Testing from Semi-independent Communicating Finite State Machines with a Slow Environment

Rob M. Hierons ~~Goldsmiths College,~~
~~University of London~~

November 24, 1998

Abstract. Some systems may be modelled as a set of Communicating Finite State Machines with a slow environment. These machines communicate through the exchange of values. While it is possible to convert such a model into one Finite State Machine, from which test cases can be derived, this process may lead to an explosion in the number of states. Alternatively, it is possible to utilize any independence that exists. The problem of producing a minimal test set, in the presence of certain types of independence and unique input/output sequences, can be represented as a variant of the Vehicle Routing Problem. Possible heuristics for solving this problem are outlined and the method is applied to an example.
Keywords: Communicating finite state machine, minimizing testing, unique input/output sequence, vehicle routing problem

1. Introduction

Several classes of system can be modelled as *Finite State Machines (FSMs)* and, in particular, communications protocols ([1]) and control systems ([2]) are often defined in this manner. Given an FSM model that defines the required behaviour of an implementation, it is important to verify the implementation against the FSM. Testing usually forms part of this verification.

Many real systems lead to large FMSs. One approach to tackling the problem of scale is to represent a system as a set of *Communicating Finite State Machines (CFSMs)* ([3], [4], and [5]). The system is, in effect, represented by a set of FSMs that interact. This leads to smaller and more understandable representations.

An CFSM is an FSM with a queue. A set of CFSMs has a *slow environment* if inputs can only be sent to the system when all the queues are empty. This is sufficient to model a number of real systems, such as communications protocols with handshake ([3]). It will be assumed that there is a slow environment and no livelocks.

The use of FSMs to model systems has lead to much interest in the generation of tests from them ([6], [7], [8], and [9]). Where these approaches can be automated there is the potential for much more effective and efficient testing. While it is possible to extend these test methods to CFSMs, a combinatorial explosion can occur. The main theme of this paper is the utilization of certain properties in order to avoid this combinatorial explosion.

In Section 2 FSMs will be introduced and in Section 3 approaches to testing against FSMs will be briefly reviewed. In Section 4 CFSMs will be defined. In Section 5 a method for testing against CFSMs, with certain properties, will be developed. This approach leads to an NP-complete combinatorial problem and heuristics for solving this problem are described in Section 6. The approach is applied to an example in Section 7. The method outlined in Sections 5 and 6 cannot be applied to certain classes of transitions, and alternative approaches for these are briefly discussed in Section 8. Finally, in Section 9, conclusions are drawn.

2. Finite State Machines

A (deterministic) FSM can be represented by a tuple $(S, s_0, \lambda, \delta, X, O)$ in which S is a finite set of *states*, s_0 is the *initial state*, λ is the *output function*, δ is the *next state function*, X is the finite *input alphabet*, and O is the finite *output alphabet*. If the system is executed with input value in while in state s_i a transition occurs producing output $\lambda(s_i, in)$ and moving the machine to state $\delta(s_i, in)$. This transition is defined by the tuple $(s_i, \delta(s_i, in), in/\lambda(s_i, in))$. The functions λ and δ can be extended, to be applied to input sequences, to λ^* and δ^*. Only deterministic FSMs will be considered and these will be referred to as FSMs.

Two states s_i and s_j are *equivalent* if, for each input sequence, they produce the same output sequence. An FSM is *minimal* if no two states in it are equivalent and two FSMs are *equivalent* if their initial states are equivalent. See [10] for more information on FSMs.

A *directed graph (digraph)* is given by a *vertex set* and an *edge set*. Each edge is defined by its initial vertex and its final vertex and may have a label. A digraph is said to be *strongly connected* if for any pair (v_i, v_j) of vertices there is some path from v_i to v_j. It will be assumed that any FSM considered is minimal and the corresponding digraph is strongly connected.

3. Testing against a Finite State Machine

When testing an implementation against an FSM F, it is assumed that the implementation behaves like some, unknown, FSM F'. Testing thus involves attempting to determine whether F and F' are equivalent.

In order to test a transition t it is necessary to move to the initial state of t, execute t, and then verify the final state. One approach to verifying a state s is to execute a *Unique Input/Output sequence (UIO)* for s: an input sequence u with the property that, for all $s' \neq s$, $\lambda^*(s', u) \neq \lambda^*(s, u)$.

One test criterion is that every transition is tested by being executed and having its final state verified. It is then desirable to find the shortest test that satisfies this criterion ([7], [8], [9]). We will look at the problem of finding the shortest test, that includes a test for every transition, in the presence of a UIO for every state.

4. Communicating Finite State Machines

4.1. Introduction. The system of CFSMs M that is formed out of $M_1, \ldots, M_k$, with a slow environment, will be denoted $M_1|M_2|...|M_k$. Each M_i has corresponding S_i, s_0^i, X_i, O_i, λ_i and δ_i. It will be assumed that this representation is deterministic, and thus that the X_i are pairwise disjoint.

A transition t from M_i is said to be a *communicating transition (CT)* if the output from t is in the input alphabet of some other machine and otherwise t is a *non-communicating transition (NCT)*. Whenever a machine M_i produces a value that is in the input alphabet of some M_j, M_j accepts this value. Thus, a CT does not directly output a value, but activates a transition in another machine.

A system of CFSMs $N = N_1|N_2|N_3$ is given in Figure 1. If N_1 is in state r_0 and a is input then N_1 moves to r_1 and outputs x: this transition is an NCT. If N_1 is in state r_0 and b is input then N_1 moves to r_2 and produces c. This triggers a transition in N_2 and thus is a CT. The output produced depends upon the state of N_2.

For an input a, two machines M_i and M_j can be combined by linking output of a by M_i to M_j. It is only possible to trigger the transitions in M_j, with input a, by output from M_i.

It may be possible to combine the M_i to form one FSM and test from this. The FSM would, however, have $O(\Pi_i|S_i|)$ states and $O((\Pi_i|S_i|)(\sum_i |X_i|))$ transitions. It is thus often desirable to use any independence that exists within the model in order to reduce the test effort.

4.2. Semi-independence. In order to move M_i to a state s_j it is necessary to execute a number of transitions. When manipulating one machine it is usually desirable to use NCTs. The set of M_i will be called *semi-independent* if the following hold:

1. Each M_i, restricted to NCTs, is strongly connected.
2. For each i and $s \in S_i$ there is a state verification process for s that contains only NCTs.

It will be assumed that any set of M_i considered is semi-independent.

If the CTs from N are removed the N_i are still connected. The UIOs for the N_i are given in Figure 2. While these UIOs do not contain CTs, when executed from another state they can lead to communication. These communications may lead to different output and thus must be considered. The UIO for u_0 cannot be applied if N_2 is in s_2 and the UIO for s_2 cannot be applied if N_1 is in r_2. While the N_i are not semi-independent, it is only necessary to be avoid these situations.

The transition t from M_i is said to *feedback*, in a particular configuration of the other machines, if the execution of t leads to a sequence of communications that includes at least one more transition from M_i. The transition t is said to be a *feedback transition* if t always feedbacks. Further, a transition t is a *weak feedback transition* if the execution of t always leads to a sequence of actions that includes feedback. A transition is said to be a *non-feedback transition* if there is some feedback free sequence containing it.

In the example, the transition in N_1 from r_0 to r_2 with input b creates feedback if N_2 is in state s_0, as N_2 produces output a which triggers a further transition in N_1. It is easy to check that N has no (weak) feedback transitions.

As it is normal to check the final state in testing, when testing a CT it is desirable to avoid feedback as this reduces the effectiveness of the state verification. Feedback transitions and weak feedback transitions will be considered separately in Section 8.

5. Testing against Semi-independent CFSMs

5.1. Testing non-communicating transitions. An NCT t from M_i should not affect any other machine. As the M_i are semi-independent it is possible to test t by moving M_i to the initial state of t, using NCTs, and then executing t followed by a UIO.

If a transition t cannot be executed directly, as it can only receive input from another machine, the shortest sequence of transitions that leads to its execution can be found.

5.2. Testing communicating transitions. If a CT is executed it is necessary to check the final state of each machine affected. It is possible to control the sequence of transitions by setting up the other machines, using NCTs, before executing the transition.

The execution of a CT can lead to a sequence of transitions being executed. If there is no feedback and the sequence is followed by the verification of the final state of the affected machines, each transition in the sequence is tested. The problem of minimizing the test effort thus reduces to finding a minimal cost set of such sequences that covers the CTs. The (weak) feedback transitions cannot be tested in this manner and these will be discussed in Section 8.

Suppose that for each M_i and $x \in X_i$ there is an NCT in M_i with input x and every transition can receive input from outside the system. Then every transition can be included in a sequence that has no feedback.

The problem of finding the minimum cost test can be represented using a graph. Each machine is represented by a set of vertices and the triggering of a transition by another is represented by an edge between the corresponding vertices. There is a special vertex, s, that is not associated with any machine. Possible sequences of CTs will be represented by circuits that include s.

The structure of the graph is outlined in Figure 3 and is summarized by:

1. For each M_i and input value a that can trigger an NCT, there is a corresponding vertex with an edge to s. This represents receiving input a and executing an NCT. The cost is the minimum, over the corresponding transitions, of the cost of testing the transition.

2. For each M_i and CT t (from M_i) there is a vertex representing t. If t can be triggered from outside M there is an edge from s, with cost 0, to this vertex.

3. For each vertex representing a CT t there is an edge to each possible event t' that can be triggered by t. The cost of the edge is the cost of testing t.

Circuits through s represent sequences of communications, ending in an NCT, if no machine is visited more than once. It will be necessary to include this constraint in any optimization procedure. The cost of a circuit (the sum of the costs of edges) gives the number of transitions executed in order to test the corresponding sequence. The minimal cost (covering) set of circuits is required.

It is important to consider the size of the graph. Suppose there are α CTs. For each machine there are at most $|X_i|$ vertices corresponding to NCTs. There are α vertices corresponding to CTs. The number of vertices is therefore of $O(\alpha + \sum_i |X_i|)$.

For each CT t from M_i to M_j there is at most one edge to vertices representing NCTs and there is one edge for each CT from M_j that can be triggered by the output of t. The number of edges corresponding to CTs is therefore of $O(\alpha^2)$. There is one edge from s for every CT and one edge to s for each input value that can lead to an NCT. Thus the number of edges is of $O(\alpha^2 + \sum_i |X_i|)$.

The problem reduces to that of finding a set of circuits that minimizes the total cost, with the constraint that no circuit may visit a machine more than once. This is similar to a *Vehicle Routing Problem* (VRP) (see [11]). The VRP is NP-complete and so heuristics are normally applied. In Section 6 the VRP will be defined and possible heuristics will be discussed.

5.3. Including the non-communicating transitions. The approach outlined tests the NCTs separately, and thus any NCT terminating a tour is tested more than once. This is based on the assumption that when testing an NCT it is best to avoid the execution of CTs. Alternatively, the NCTs can be included in the optimization problem. We will now consider this situation.

If an NCT t terminates a sequence triggered by a CT it need not be tested elsewhere and thus the execution of t can be given cost 0. This is only the case for one use of t. The graph can thus be extended by having two copies of each terminating transition: one with cost 0 and the other with the normal cost. The vertex with cost 0 can only be used once.

The solution to the corresponding VRP will give a set of NCTs that are tested. The others are tested separately.

6. Heuristics

The VRP is: given a set of customers (vertices), each with a demand, a set of edges between them with costs, and a set of vehicles (each with a given capacity) find a set of routes with the following properties:

1. Each customer is visited by exactly one vehicle
2. The total demand on any route is no more than the capacity of the corresponding vehicle
3. It is the cheapest such set of routes.

There are a number of approaches to solving the VRP ([11]) including the use of Tabu Search ([12]) and Genetic algorithms ([13]). The problem outlined in Section 5.2 is similar to a VRP except that more than one vehicle may visit a customer (a vertex representing a CT) and there is a labelling of customers such that no vehicle may visit two customers with the same label. The label represents the machine containing the transition. These differences can easily be incorporated into some of the algorithms devised for the VRP.

The following algorithm is a variant of the *Savings Algorithm* ([14]), outlined in [11]:

The Savings Algorithm starts with each customer being on a separate (minimal) route. For each pair of customers the saving, produced by putting them on the same feasible route, is determined. Initially the two customers, for which the saving is maximal, are combined. Customers are added (without violating the constraints), in order of descending saving, to either end of the current route until no more customers can be added to form a valid route. Another route is then started. The process continues until all customers are covered by routes.

This can be applied to the problem outlined in Section 5.2: it is only necessary to change the constraints that determine whether a route is feasible.

7. Example

These approaches will now be applied to N. The graph for N is given in Figure 4, in which $nc(\alpha)$ denotes an NCT with input α. Those terminating transitions not required are not included and the edges to the vertices representing CTs are not shown. The states used for each termination are given in Figure 5 while the CTs are listed in Figure 6.

There are three non-trivial sequences (i.e. not simply a CT followed by an NCT). These are: $t_6 t_3 nc(a)$, $t_3 t_1 nc(e)$, and $t_1 t_6 nc(c)$. Thus one of these is executed and every other CT is tested by being followed by an NCT. The sequences $t_6 t_3 nc(a)$ and $t_3 t_1 nc(e)$ give the minimum cost and the former produces the test given in Figure 7 in which, for each sequence, the transition being tested is underlined. In Figure 7 only the input and output for N are shown: any values that get passed are not included.

As noted earlier, there are restrictions on the use of some UIOs but these do not affect the test produced. If a test case was affected it might be possible to prefixed it with input to change the state of a machine.

A number of the tests overlap and thus the test set can be reduced. An example is the tests for the NCTs $r_0 - a/x- > r_1$ and $r_2 - a/y- > r_1$.

If the variant of the Savings Algorithm described in Section 6 is applied, the same test sets are produced. There are three possible savings, corresponding to $t_6 t_3 nc(a)$, $t_3 t_1 nc(e)$, and $t_1 t_6 nc(c)$. The routes $t_6 t_3 nc(a)$ and $t_3 t_1 nc(e)$ give the greatest saving and so one of these is chosen. There are no more savings and thus the process terminates.

8. Testing feedback transitions

The existence of a large number of (weak) feedback transitions would suggest that there is a high degree of dependency between the FSMs. In some cases this may suggests that the model has low testability and could indicate a poor representation that will be difficult to understand. When there are many (weak) feedback transitions it may be necessary to apply alternative approaches (see e.g. [4]).

If there are few feedback transitions, the obvious approach is to find, for each such transition t, the shortest sequence of transitions that tests t. Each path can be found using, for example, Dijkstra's algorithm (see e.g. [15]).

9. Conclusions

If a system is described by a set of CFSMs with a slow environment it is possible to create one FSM that represents the behaviour, and test from this. The test size may, however, be reduced by taking advantage of any independence. If the machines are semi-independent, the problem of finding the shortest test set is reduced to three problems: testing the non-communicating transitions, testing the non-feedback communicating transitions, and testing the (weak) feedback transitions.

The problem of minimizing the test effort is simple for the non-communicating transitions. The problem of testing the non-feedback communicating transitions can be represented as a variant of the Vehicle Routing Problem. One heuristic has been given, and alternative approaches have been suggested.

A feedback transition t from M_i can be tested by finding the shortest test sequence that contains t. The feedback clearly reduces the confidence that can be derived from correct test execution. A transition $(s, s', in/out)$ is invertible if it is the only transition with input *in* and output *out* that enters state s' ([9]). Invertible transitions thus preserve state information. One possible solution to the problem of testing a feedback transition from M_i is, when possible, to avoid using non-invertible transitions from M_i.

10. References

1 TANENBAUM, A.S.: 'Computer Networks' (Prentice Hall, International Editions, 1996) 3rd edn.

2 POMERANZ, I. and REDDY, S.M.: 'Test Generation for Multiple State-Table Faults in Finite-State Machines', *IEEE Transactions on Computers*, 1997, **46**, (7), pp. 783-794

3 LUO, G., VON BOCHMANN, G., and PETRENKO, A.: 'Test Selection Based on Communicating Nondeterministic Finite-State Machines Using a Generalized Wp-Method', *IEEE Transactions on Software Engineering*, 1994, **20**, (2), pp. 149-162

4 PETRENKO, A., YEVTUSHENKO, N., VON BOCHMANN, G., and DSSOULI, R.: 'Testing in Context: framework and test derivation', *Computer Communications*, 1996, **19**, (14), pp. 1236-1249

5 HUANG, C-M., CHANG, Y-I., and HUANG D-T.: 'Reverse Protocol Verification: Concept, Algorithm and Application', *The Computer Journal*, 1996, **39**, (6), pp. 511-524

6 CHOW, T.S.: 'Testing Software Design Modelled by Finite State Machines', *IEEE Transactions on Software Engineering*, 1978, **4**, pp. 178-187

7 AHO, A.V., DAHBURA, A.T., LEE, D., and UYAR, M.U.: 'An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences

and Rural Chinese Postman Tours' *Protocol Specification, Testing, and Verification VIII*, 1988, Atlantic City, pp. 75-86, North-Holland

8 YANG, B. and URAL, H.: 1990. 'Protocol Conformance Test Generation Using Multiple UIO Sequences with Overlapping' *ACM SIGCOMM 90: Communications, Architectures, and Protocols*, Sept 24-27 1990, Twente, The Netherlands, pp. 118-125

9 HIERONS, R.M.: 'Extending Test Sequence Overlap by Invertibility', *The Computer Journal*, 1996, **39**, (4) pp. 325-330

10 KOHAVI, Z.: 'Switching and Finite State Automata Theory' (McGraw-Hill, New York 1978)

11 CHRISTOFIDES, N.: 'Vehicle Routing', in *The Traveling Salesman*, Editors LAWLER E.L., LENSTRA J.K., RINNOY KAN A.H.G., and SHMOYS D.B., pp431-448 (Wiley, 1985)

12 ROCHAT, Y. and TAILLARD, E.: 'Probabilistic diversification and intensification in local search for vehicle routing', *Journal of Heuristics*, 1995, **1**, (1), pp. 147-167

13 THANGIAH, S.R., OSMAN, I.H., VINAYAGAMOORTHY, R., and SUN, T.: 'Algorithms For The Vehicle Routing Problem With Time Deadlines', *American Journal of Mathematical and Management Sciences*, 1994, **13**, (3&4), pp. 323-355

14 CLARKE, G. and WRIGHT, J.W.: 'Scheduling of vehicles from a central depot to a number of delivery points', *Operational Research*, 1964, **12**, pp. 568-581

15 GIBBONS, A.: 'Algorithmic Graph Theory' (Cambridge University Press, 1985)