442.

[23] R. M. Hierons, M. Harman, Testing conformance of a deterministic implementation to a non-deterministic stream X-machine, Theoretical Computer Science 323 (1–3) (2004) 191–233.

[24] R. M. Hierons, F. Ipate, Testing a deterministic implementation against a non-controllable non-deterministic stream X-machine, Formal Aspects of Computing 20 (6) (2008) 597–617.

[25] F. Ipate, M. Holcombe, Generating test sets from non-deterministic stream X-machines, Formal Aspects of Computing 12 (6) (2000) 443–458.

[26] F. Ipate, Testing against a non-controllable stream X-machine using state counting, Theoretical Computer Science 353 (1–3) (2006) 291–316.

[27] F. Ipate, M. Holcombe, An integration testing method that is proved to find all faults, International Journal of Computer Mathematics 63 (1997) 159–178.

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

[29] N. Yevtushenko, A. Petrenko, Synthesis of test experiments in some classes of automata, Automatic Control and Computer Sciences 4.

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

[31] M. O. Rabin, D. Scott, Finite automata and their decision problems, IBM Journal of Research and Development 3 (2) (1959) 114–125.

[32] R. M. Hierons, H. Ural, Optimizing the length of checking sequences, IEEE Transactions on Computers 55 (5) (2006) 618–629.

[33] M. C. Gaudel, Testing can be formal too, in: 6th International Joint Conference CAAP/FASE Theory and Practice of Software Development (TAPSOFT'95), Vol. 915 of Lecture Notes in Computer Science, Springer, 1995, pp. 82–96.

[34] M.-C. Gaudel, Testing from formal specifications, a generic approach, in: Ada–Europe, Vol. 2043 of Springer Lecture Notes in Computer Science, Springer-Verlag, 2001, pp. 35–48.

[35] M.-C. Gaudel, P. R. James, Testing algebraic data types and processes: a unifying theory, Formal Aspects of Computing 10 (5–6) (1998) 436–451.

[36] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lüttgen, A. J. H. Simons, S. A. Vilkomir, M. R. Woodward, H. Zedan, Using formal specifications to support testing, ACM Compututing Surveys 41 (2).

[37] F. Ipate, M. Holcombe, A method for refining and testing generalised machine specifications, International Journal of Computer Mathematics 68 (1998) 197–219.

[38] Y. Lei, R. H. Carver, D. C. Kung, V. Gupta, M. Hernandez, A state exploration-based approach to testing java monitors, in: 17th International Symposium on Software Reliability Engineering (ISSRE 2006), 2006, pp. 256–265.

[39] K.-C. Tai, R. H. Carver, E. E. Obaid, Debugging concurrent ada programs by deterministic execution, IEEE Transactions on Software Engineering 17 (1) (1991) 45–63.

[40] Y. Lei, R. H. Carver, Reachability testing of concurrent programs, IEEE Transactions on Software Engineering 32 (6) (2006) 382–403.

[41] G. Luo, A. Petrenko, G. v. Bochmann, Selecting test sequences for partially-specified nondeterministic finite state machines, in: The 7th IFIP Workshop on Protocol Test Systems, Chapman and Hall, Tokyo, Japan, 1994, pp. 95–110.

[42] R. M. Hierons, H. Ural, Reduced length checking sequences, IEEE Transactions on Computers 51 (9) (2002) 1111–1117.

[43] R. M. Hierons, Adaptive testing of a deterministic implementation against a nondetermistic finite state machine, The Computer Journal 41 (5) (1998) 349–355.

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

[45] P. Tripathy, K. Naik, Generation of adaptive test cases from non-deterministic finite state models, in: Proceedings of the 5th International Workshop on Protocol Test Systems, Montreal, 1992, pp. 309–320.