Testing From a Finite State Machine: Extending Invertibility to Sequences

Robert M. Hierons, Goldsmiths College, University of London

Abstract

When testing a system modelled as a finite state machine it is desirable to minimize the effort required. Yang and Ural [1990] demonstrate that it is possible to utilize test sequence overlap in order to reduce the test effort and Hierons [1996] represents this overlap by using *invertible transitions*. In this paper invertibility will be extended to *sequences* in order to further reduce the test effort and encapsulate a more general type of test sequence overlap. It will also be shown that certain properties of invertible sequences can be used in the generation of state identification sequences.

1 Introduction

A finite state machine (FSM) can be used to model a software system. In particular, an FSM can be used to model the control section of a communications protocol (Huang and Hsu [1994]). If some FSM model F exists and an implementation I, that is intended to implement F, has been produced it is important to verify I relative to F. In order to do this it is necessary to test I. When testing I against F it is normal to assume that I can be modelled as an FSM and the testing problem then becomes an instance of the FSM equivalence problem.

A number of specification languages, such as SDL and ESTELLE, are extensions to the FSM formalism. Many specifications in such languages can be converted into FSMs from which tests can be generated (Luo and Chen [1989], Luo et al. [1994b], Petrenko et al. [1994a]).

An alternative approach, to modelling a communications protocol, is to use a process algebra such as LOTOS. There has been much work on generating conformance relations and canonical testers from process algebra descriptions (Brinksma [1988], Wezeman [1989]). It has also been noted that equivalent conformance relations can be defined for specification languages such as SDL and ESTELLE and thus, potentially, for FSM (Phalippou [1993]). When the specification is finite, it can be modelled as an FSM and FSM based testing techniques can then be applied (Fujiwara and v. Bochmann [1992]).

A number of techniques have been developed for testing from an FSM. These are based on several different test criteria, including simply *executing* every transition (Sidhu and Leung [1988]), *testing* every transition (Sidhu and Leung [1988], Aho et al. [1988], Yang and Ural [1990], Hierons [1996]) and producing a *checking sequence*: a test that will distinguish between the FSM model and any non-equivalent FSM that has no more states (Rezaki and Ural [1995], Ural et al. [1997]). Given a test criterion, it is desirable to produce the *shortest* test that satisfies this criterion. Here the problem of finding the shortest test sequence, that includes a test for every transition, will be considered.

While, ideally, a checking sequence should be produced, in some cases this may not be practical and weaker criteria are used. The relative effectiveness of the related methods, at finding faults, is still an open question. The experience of Motteler et al. [1994] and Sidhu and Leung [1988] suggests that test sequences that test every transition are usually effective at locating faults.

The test generation problem is further complicated if the system under test is embedded in some environment and all communications go through this environment. If there is a model of the environment, this model must be considered when deriving tests (Petrenko et al. [1996]).

Hierons [1996] discusses the application of invertible transitions to test sequence generation. In Section 2 invertibility will be extended to sequences and a number of properties will be derived. The relationship between invertibility and state identification techniques will also be investigate and it will be demonstrated that this can be used in the generation of state identification sequences. An algorithm for finding invertible sequences and UIOs is given in Section 2.4. In Section 3 an algorithm is introduced that both extends the applicability of the algorithm given in Hierons [1996] and applies invertible sequences to reduce the length of the test sequence produced. This algorithm

is then applied to a small example, in order to illustrate the method, and compared to alternative algorithms. Finally, conclusions are drawn.

2 Invertible Sequences

2.1 Some definitions

A *Finite State Machine* F with *input alphabet* Σ and *output alphabet* Θ can be represented by a tuple (S, T, s_1); S is the finite set of *states*, T is the finite set of *transitions* between these states, and s_1 is the *initial state*. Each transition is in the form $(s, s', x/y)$ where s is the *initial state*, s' is the *final state*, $x \in \Sigma$ is the *input* involved in this transition, and $y \in \Theta$ is the output generated by this transition.

An FSM is said to be *completely specified* if for each input value $x \in \Sigma$ and state $s_i \in S$ there is a transition from s_i with input x. An FSM is *deterministic* if for every state s_i and input x there is at most one transition from s_i with input x. If an FSM is deterministic it is possible to represent the transitions by (possibly partial) functions δ and λ, the *next state* and *output* functions respectively. Thus, if a transition with input x is executed from state s_i output $\lambda(s_i, x)$ is produced and the FSM moves to state $\delta(s_i, x)$. These functions can be extended in a natural way to functions δ^* and λ^* that give the final state and output respectively when executing a sequence of input values from a state. As is usual, it will be assumed that any FSM considered is deterministic and completely specified.

Two states s_i and s_j are said to be *equivalent* if for every input sequence X, $\lambda^*(s_i, X) = \lambda^*(s_j, X)$. An FSM is *minimal* if no two states are equivalent and two FSM are *equivalent* if their initial states are equivalent. It will be assumed that any FSM being considered is minimal as any (deterministic) FSM can be converted to an equivalent (deterministic) minimal FSM (Moore [1956]). See e.g. Kohavi [1978] for more information on FSM.

When testing a transitions it is necessary to check its final state. In order to do this one of the following approaches can be applied:

1. A distinguishing sequence (DS)
2. Unique input/output sequences (UIO)
3. A characterizing set

A distinguishing sequence is a sequence that produces a different output for each state. A UIO u for a state s has the property that for each $s' \neq s$, $\lambda^*(s,u) \neq \lambda^*(s',u)$, and thus u is capable of verifying state s, but not necessarily any other state. Kohavi and Kohavi [1968] note that, when a preset test sequence is not required, an adaptive distinguishing sequence can be used. Adaptive distinguishing sequences have the advantage that there is a polynomial upper bound for their length, when they exist (Lee and Yannakakis [1994]).

Some FSM do not have either a DS or a UIO for every state. It is then necessary to use a characterizing set W: a set of input sequences with the property that for every pair of states $s \neq s'$ there is some $w_i \in W$ such that $\lambda^*(s,w_i) \neq \lambda^*(s',w_i)$. Thus, the output sequences produced by executing each $w_i \in W$ from s verifies s.

A *directed graph (digraph)* G is defined by an ordered pair (V,E), where V is a set of vertices and E is a set of edges between vertices. An edge can have a label and thus each edge is represented by a tuple (v_i,v_j,l) where v_i is the initial vertex, v_j is the final vertex, and l is the label. Given a vertex v in a digraph (V,E) the number of edges entering v is denoted by $indegree_E(v)$ and the number of edges leaving v is denoted by $outdegree_E(v)$. Clearly an FSM can be represented by a digraph and throughout this paper the two formalisms will be considered to be equivalent and so the two sets of terminology will be used interchangeably.

A *network* is a digraph in which every edge is given a non-negative integer capacity and there are two special vertices; the *source* and the *sink*. A *flow* for a network is the assignment of an integer flow to each edge such that the flow at an edge does not exceed the capacity of the edge and the flow is conserved at every vertex except for the source and the sink. The net flow through the network is the net flow leaving the source, which is equal to the net flow entering the sink. If each edge is given a cost, the *cost* of the flow is the sum, over the edges, of the cost of the edge multiplied by the flow through the edge. See e.g. Gibbons [1985] for more information on graphs, digraphs and networks.

Hierons [1996] say that a transition $(s,s',x/y)$ is an *invertible transition (IT)* if it is the only transition entering state s' that involves input x and output y. A consequence of a transition being invertible is that if a transition involving input x and output y has been executed and this results in the FSM being in state s' it is known that the FSM was previously in state s.

A sequence of transitions $t = t_1 \ldots t_m$, with $t_i = (s_i, s_{i+1}, x_i/y_i)$, is said

to be an *invertible sequence (IS)* if it is the only sequence involving input sequence $x_1 \ldots x_m$ and output sequence $y_1 \ldots y_m$ that ends at s_{m+1}. Clearly an invertible transition is an invertible sequence of length 1.

An IS will be called *prime* if it is not in the form of one non-empty IS followed by another non-empty IS. Prime invertible sequences will be used to reduce the test generation effort. It should be noted that if an IS is not prime, it can be represented as a sequence of prime ISs and this decomposition is unique (Hierons [1997]). An IS is said to be a *minimal (s_i,s_j) IS* if it is a shortest length IS from state s_i to state s_j. Such an IS need not be prime.

An input x is an *invertible input (II)* if every transition involving it is invertible. A sequence of inputs is an *invertible input sequence (IIS)* if every sequence of transitions with this input sequence is an invertible sequence.

Given $F = (S, T, s_1)$ the set of ITs in T is denoted by T_I, $T_R = T \setminus T_I$, and F_I is the machine (S, T_I, s_1). T_{II} is the set of transitions from T that involve invertible input and $F_{II} = (S, T_{II}, s_1)$.

2.2 Some properties of invertible sequences

The following demonstrates that the notion of an invertible sequence is an extension of the notion of an invertible transition.

Lemma 1 *An IS can contain transitions that are not ITs.*

Proof
To demonstrate this, it is sufficient to look at the FSM, taken from Aho et al. [1988], shown in Figure 1. In this FSM the sequence $(v_1, v_2, b/x)(v_2, v_5, a/x)$ is an IS while the transition $(v_2, v_5, a/x)$ is not invertible. □

The following results will be used in the generation of invertible sequences and in the test sequence generation algorithm.

Lemma 2 *If $t = rs$ is and IS (r and s are sequences) then so is r.*

Proof
A proof by contradiction will be produced. Suppose $t = rs$ is an IS and r is not an IS. Then there must be some r' with a different initial state than r that has the same input, output, and final state as r. But then $r's$ has the same input, output, and final state as rs but a different initial state, which contradicts rs being an IS. Thus r must be an IS if rs is an IS. □

Lemma 3 *If r and s are ISs with the final state of r being the initial state of s then rs is an IS*

Proof
As s is an IS, from its final state, input and output its initial state can be identified. Thus the final state of r is known if rs is executed and the final state of rs is known. As r is an IS, from this and the input and output of r the initial state of r is known, which is the initial state of rs. Thus rs is an IS. □

The following will be used in the generation of prime invertible sequences.

Lemma 4 *Any non-empty prime IS starts with an IT and any prime IS of length greater than 1 ends with a transition from T_R.*

Proof
Suppose that t is a non-empty IS. Then $t = rs$ for some transition r, and from Lemma 2 as rs is an IS r is also an IS. Thus, as r is an IS of length 1, r is an IT. Therefore any non-empty IS must start with an IT.

If t has length greater than 1 then $t = r's'$ for some non-empty r' and some transition s'. By Lemma 2, r' is an IS as t is an IS. As t is a prime IS and r' is an IS, s' is not an IS. Thus s' is not an IT and so s' is a transition from T_R. □

Lemma 5 *The following prove that ISs do not have certain intuitively appealing properties.*

1. *There need not be an upper bound on the length of prime ISs*
2. *The number of transitions from T_R in prime ISs is not bounded above*
3. *The existence of a prime IS of length $m+1$ does not imply the existence of a prime IS of length m.*
4. *A prime IS can be in the form rs where r is an IS that is not prime.*

Proof
1)
To show that there need not be an upper bound on the length of prime ISs it is sufficient to prove that, in the FSM given in Figure 2, all sequences of the form $1/x(2/a1/b)^m1/y$ are prime ISs. It is clear that these sequences are

ISs so it is sufficient to prove that they are prime. A proof by contradiction will be produced.

Suppose that some IS $t = 1/x(2/a1/b)^m 1/y$ is not prime, so $t = rs$ for some non-empty ISs r and s. Then s is either $1/y$ or of the form $(2/a1/b)^m 1/y$ or of the form $1/b(2/a1/b)^m 1/y$. But it is clear that the input, output, and final state of $1/y$ allows two possible initial states, s_1 and s_3. It is also clear that any sequence involving the input, output, and final state of $(2/a1/b)^m 1/y$ could have started at either s_1 or s_3, and that any sequence involving the input, output, and final state of $1/b(2/a1/b)^m 1/y$ could have started at either s_2 or s_4. Thus if $t = rs$, r non-empty, then s is not a non-empty IS, and so every sequence in the form $1/x(2/a1/b)^m 1/y$ is a prime IS. □

2)

In order to demonstrate that the number of transitions from T_R in prime ISs is not bounded above, it is sufficient to alter the above example in order to make the transition from s_1 to s_2 non-invertible. In order to do this it is sufficient to change the transition from s_6 with input 2 to give output a and go to state s_2 Thus given any $m \geq 2$ there is a prime IS $1/x(2/a, 1/b)^m 1/y$ with m elements from T_R. □

3)

The FSM given in Figure 2 is again considered. Any prime IS of length greater than one must end in an element of T_R, the only such elements being the transitions associates with $1/y$. Sequences of the form $(2/a1/b)^m 1/y$ or of the form $1/b(2/a1/b)^m 1/y$ are not ISs. Thus the only prime ISs in the FSM in Figure 2 of length greater than one are those of the form $1/x(2/a1/b)^m 1/y$ or of the form $2/z(2/a1/b)^m 1/y$. Thus the prime ISs are either of length one or are of even length, and thus for each $m > 1$ there is a prime IS of length $2m$ but no prime IS of length $2m - 1$. □

4)

In the FSM shown in Figure 2 each sequence t of the form $1/x(2/a1/b)^m 1/y$ or $2/z(2/a1/b)^m 1/y$ is a prime IS. For any non-empty r, s with $t = rs$ and $|r| > 1$, r is a non-prime IS as it has length at least 2 and all of its elements are ITs. □

These results show that it is not, in general, possible to find all prime ISs and even if there is a finite number of prime ISs it is difficult to know when to stop searching. Clearly there are bounds on the size of minimal (s_i, s_j) ISs but these may be large.

2.3 Invertible sequences related to UIOs

This section contains results that show how invertible sequences can be used in the generation of UIOs and DSs, and in solving certain decision problems.

Lemma 6 *Every UIO is an IS.*

Proof
This is follows from the definition of UIOs, as from the input and output of the sequence the initial state is identified. □

It should be noted that while every UIO is an IS, not every IS is a UIO.

Corollary 1 *Every UIO starts with an IT.*

Proof
This follows from Lemma 6 and Lemma 4 which state that every UIO is a non-empty IS and every non-empty IS starts with an IT.

□

The following result shows that it is possible to use ISs to extend the set of UIOs.

Lemma 7 *If t_1 is an IS and t_2 is a UIO starting at the final state of t_1 then t_1t_2 is a UIO for the initial state of t_1.*

Proof
Let s_1 and s_2 denote the initial states of t_1 and t_2 respectively. If t_2 is executed from s_2 the state s_2 is identified, as t_2 is a UIO. Thus, if t_1t_2 is executed from s_1 the intermediate state s_2 is identified. But, as t_1 is an IS, and its final state s_2 is known as well as its input and output, its initial state s_1 is known. Thus, executing t_1t_2 identifies its initial state s_1 and so t_1t_2 is a UIO. □

Lemma 8 *Let r be a minimal length distinguishing sequence for some FSM F, and let the first element of r be x. Then x is an II and there are states s_i and s_j such that $\lambda(s_i, x) \neq \lambda(s_j, x)$.*

Proof
As a DS is a UIO for every state, x must be an IT from each state. Therefore x is an II.

For the second part there are two cases:

Case 1: There is a pair of states (s_i,s_j) such that $\delta(s_i, x) = \delta(s_j, x)$. In this case $\lambda(s_i, x) \neq \lambda(s_j, x)$ as x is an II.
Case 2: The input x does not map any states together. In this case it must permute the states. If x produces the same output from all states and $r = xr'$ then r' must distinguish these states and thus must itself be a DS. This contradicts the minimality of r. Thus x cannot produce the same output value for every state. □

The above results provide necessary, but not sufficient, conditions for an FSM to have a DS and for a state to have a UIO. It is thus possible to eliminate some FSM/states immediately. The results also reduce the options for the first input and so reduce the size of the search space required when looking for a DS or UIOs.

Lemma 9 *F_I being strongly connected does not imply that each state of F has a UIO.*

Proof
This can be seen by looking at the FSM in Figure 3 which is clearly minimal. In this FSM the only ITs are those involving input x. While these strongly connect the states they simply permute the states giving constant output.

As any UIO must start with an IT, UIOs must be in the form x^m $(m > 0)$ followed by y or z and some sequence. But the application of y or z collapses pairs of states, as y sends S_2 and S_4 to the same state with output 2 and sends S_1 and S_3 to the same state with output 1, while z sends S_1 and S_2 to the same state with output 1 and sends S_3 and S_4 to the same state with output 2.

Thus, as the application of x^m simply permutes the states with constant output, a sequence of the form x^m followed by either y or z cannot be an IS and thus the only ISs are of the form x^m. Therefore, as every UIO is an IS and sequences of the form x^m cannot be UIOs, the FSM cannot have a UIO for any state. □

Lemma 10 *If F_I is strongly connected and some state of F has a UIO then every state of F has a UIO.*

Proof
Give a UIO u for state s of F, in order to generate a UIO for state $s' \neq s$ it is sufficient to take a path p from s' to s in F_I and follow it by u. Such a path p must exist, as F_I is strongly connected, and is an IS. Thus, by Lemma 7, pu is a UIO as required. □

Corollary 2 *If F_I is strongly connected then either every state of F has a UIO or no state of F has a UIO.*

Proof
This follows directly from Lemma 10. □

Lemma 11 *If F_{II} is minimal then F has a DS of length at most $|S|^2$.*

Proof
Take some pair of states s_1 and s_2. As F_{II} is minimal there is some sequence r_1, $|r_1| \leq |S|$, of inputs from F_{II} that distinguishes between s_1 and s_2. The sequence r_1 induces an equivalence relation $\sim_{r_1}$ on S that is defined by: $s_i \sim_{r_1} s_j$ if and only if $\lambda^*(s_i, r_1) = \lambda^*(s_j, r_1)$. Clearly, as the values in r_1 are from F_{II}, if $s_i \sim_{r_1} s_j$ then $\delta^*(s_i, r_1) \neq \delta^*(s_j, r_1)$.

If there is some pair of states (s_i,s_j) such that $s_i \sim_{r_1} s_j$ then there is some sequence r_2, $|r_2| \leq |S|$, from F_{II} that distinguishes between $\delta^*(s_i, r_1)$ and $\delta^*(s_j, r_1)$. Then $r_1 r_2$ induces an equivalence relation on S and this has at least one more equivalence class than $\sim_{r_1}$.

This process can be repeated until there is some sequence $r = r_1 r_2 \ldots r_k$ with $|S|$ equivalence classes. Then clearly $k \leq |S|$ and $|r_i| \leq |S|$, for $1 \leq i \leq k$, and thus $|r| \leq |S|^2$. As $\sim_r$ has $|S|$ equivalence classes it is a DS. Thus F has a DS of length at most $|S|^2$. □

It should be noted that the above proof suggests an algorithm for generating DSs of length at most $|S|^2$ when F_{II} is minimal. This upper bound is useful, as there is no polynomial upper bound on the length of DSs or UIOs (Lee and Yannakakis [1994]), although it has been suggested that DSs and UIOs are typically short (Hennie [1964], Shen et al. [1990]).

2.4 Finding invertible sequences and UIOs

Given an FSM $F = (S, T, s_1)$ there are two approaches to finding ISs, either searching forward starting with invertible transitions or searching backwards from non-invertible transitions, as a prime IS of length greater than one starts with a transition from T_I and ends in a transition from T_R. If the set of non-invertible transitions, T_R, is much smaller than the set of invertible transitions, T_I, it can be advantageous to search backwards in order to find the shorter ISs, as there will be far fewer starting transitions for the search. In general, however, it is better to search forward starting with elements of T_I, as

when searching forward any non-invertible sequence can be eliminated from the search. This is because, by Lemma 2, a sequence t being non-invertible implies that for any sequence r, tr is also non-invertible. In contrast, when searching backwards non-invertible sequences cannot be eliminated from the search as it is possible that they can be extended backwards to produce invertible sequences.

The forward search for ISs can be performed using, at the $(m+1)$th step, a set of ISs of length m and for each of these ISs the set of other final states that can be reached with the same input and output sequence. The set of ISs of length m will be denoted I_m and for each $t = t_1 \dots t_m$ in I_m, $t_i = (s_{\sigma(i)}, s_{\sigma(i+1)}, x_i/y_i)$, for some function $\sigma : \{1, \dots, m\} -> \{1, \dots, n\}$,

$S_t = \{\delta^*(s', x_1 \dots x_m) \mid s' \in S - s_{\sigma(1)} \wedge \lambda^*(s', x_1 \dots x_m) = y_1 \dots y_m\}$

This is the set of other final states that can be reached with this input and output.

Then $I_1 = T_I$ and for each $t = (s_i, s_j, x/y)$ in T_I:

$$S_t = \{s \neq s_j \mid \exists s' \bullet (s', s, x/y) \in T\}$$

Both I_{m+1} and the S_t can be defined inductively by:

$$I_{m+1} = \{t_1 \dots t_{m+1}, t_i = (s_{\sigma(i)}, s_{\sigma(i+1)}, x_i/y_i) \mid t_1 \dots t_m \in I_m \wedge ((s, s_{\sigma(m+2)}, x_{n+1}/y_{n+1}) \in T \Rightarrow s \notin S_{t_1 \dots t_m})\}$$

$$S_{t_1 \dots t_{m+1}} = \{s \mid \exists s' \in S_{t_1, \dots, t_m} \bullet (s', s, x_{m+1}/y_{m+1}) \in T\}$$

It should be noted that if $S_t = \{\}$ then t is a UIO and so this method can be used to find UIOs. The searching of the set of ISs when looking for UIOs has the advantage over the direct approach, as described in Sabnani and Dahbura [1988], that it limits the size of the search. Thus, as a sequence that is not an IS cannot be extended to form an IS, any extensions of a sequence that is not an IS can be eliminated from the search.

As is noted in Sabnani and Dahbura [1988], for testing it is only necessary to look for UIOs of length at most $2|S|^2$. This is because every FSM has a characterizing set and it is possible to test with effort at most $2|S|^2$ using a characterizing set. As ISs will be used to avoid using UIOs, only ISs of length at most $2|S|^2$ need be generated.

3 Testing from an FSM

3.1 Introduction

In order to test against an FSM model it is necessary to check the transitions. Testing a transition involves moving to its initial state, executing the transition, and then checking the final state. In this paper it will be assumed that any FSM used has a UIO for each state and that the problem is to find the shortest sequence that contains a test for every transition. See e.g. Chow [1978], Fujiwara et al. [1991], Petrenko et al. [1994b] for information on testing from an FSM model that does not have a UIO for each state.

It has been noted that the conditions placed on the FSM can be weakened. The problem of testing from a nondeterministic FSM has been considered (Fujiwara et al. [1991], Fujiwara and v. Bochmann [1992], Evtushenko et al. [1991]). Petrenko et al. [1994b] further weaken the conditions assumed by introducing a test technique that uses a characterizing set and does not require the FSM model to be either deterministic or completely specified. Tripathy and Naik [1992] extended the idea of a UIO to a non-deterministic FSM by using an adaptive identification process.

When producing a test sequence that tests the individual transitions by using UIOs, each transition t is tested by a sequence of the form tu, where u is a UIO for the final state of t. Such sequences will be called *test subsequences*. If a sequence v contains a test subsequence for each transition, v is said to be a *test sequence*. The problem is to find the *shortest test sequence*.

Aho et al. [1988] express the problem of finding a test sequence as that of minimally connecting the test subsequences. They represent the FSM by a digraph and for each test subsequence tu they add an edge from the initial state of t to the final state of u. They look for the shortest sequence, in the digraph, that contains every test subsequence. This problem corresponds to the *Rural Chinese Postman Problem (RCPP)*. While the RCPP is known to be NP-complete (Lenstra and Rinnooy Kan [1976]), Aho et al. [1988] apply a low order polynomial algorithm that solves the problem if either the FSM has *reset capacity* (there is an input that takes every state to the initial state) or has *loops* (transitions with equal initial and final states) for each state.

Shen et al. [1990] note that a state may have more than one UIO and that shorter test sequences can be produced by an appropriate choice of UIO.

Yang and Ural [1990] utilize overlap between test subsequences. They look for pairs of test subsequences t_1 and t_2 with the property that t_1 can

be extended to be of the form of a single transition followed by t_2. More formally, there exists a transition t_0 and a (possibly empty) sequence t'_2 such that $t_1 t'_2 = t_0 t_2$. Thus when $t_1 t'_2$ is executed the first two transitions are tested using only one UIO. They build sequences from overlapping test subsequences and connect these sequences. While this can reduce the length of the test sequence, it need not be optimal as it does not include a method for finding the choice of sequences that leads to the shortest test.

Hierons [1996] proves that this form of overlap is fully represented by the invertibility of transitions, as this overlap exists if and only if the first transition of t_2 is an IT. Invertible transitions can also be used to extend the set of UIOs as, by Lemma 7, if t is an invertible transition and u is a UIO for the final state of t then tu is a UIO.

A more general form of overlap is where there are two test subsequences t_1 and t_2 such that t_1 ends with some initial section of t_2. More formally, there exist sequences t'_1 and t'_2 (t'_2 is non-empty) such that $t_1 t'_1 = t'_2 t_2$ and $|t'_2| < |t_1|$. If the sequence $t_1 t'_1$ is executed the first transition of t_1 and the first transition of t_2 are both tested. The following results demonstrate that this form of overlap exists if and only if $t'_2 t_2$ is in the form of a transition followed by an IS followed by a UIO, and thus that if a transition is followed by an IS and then a UIO both the initial transition and the last transition of the IS are tested. This shows that ISs fully represent this more general form of overlap.

Theorem 1 *If there exist test subsequences t_1 and t_2 such that there are (possibly empty) sequences t'_1 and t'_2 and transitions t and t' with $t_1 t'_1 = t t'_2 t_2$, $t_1 = t u_1$, $t_2 = t' u_2$, and $|t_1| > |t t'_2|$ then $t'_2 t'$ is an IS.*

Proof
As $t t'_2 t'$ is contained in the beginning of the test subsequence t_1, $t'_2 t'$ is contained in the beginning of the UIO u_1. By Lemma 6 u_1 is an IS. Also, by Lemma 2, if rs is an IS then r is an IS and thus, as $t'_2 t'$ is contained in the beginning of the IS u_1, $t'_2 t'$ must be an IS. □

Theorem 2 *If there exists a test subsequence t_1, sequence t_2, and transitions t and t' such that the final state of t is the initial state of t_2, $t_1 = t' u$, and $t_2 t'$ is an IS then $t t_2 t_1$ is a test subsequence for t that overlaps with the test subsequence t_1.*

Proof
As t_1 is a test subsequence, u is a UIO. The sequence $t_2t'u = t_2t_1$ is therefore in the form of an IS followed by a UIO and so, by Lemma 7, is a UIO. Thus tt_2t_1 is a test subsequence. □

This link between ISs and test subsequence overlap will be utilized in order to reduce the test sequence length. The use of this, and the use of ISs to give more UIOs, will now be described in detail.

3.2 Invertible sequences and Testing

It has been shown that ISs can be used both to represent test subsequence overlap and to extend the set of UIOs. An IS can therefore play two separate roles in testing: either allowing the final state of its last transition to be verified (and thus testing it without using an extra UIO) or connecting tests. An algorithm, based on graph and network theory, that utilizes these properties will now be given. This will extend the algorithm given in Hierons [1996] by using ISs. It will also allow transitions from T_I to be tested as if they were from T_R; this extends the applicability of the algorithm as in some cases it is not possible to utilize the invertibility of all of the elements of T_I. The algorithm will be divided into 3 steps.

3.2.1 Step 1

Given an FSM $F = (S, T, s_1)$ ($|S| = n$), represented by a digraph G, the transition sets T_I and T_R are produced. From this a network N, with vertex set $V' = W \cup X \cup Y \cup Z \cup \{s, t\}$ in which the source is s and the sink is t, is produced. This network is shown in Figure 4. In Step 3 edges from Z to W, representing the transitions being tested, will be added and a tour generated.

The vertex set W represents the final states of transitions being tested, the set X represents the initial states of transitions to be tested as non-invertible transition, and the set Y represents the initial states of transitions to be tested as invertible transitions. The sets X and Y are connected to the set Z which represents the initial states of transitions being tested. This stage of the algorithm involves producing a min cost max flow for N, whose edges will now be described.

The capacity of the edge from s to w_i ($1 \leq i \leq n$) is $indegree_T(s_i)$ and the capacity of the edge from $z_i \in Z$ to t is $outdegree_T(s_i)$. The flow from each y_i to the corresponding z_i is limited to $outdegree_{T_I}(s_i)$, as this is the maximum

number of transitions leaving s_i that can be tested as invertible transitions. For each i, $1 \leq i \leq n$, there is an edge from w_i to y_i with infinite capacity. The flow from each x_i to the corresponding z_i is not limited as it may be necessary to test some transitions from T_I as if they were not invertible. None of these edges has a cost, as each corresponds to the execution of a transition being tested; in testing every transition is executed in this manner.

Given a *prime* IS of the form t_0x (non-empty sequence t_0 and final transition x) in which the initial state of t_0 is s_i and the final state of t_0 (and thus the initial state of x) is s_j an edge from w_i to z_j,with cost $|t_0|$ and capacity 1, is included. This edge represents testing x by executing the IS t_0x and later verifying its final state, which is why it has capacity 1 and provides flow of 1 to z_j. Prime ISs are used as any non-prime IS can be produced from this and it is vital that the elements tested in this manner are from T_R (Lemma 4 tells us that prime ISs of length greater than 1 end in elements from T_R) as otherwise the capacity from y_j would need reducing.

The edges from W to X represent the UIOs and thus for each UIO with initial state s_i, final state s_j, and length m there is an edge from w_i to x_j with cost m. Edges between the vertices of X represent executing transitions in order to get to the initial state of a transition from T_R and thus a copy of each transition from T is included and give infinite capacity and cost 1.

The edges between the vertices of W represent transitions joining testing sequences and thus must be invertible. A copy of the elements from T_I and the set of known prime ISs is therefore included; each is given capacity infinity and the cost is the length of the sequence (clearly 1 for individual transitions).

A max flow/ min cost F' for N is now found. The flow can be seen as a set of transitions/sequences that can be executed by following edges from the flow plus edges from Z to W representing the transitions (these replace the flow from s and to t). The max flows will represent the set of sequences that contain a test for each transition, and for a max flow the corresponding test has length $|T|$ plus the cost of the flow. From F' a symmetric digraph G' will be produced and an Euler Tour of G' will give the test sequence (this process will be described in Step 3).

3.2.2 Step 2

If the full flow from Y is used in F', the algorithm now goes to Step 3. If, however, some of the transitions from T_I are tested as if they were transitions

from T_R (i.e. the capacity of the edges from Y is not fully used) it is necessary to determine which transitions from T_I are to be treated in this manner: the extra flow leaving some x_i must be associated with the extra flow from W to X. Some set $A \subseteq T_I$ of transitions, whose testing as elements of T_R is consistent with the flow F', is found.

The set A is found by producing a max flow for a network N' with vertex set $V'' = \{s, t\} \cup B \cup C$, where s is the source, t is the sink, each vertex in B corresponds to the initial state of a transition, and each vertex in C corresponds to the final state of a transition. For each transition in T_I that goes from s_i to s_j an edge from b_i to c_j with capacity 1 is included. For each w_i with flow $outdegree_{T_R}(s_i) + e_i$ to X in F' an edge from c_i to t with capacity e_i is included. For each x_i with flow $outdegree_{T_R}(s_i) + f_i$ to z_i in F' there is an edge from s to b_i with capacity f_i. The network is shown in Figure 5. A max flow for this network gives a set of edges from T_I whose treatment as non-invertible will allow a tour associated with the flow F'.

3.2.3 Step 3

Having found the set A and the set A' of transitions tested as part of an IS, it is possible to produce the graph $G' = (V''', E''')$, $V''' = P \cup Q$, shown in Figure 6. Effectively the vertices in P represent the situation after executing a UIO and before executing a transition from T_R while the vertices from Q represent the situation before executing a UIO and thus the edges between the q_i must be invertible (ISs or ITs).

The edges will represent transitions or sequences of transitions involved in testing and an Euler Tour will represent the test sequence. For each transition that is to be tested as non-invertible and that is not tested as part of an IS, from state s_i to state s_j, there is a corresponding edge from p_i to q_j. This transition set is $T_R \cup A - A'$. For each UIO from state s_i to s_j with flow m in F' there are m edges from q_i to p_j; each represents the execution of this UIO. For each transition in $T_I - A$ from state s_i to s_j an edge from q_i to q_j is included and for each transition $x \in A'$, tested as part of a IS t_0x with initial state s_i and final state s_j, there is an edge from q_i to q_j.

For each unit of flow from w_i to w_j in F' there is an edge from q_i to q_j representing this IT or IS. For each unit of flow from x_i to x_j in F' a corresponding edge from p_i to p_j is added.

Suppose W is a walk that covers every edge of G'. In W a non-invertible transition, that is not tested as part of an invertible sequence, is represented

by an edge to Q and thus is followed by a number of ISs and ITs and then finally a UIO. Similarly any transition that is either being tested as an IT or as part of an IS will be followed by a number of ISs and ITs and then a UIO. Thus W will contain a test for every transition.

It is easy to verify that, as flow is conserved in a network, this graph is symmetric. An Euler Tour of G' can therefore be found as long as G', with the isolated vertices removed, is connected. Possible approaches to dealing with G' being disconnected will be discussed in Section 3.3.

The Euler Tour of G', with each edge replaced by the corresponding transition or sequences of transitions, gives the test sequence, of length $cost(F') + |T|$, unless it does not include a UIO; in this case a UIO can be added to the end. The algorithm will be applied to an example in Section 3.4.

3.3 The connectivity of G'

It is possible for the digraph G' to be symmetric but, even with the isolated vertices removed, disconnected. If this is the case G' does not have an Euler Tour, though an Euler Tour can be produced for each component. As a tour of the whole digraph is required it is necessary to add edges to connect G' while maintaining its symmetry. This can be done by adding circuits to G'. It is important to connect these tours at the correct points, which are the sections that lie after the execution of a UIO and before the next execution of a transition to be tested. These correspond to vertices in P.

Clearly it is desirable to find the smallest set of circuits, in terms of total number of transitions, that connects G'. One approach is to initially find the pair of components that requires the shortest circuit to connect it and add this circuit forming a new graph G'_1. This process is repeated until some connected G'_r is found. An Euler Tour of G'_r provides the test sequence.

The advantage of this rather naive algorithm is that its computational complexity is low. Unfortunately, however, the solution need not be optimal, but this is to be expected as the problem of minimally connecting the components is NP-complete. An alternative approach is given in Ural et al. [1997].

3.4 Example

The algorithm outlined in Section 3.2 will now be applied to the FSM F with state set $S = \{s_1, s_2, s_3, s_4, s_5\}$ input alphabet $\Sigma = \{a, b, c\}$, output alphabet $\Theta = \{x, y\}$, and whose transitions are given in Table 1. The entries in Table 1 give the output and next state for the initial state and input given by the row and column respectively. The sets T_I and T_R are given in Tables 2 and 3 respectively. The set of UIOs to be used is given in Table 4 - these are the shortest UIOs for each state.

	a	b	c
s_1	x, s_2	x, s_3	x, s_5
s_2	x, s_4	x, s_5	x, s_3
s_3	x, s_4	x, s_5	y, s_5
s_4	x, s_1	x, s_5	y, s_2
s_5	x, s_1	x, s_5	x, s_4

Table 1: the FSM F

	a	b	c
s_1	x, s_2	x, s_3	x, s_5
s_2			x, s_3
s_3			y, s_5
s_4			y, s_2
s_5			x, s_4

Table 2: the set T_I

	a	b	c
s_1			
s_2	x, s_4	x, s_5	
s_3	x, s_4	x, s_5	
s_4	x, s_1	x, s_5	
s_5	x, s_1	x, s_5	

Table 3: the set T_R

	UIO	Final State
s_1	$b/x, c/y$	s_5
s_2	$c/x, a/x, c/y$	s_2
s_3	$c/y, a/x, c/x$	s_5
s_4	$c/y, a/x, c/y$	s_2
s_5	$c/x, c/y, a/x, c/y$	s_2

Table 4: the UIOs

There are a number of prime ISs. The ones to be use, and their intermediate states, are given in Table 5.

t_0	x
$s_3 - c/y -> s_5$	$s_5 - a/x -> s_1$
$s_5 - c/x -> s_4$	$s_4 - a/x -> s_1$
$s_2 - c/x -> s_3$	$s_3 - a/x -> s_4$
$s_4 - c/y -> s_2$	$s_2 - a/x -> s_4$

Table 5: the ISs

The algorithm produces the network and min cost/ max flow F' shown in Figure 7, in which only the edges with non-zero flow are shown. The flow F' has cost 23 and thus the test sequence produced has length $23 + 15 = 38$. If ISs are not used, but ITs are, a test sequence of length 54 is produced.

The symmetric graph G' is defined by:

Vertex set $V = \{p_1, p_2, p_3, p_4, p_5, q_1, q_2, q_3, q_4, q_5\}$.

The edges are:

1. Corresponding to A': $q_3 - c/y$, $a/x- > q_1$, $q_5 - c/x, a/x- > q_1$, $q_2 - c/x, a/x- > q_4$, $q_4 - c/y, a/x- > q_4$
2. Corresponding to $T_R - A'$: $p_2 - b/x- > q_5, p_3 - b/x- > q_5, p_4 - b/x- > q_5$, $p_5 - b/x- > q_5$
3. Corresponding to the UIOs: $q_1- > p_5$, $q_1- > p_5$, $q_4- > p_2$, $q_5- > p_2$
4. Corresponding to T_I: $q_1 - a/x- > q_2$, $q_1 - b/x- > q_3$, $q_1 - c/x- > q_5$, $q_2 - c/x- > q_3$, $q_3 - c/y- > q_5$, $q_4 - c/y- > q_2$, $q_5 - c/x- > q_4$
5. Corresponding to connecting ISs: $q_5- > q_1$, $q_5- > q_1$, $q_5- > q_1$
6. Corresponding to connecting transitions between the x_i: $p_2 - c/x- > p_3$, $p_5 - c/x- > p_4$

It is easy to check that this digraph, G', is symmetric. As G', with the isolated vertex p_1 removed, is connected an Euler Tour can be produced, as required. This tour, in which $UIO_{i,j}$ denotes the UIO from state s_i to state s_j and IS denotes an IS used to connect testing, is:

$$
\begin{aligned}
p_2 - b/x &\rightarrow q_5 - c/x, a/x \rightarrow q_1 - a/x \rightarrow q_2 - c/x \rightarrow q_3 - c/y \rightarrow q_5 \\
q_5 - c/x &\rightarrow q_4 - c/y \rightarrow q_2 - c/x, a/x \rightarrow q_4 - c/y, a/x \rightarrow q_4 - UIO_{4,2} \rightarrow p_2 \\
p_2 - c/x &\rightarrow p_3 - b/x \rightarrow q_5 - IS \rightarrow q_1 - c/x \rightarrow q_5 - IS \rightarrow q_1 \\
q_1 - b/x &\rightarrow q_3 - c/y, a/x \rightarrow q_1 - UIO_{1,5} \rightarrow p_5 - b/x \rightarrow q_5 \\
q_5 - IS &\rightarrow q_1 - UIO_{1,5} \rightarrow p_5 - c/x \rightarrow p_4 - b/x \rightarrow q_5 - UIO_{5,2} \rightarrow p_2
\end{aligned}
$$

3.5 A comparison with other techniques

There are a number of techniques that aim to generate a test sequence that includes a test for every transition (Aho et al. [1988], Yang and Ural [1990], Hierons [1996]). The algorithm outlined in Section 3.2 subsumes the algorithm given in Hierons [1996] and, as it allows invertible transitions to be tested as transitions from T_R, is generally more applicable. The example given in Section 3.4 demonstrates that the algorithm outlined in this paper can lead to a shorter test sequence than that given in Hierons [1996] and clearly it can never produce a longer test sequence. The algorithm given in Hierons [1996] subsumes those of Yang and Ural [1990] and Aho et al. [1990]

and thus again cannot produce a longer test sequence than these. It is also important to note that all of these algorithms have the same computational complexity as they are based on network optimization for networks of the same order.

It is more difficult to compare the algorithm given in this paper with different classes of algorithm, such as the W and Wp algorithms (Chow [1978], Fujiwara et al. [1991]). The worst case behaviour of the W and Wp methods is certainly better than those based on the use of UIOs or a DS, as there is no polynomial upper bound on the length of UIOs and DSs (Lee and Yannakakis [1994]). It has, however, been noted that UIOs are usually quite short and thus that the tests produced using UIOs are typically much shorter than those used producing the W method (Sidhu and Leung [1988]) and, presumably, the Wp method. This is because, when using a characterizing set, it is necessary to execute each transition a number of times.

It is important to note that the problem of producing a checking sequence has not been addressed in this paper. In order to produce a checking sequence it is necessary to verify the UIOs used, and thus the use of multiple UIOs for each state may not reduce the total length of a checking sequence.

4 Conclusions

Invertible sequences are strongly linked to state identification sequences and can be utilized in generating a set of UIOs or a DS. If the FSM F_{II}, formed by taking the transitions given by invertible inputs, is minimal it is known that F has a DS of length at most $|S|^2$ and an algorithm for generating this DS has been outlined.

Certain properties of ITs help us decide whether an FSM has a DS or UIOs for each state. In particular, if F_I is strongly connected then either F has a UIO for every state or no state of F has a UIO. If some state of an FSM has no ITs leaving it then the state does not have a UIO.

Invertible sequences can be used to connect transitions that are being tested without losing information about the state; if the final state of an IS is known then so is its initial state. If the final state of the IS has been verified, the last transition of the IS and the transition that preceded the IS have both been tested. This can help reduce the number of UIOs needed in testing, and thus reduce the length of the test sequence produced without increasing the computational complexity of the algorithm.

The algorithm outlined in this paper generates shorter test sequences when it is simply necessary for there to be a test for every transition. It does not, however, produce a checking sequence. In order to produce a checking sequence a further sequence must be added. This extra sequence may be longer for methods, such as this, that use multiple UIOs.

5 References

1. A.V. Aho, A.T. Dahbura, D. Lee, and M.U. Uyar, 1988, An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours *Proceedings of Protocol Specification, Testing, and Verification VIII*, pp75-86, Atlantic City, North-Holland.

2. E. Brinksma, 1988, A Theory For The Derivation of Tests, *Proceedings of Protocol Specification, Testing, and Verification VIII*, pp63-74, Atlantic City, North-Holland.

3. T.S. Chow, 1978, Testing Software Design Modelled by Finite State Machines, *IEEE Transactions on Software Engineering*, **4** 3, March 1978, pp178-187.

4. N.V. Evtushenko, A.V. Lebedev, and A.F. Petrenko, 1991, On Checking Experiments With Nondeterministic Automata, *Automatic Control and Computer Sciences*, **6**, pp81-85.

5. S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, 1991, Test Selection Based on Finite State Models, *IEEE Transactions on Software Engineering*, **17** 6, June 1991, pp591-603.

6. S. Fujiwara and G. v. Bochmann, 1992, Testing Non-deterministic State Machines with Fault Coverage, *Proceedings of Protocol Test Systems, IV*, pp267-280.

7. A. Gibbons, 1985, *Algorithmic Graph Theory*, Cambridge University Press.

8. F.C. Hennie, 1964, Fault-detecting experiments for sequential circuits, *Proceedings of Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, November 1964, pp95-110.

9. R.M. Hierons, 1996, Extending Test Sequence Overlap by Invertibility, *The Computer Journal*, **39** 4, pp325-330.

10. R.M. Hierons, 1997, Invertible Sequences and State Identification, *Goldsmiths Mathematics and Computing Technical Report #970701.*

11. C-M Huang and J-M Hsu, 1994, An Incremental Protocol Verification Method, *The Computer Journal*, **37** 8, pp698-710.

12. Z. Kohavi, 1978, *Switching and Finite State Automata Theory*, McGraw-Hill.

13. I. Kohavi and Z. Kohavi, 1968, Variable-Length Distinguishing Sequences and Their Application to the Design of Fault-Detection Experiments, *IEEE Transactions on Computers*, August 1968, pp792-795.

14. D. Lee and M. Yannakakis, 1994, Testing Finite-State Machines: State Identification and Verification, *IEEE Transactions on Computers*, **43** 3, pp306-320.

15. J.L. Lenstra and A.H.G. Rinnooy Kan, 1976, On General Routing Problems, *Networks*, **6**, pp273-280.

16. G. Luo and J. Chen, 1989, Generating Test Sequences For Communication Protocol Modelled by CNFSM, *Proceedings of 3rd Pan Pacific Computing Conference*, pp688-694.

17. G. Luo, G. v. Bochmann, and A. Petrenko, 1994a, Test Selection Based on Communicating Nondeterministic Finite-State Machines Using a Generalized Wp-Method, *IEEE Transactions on Software Engineering*, **20** 2, pp149-161.

18. G. Luo, A. Das, and G. v. Bochmann, 1994b, Generating Tests For Control Portion of SDL Specifications, *Proceedings of Protocol Test Systems, VI (C-19)*, pp51-66.

19. E.P. Moore, 1956, Gedanken-Experiments, in *Automata Studies*, Editors C. Shannon and J. McCarthy, Princeton University Press, pp129-153.

20. H. Motteler, A. Chung, and D. Sidhu, 1994, Fault Coverage of UIO-based Methods for Protocol Testing, *Proceedings of Protocol Test Systems, VI (C-19)*, pp21-33.

21. A. Petrenko, G. v. Bochmann, and R. Dssouli, 1994a, Conformance Relations and Test Derivation, *Proceedings of Protocol Test Systems, VI (C-19)*, pp157-178.

22. A. Petrenko, N. Yevtushenko, A. Lebedev, and A. Das, 1994b, Nondeterministic State Machines in Protocol Conformance Testing, *Proceedings of Protocol Test Systems, VI (C-19)*, pp363-378.

23. A. Petrenko, N. Yevtushenko, G. v. Bochmann, and R. Dssouli, 1996, Testing in Context: Framework and Test Derivation, *Computer Communications*, **19**, pp1236-1249.

24. M. Phalippou, 1993, The Limited Power Of Testing, *Proceedings of Protocol Test Systems, V (C-11)*, pp43-54.

25. A. Rezaki and H. Ural, 1995, Construction of checking sequences based on characterization sets, *Computer Communications*, **18** 12, pp911-920.

26. K. Sabnani and A. Dahbura, 1988, A Protocol Test Generation Procedure, *Computer Networks*, **15** 4, pp285-297.

27. Y.N. Shen, F. Lombardi, and A.T. Dahbura, 1990, Protocol Conformance Testing Using Multiple UIO Sequences, *Proceedings of Protocol Specification, Testing, and Verification IX*, pp131-143, Twente, Netherlands, North-Holland.

28. D. Sidhu and T. K. Leung, 1988, Experience with Test Generation for Real Protocols, *ACM SIGCOMM 88*, pp257-261.

29. P. Tripathy and K. Naik, 1992, Generation of Adaptive Test Cases From Non-deterministic Finite State Models, *Proceedings of the 5th International Workshop on Protocol Test Systems*, Sept 1992, Montreal, pp309-320.

30. B. Yang and H. Ural, 1990, Protocol Conformance Test Generation Using Multiple UIO Sequences with Overlapping, *ACM SIGCOMM 90: Communications, Architectures, and Protocols*, Sept 24-27 p118-125, Twente, Netherlands, North-Holland.

31. H. Ural, X. Wu, and F. Zhang, 1997, On Minimizing the Lengths of Checking Sequences, *IEEE Transactions on Computers*, **46** 1, pp93-99.

32. C.D. Wezeman, 1989, The CO-OP Method For Compositional Derivation of Conformance Testers, *Proceedings of Protocol Specification, Testing, and Verification IX*, pp145-158, Atlantic City, North-Holland.