Automated Software Engineering manuscript No.
(will be inserted by the editor)

Estimating the feasibility of transition paths in Extended Finite State Machines

Karnig Derderian · Robert M. Hierons · Mark Harman · Qiang Guo

September 3, 2009

Abstract There has been significant interest in automating testing on the bases of an extended finite state machine (EFSM) model of the required behaviour of the implementation under test (IUT). Many test criteria require that certain parts of the EFSM are executed. For example, we may want to execute every transition of the EFSM. In order to find a test suite (set of input sequences) that achieves this we might first derive a set of paths through the EFSM that satisfy the criterion using, for example, algorithms from graph theory. We then attempt to produce input sequences that trigger these paths. Unfortunately, however, the EFSM might have infeasible paths and the problem of determining whether a path is feasible is generally undecidable. This paper describes an approach in which a fitness function is used to estimate how easy it is to find an input sequence to trigger a given path through an EFSM. Such a fitness function could be used in a search-based approach in which we search for a path with good fitness that achieves a test objective, such as executing a particular transition, and then search for an input sequence that triggers the path. If this second search fails then we search for another path with good fitness and repeat the process. We give a computationally inexpensive approach (fitness function) that estimates the feasibility of a path. In order to evaluate this fitness function we compared the fitness of a path

K. Derderian
School of Information Systems, Computing and Mathematics, Brunel University
E-mail: karnig.derderian@brunel.ac.uk

R. M. Hierons
School of Information Systems, Computing and Mathematics, Brunel University
E-mail: rob.hierons@brunel.ac.uk

M. Harman
Department of Computer Science, King's College London, UK
E-mail: Mark.Harman@kcl.ac.uk

Q. Guo
Department of Computer Science, University of Sheffield, UK E-mail: Q.Guo@dcs.shef.ac.uk

with the ease with which an input sequence can be produced using search to trigger the path and we used random sampling in order to estimate this. The empirical evidence suggests that a reasonably good correlation (0.72 and 0.62) exists between the fitness of a path, produced using the proposed fitness function, and an estimate of the ease with which we can randomly generate an input sequence to trigger the path.

Keywords EFSM, transition feasibility, state-based testing, automated test generation

1 Introduction

Finite state machines (FSMs) and extended finite state machines (EFSMs) have been used to model state based systems in different areas like sequential circuits [10], software development [3] and communication protocols [1, 14, 19, 26–30]. They have been found to be an effective method for modelling and there are automated techniques and tools that can be used with them. To ensure the reliability of systems once implemented they must be tested for conformance to their specifications.

While FSMs can be used to model the control structure of a system, they are not suitable for modelling state-based systems that have data. For example, if we are modelling a simple vending machine and wish to include in this model the amount of change in the machine then the state of the machine includes the change currently in the machine. This leads to a vast state space if we use FSMs even for such a simple system. In contrast, if we use EFSMs then we can represent the change with a few variables and obtain a relatively small model. As a result, most state-based modelling languages, such as statecharts and SDL, use EFSMs rather than FSMs.

This paper is motivated by problems related to testing a state-based system against an EFSM model M. In testing we use a test suite, which is a set of input sequences to be applied to the *implementation under test (IUT)*. Each such input sequence defines a path through M: the path traversed if we simulate the execution of M with the input sequence. In addition, test criteria, which give the desired properties of the test suite, are often expressed in terms of paths through the EFSM. For example, we may want a test suite that leads to every transition of M being executed (see, for example, [20]).

There are two main approaches to automating test generation from an EFSM M. The first approach is to expand out the data to form an FSM but this leads to a combinatorial explosion and often is impractical. An alternative is to devise a set of paths through the EFSM that, between them, satisfy the test criterion. We can then attempt to find input sequences that lead to these paths in M being followed. Unfortunately, however, these paths may be infeasible and even if they are feasible it may be difficult to find input to trigger them. While there has been work on transforming an EFSM to one that has no infeasible paths [6, 7], this has only been achieved for EFSMs in which

all operations and guards are linear. In addition, the problem of determining whether a path in an EFSM is feasible is generally undecidable.

This paper aims to contribute to the use of search-based techniques, such as genetic algorithms and simulated annealing, to automating test generation from an EFSM. Such approaches use fitness functions that allow the test generation problem to be expressed as one of finding an input sequence with optimal fitness. There has been significant interest in the use of search-based techniques to automate white-box testing (see, for example, [16,18,21]), in which we require test data that executes certain structures in the code. There has also been recent interest in applying search-based techniques when testing from an EFSM and the problem of finding an input sequence to traverse a given path has been investigated (see, for example, [15]). Unfortunately, a chosen path through an EFSM may be infeasible and this paper proposes the use of a simple fitness function whose aim is to direct testing towards paths that are likely to be feasible and relatively easy to trigger using search. If we can define such a fitness function then this can be used in a search for a path that satisfies a test objective (part of a test criterion) and we then attempt to find an input sequence to follow the chosen path. If we fail to find such an input sequence then we can iterate.

Since path feasibility is generally undecidable any fitness function devised to direct testing towards paths that are likely to be feasible can only be an approximation. In addition, if we are to use such a fitness function in searching for an appropriate path then we require a fitness function that is computationally cheap to evaluate: search-based techniques often require fitness functions to be evaluated many thousands of times. As a result, we require a relatively simple fitness function and it seems likely that there will be a trade-off between complexity and precision.

This paper makes two main contributions. First, it proposes a fitness function that aims to estimate how easy it is to trigger a path and that is computationally cheap to evaluate. The second major contribution is that we evaluate the fitness function using experiments on two examples. In the experiments we generated a set of paths and for each path we evaluated the path's fitness and estimated how easy it is to find an input sequence to execute the path using search: this was estimated by attempting to find an appropriate input sequence using random search. While the proposed fitness function is (deliberately) quite simple and cheap to compute, the experiments found there to often be a reasonably good correlation between the fitness of a path and the estimate of how easy it is to find an input sequence to execute the path. This result is extremely promising and suggests that automated test generation methods might use this fitness function, or something similar, to search for appropriate paths through an EFSM that are likely to be feasible and relatively easy to trigger using search.

The paper is structured as follows. In Section 2 it outlines the problem and describes EFSMs. The approach used to search for *feasible transition paths (FTPs)* and the proposed fitness function are described in Section 3, including preprocessing steps that have to be carried out once for every EFSM. The

experiments are described in Section 4 and here we also analyse the results. Finally, in Section 5 we draw conclusions and discuss possible future work.

2 EFSM abstraction model

In this section we start in Section 2.1 by describing EFSMs. In Section 2.2 we then discuss the problem of testing from an EFSM and in Section 2.3 we then discuss the abstraction and representation we use.

2.1 EFSM model

FSMs are known to model appropriately sequential circuits and control portions of communication protocols. However FSMs are not powerful enough for some applications where EFSMs are used instead. EFSMs have been widely used in telecommunications, and are also now being applied in areas ranging over aircraft, train control, medical and packaging systems. Examples of languages based on EFSMs include SDL, Estelle and Statecharts [9].

By EFSMs we mean Mealy (finite state) machines with parameterised input and output, internal variables, operations and predicates defined over internal variables and input parameters. In the following we use NIL to refer to either an absence of input (no input is required to trigger a transition/operation) or to there being no guard/precondition, depending on the context in which it is used.

An EFSM M can be defined as $(S, s_0, V, \sigma_0, P, I, O, T)$ where

- S is the finite set of logical states
- $s_0 \in S$ is the initial state
- V is the finite set of internal variables
- σ_0 denotes the mapping from the variables in V to their initial values
- P is the set of input and output parameters
- I is the set of input declarations
- O is the set of output declarations
- T is the finite set of transitions.

A transition $t \in T$ is defined by (s_s, g_I, g_D, op, s_f) where

- s_s is the start state of t;
- g_I is the input guard expressed as (i, P^i, g_{P^i}) where
  - $i \in I \cup \{NIL\}$;
  - $P^i \subseteq P$; and
  - g_{P^i} is the input parameter guard that can either be NIL or a logical expression in terms of variables in V' and P' where $V' \subseteq V$, $\emptyset \neq P' \subseteq P^i$;
- g_D is the domain guard and can be either NIL or represented as a logical expression in terms of variables in V' where $V' \subseteq V$;

- op is the sequential operation which is made up of simple output and assignment statements; and
- s_f is the final state of t.

The label of a transition in an EFSM has two guards that decide the feasibility of the transition: the input guard g_I and the domain guard g_D. In order for a transition to be triggered its guard g_I must be satisfied. Some inputs may carry values or specific input parameters and M may guard those values with the input parameter guard g_P. Hence the values of the input parameters may determine whether a transition is triggered and affect the output of M. The input guard $(NIL, \emptyset, NIL)$ represents no input being required, which makes the transition spontaneous. g_D is the guard, or precondition, on the values of the system variables (for example, $v > 4$, where $v \in V$). Note that in order to satisfy the domain guard g_D of a transition t, it might be necessary to have taken some specific path to the start state of t. op is a set of sequential statements such as $v := v + 1$ and $!o$ where $v \in V$, $o \in O$ and $!o$ means 'output o to the environment'. Literal outputs (output directly observable by the user) are denoted with ! and output functions (an output function may produce different output depending on the parameters it receives) without it (for example, $!o$ and $u(v)$). If a transition t has an output function, then the output value produced by t is determined by the parameters passed to the output function (the parameters can include internal variables and input parameters for that transition). The operation of a transition in an EFSM has only simple statements such as output statements and assignment statements, no branching statements are allowed.

We assume that none of the spontaneous transitions in an EFSM are without any guards, $g_I = (NIL, \emptyset, NIL)$ and $g_D = NIL$, because they will be uncontrollable. When a transition in an EFSM is triggered, all the actions of the operation specified in its label are performed consecutively and only once.

Definition 1 An EFSM M is deterministic if any pair of transitions t and t' initiating from the same state s that share the same input declaration x have mutually exclusive guards.

When an EFSM receives an input, the choice of transition triggered depends not only on the input parameters but also on the values of the internal variables. In FSM there are no internal variables and guards on the transitions. This makes triggering a transition in EFSMs more complex than in FSMs. As an example consider the Initiator process of the Inres protocol [13] represented as an EFSM in Figure 1. There are two transitions initiating from state S_w with the input declaration $T_{expired}$ - t_2 and t_3. However they have mutually exclusive conditions - $counter < 4$ and $counter \geq 4$ respectively.

Definition 2 An EFSM is strongly connected if for every ordered pair of states (s, s') there is some feasible path from s to s'.

We assume that any EFSM considered is deterministic and strongly connected. Consider the EFSM in Figure 1 again. There is an FTP between every pair of states and so this EFSM is strongly connected.

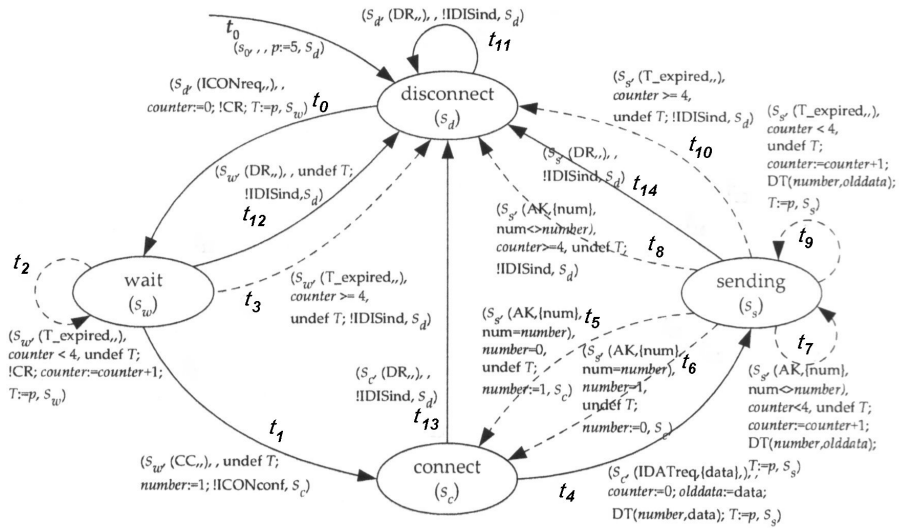

Fig. 1 Inres protocol as an EFSM M_1

2.2 Testing from EFSMs

Usually the implementation of a system specified by an FSM or EFSM is tested for conformance by applying a sequence of inputs and verifying that the corresponding sequence of outputs is that which is expected. This commonly involves executing a number of transition paths, until all transitions have been tested at least once. In EFSMs, in order to follow a transition path it is necessary to satisfy all of the guards involved, in addition to using a specific input sequence to trigger these transitions.

Definition 3 Given an EFSM M a transition path (TP) represents a sequence of transitions in M where every transition starts from the state where the previous transition finished.

Typically the machines that arise are complex and brute force testing is infeasible [22].

When systems are tested against an FSM, often a fault can be categorised as either an output fault (wrong output is produced by a transition) or a state transfer fault (the state after a transition is wrong). A test strategy for a transition with starting state s_i and input x would involve moving M to s_i, applying x, verifying the output, and checking the transition's end state. There are two main approaches to checking the state after a transition. State identification sequences determine the current state of the system [22]. In contrast, state verification sequences check that the current state is that expected [22]. Thus, the result of applying a state verification sequence tells us whether the system was in the correct state but if the state was wrong then it might not be enough to identify the actual (incorrect) state.

In EFSMs, test sequence generation is more complex than it is for FSMs. In FSMs all paths are feasible since there are no guards [6]. With EFSMs, however, in order to trigger a transition path it is necessary to satisfy the transition guards. A transition guard may refer to the values of the internal variables and the input parameters, which in turn can assume different values after each transition. Some transition paths might have no conditions, some might have conditions that are rarely satisfied and some transition paths will be infeasible. The existence of infeasible transition paths creates difficulties for automating test generation.

There are two main approaches for using FSM test techniques when testing from an EFSM. First, we can expand out the data in the EFSM to form an FSM but this can lead to a combinatorial explosion. Alternatively, when testing from an EFSM M we can abstract out the data to form an FSM $A(M)$. An FSM based automated test generation technique, when applied to $A(M)$, returns a set of paths through $A(M)$ and each such path corresponds to a path in M. However, there is no guarantee that these paths are feasible in M since the transitions may have guards. Recent work has shown how this problem can be overcome by transforming an EFSM M into an EFSM M' in which all paths are feasible [7]. Once this transformation has taken place, FSM based methods are used to produce paths through $A(M')$ in the knowledge that each such path corresponds to a feasible path in M'. However, this work requires that all operations and guards are linear and has exponential complexity. As a result, test generation for EFSMs is still an open research problem [7,14] and this motivates the approach described in this paper.

The general problem of finding a (an arbitrary) feasible transition sequence for an EFSM is uncomputable, as is generating the necessary input sequence to trigger such a transition sequence. While a random algorithm could be used it does not always produce acceptable results. Test case generation and optimisation for FSM based systems has been of interest [6,11,12,24]. Heuristic search techniques like genetic algorithms (GAs) have been used in problems like generating test sequences for FSMs [5,8] and generating test input sequences for communicating FSMs [4]. Heuristic search techniques can also be applied to the FTP generation problem if a robust fitness function can be defined. Hence a function that can estimate the likelihood that a TP can be triggered may help in areas like test sequence generation since it can be used to bias the search towards TPs that are likely to be feasible and relatively easy to trigger.

2.3 EFSM abstraction model

Now consider the problem of finding an input sequence that triggers a feasible transition path (FTP) from state s_i to state s_j of an EFSM M.

Definition 4 A forward feasible transition path (F-FTP) for state s_i of an EFSM M is a sequence of transitions initiating from s_i that is feasible for at least one combination of values of the finite set of internal variables V of M.

State identification and state verification sequences for an EFSM must trigger an F-FTP. Methods that can help identify F-FTPs can potentially be used to help in state identification and state verification sequence generation.

Definition 5 A backward feasible transition path (B-FTP) for state s_j of an EFSM M is a sequence of transitions ending at s_j that is feasible for at least one combination of values of the finite set of internal variables V of M.

A BF-FTP is a feasible transition path with specified both start state s_i and end state s_j. This paper uses FTP to refer to all three types.

In a transition path of an FSMs each transition can be represented by its start state and input (s_s, i). However with EFSMs this information is not sufficient because there can be more than one transitions sharing the same start state and input due to them having mutually exclusive guards. Instead a transition t in an EFSM M can be identified by its start state, input declaration, input parameter, the input parameter guard and the domain guard $(s_s, i, P^i, g_{P^i}, g_D)$. g_{P^i} and g_D for a transition t in M can both be logical expressions and their results may depend on input parameter P^i of t and the values of some of the internal variables of M. Transitions sharing the same start state and input declaration can be classified according to their input guard predicate and domain guard predicate. To identify these for every set of transitions sharing the same start state s and input declaration i, the number of unique input guards (input predicate branches) and unique domain guards (domain predicate branches) is counted and a predicate dependency tree for state s and input declaration i can be constructed.

Consider the Sending state (S_s) in the EFSM M_1 on Figure 1. There are four transitions initiating from this state that share the same input declaration AK and input parameter num. A partial transition table for this state (Figure 2) represents all the outgoing transitions from state S_s of M_1 with the same input that differ only in their input parameter guards and domain guards.

i	P^i	g_{P^i}	g_D	PB	t
AK	num	$num = number$	$number = 0$	$PB1$	t_5
AK	num	$num = number$	$number = 1$	$PB2$	t_6
AK	num	$num \neq number$	$counter \geq 4$	$PB3$	t_8
AK	num	$num \neq number$	$counter < 4$	$PB4$	t_7

Fig. 2 Partial transition table for transitions starting from S_s in EFSM M_1 (Figure 1)

A transition in a transition path of an EFSM can be identified by a tuple (s_s, i, g_{P^i}, g_D) in which s_s is its start state, i is its input, g_{P^i} is its input guard and g_D is its domain guard. The input parameter P^i is not required in order to be able to uniquely identify a transition in M. Note how in this case some transitions with different domain guards share a common input predicate guard.

Not all transitions in EFSMs have input parameter guards and domain guards and so transitions in an EFSM M can be categorised in the following way:

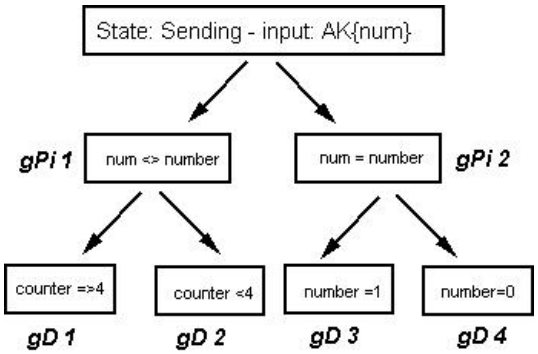

Fig. 3 Predicate dependency graph for the transitions of Figure 2

- **simple transitions** are those transitions that have *no* input parameter guard and *no* domain guard, $g_{P^i} = NIL$ and $g_D = NIL$.
- g_{P^i} **transitions** are those transitions that *have* an input parameter guard but *not* a domain guard, $g_{P^i} \neq NIL$ and $g_D = NIL$.
- g_D **transitions** are those transitions that *have* a domain guard but *not* an input parameter guard, $g_D \neq NIL$ and $g_{P^i} = NIL$.
- g_{P^i}-g_D **transitions** are those transitions that *have* both an input parameter guard and a domain guard, $g_{P^i} \neq NIL$ and $g_D \neq NIL$.

The sequential operations *op* for a given transition t can consist of simple assignments and output statements. However in EFSMs besides output declarations there are also output functions that take a number of parameters and generate an output based on these parameters. An assignment statement in the *op* part of a transition would not generate any observable output. However such assignment statements can still contribute to the value of the output generated in a later transition if the assignment changes the value of one of the output function's parameters. It can also affect the feasibility of the remaining transitions in the transition path and should also be considered.

Definition 6 An input sequence (IS) is a sequence of input declarations $i \in I$ with associated input parameters $P^i \subseteq P$ of an EFSM M.

Instead of using g_{P^i} and g_D together in order to identify a transition we can simply use a label.

Definition 7 A predicate branch (PB) is a label that represents a pair of g_{P^i} and g_D for a given state s and input declaration i. A PB identifies a transition within a set of transitions with the same start state and input declaration.

Since a PB identifies a transition t within the set of transitions with the same start state and input declaration as t, the combination of a PB, input declaration and a starting state identifies a transition. In addition, the ending state s' of the transition t is unique and so if we follow this with another PB and input declaration then we identify the next transition t' since we know

that the starting state of t' is s'. As a result, we can use a sequence of pairs of PBs and input declarations to identify a TP through the $EFSM$, that starts at the initial state, although the path may be infeasible. For example consider the transitions listed in Figure 2. Each transition can be identified by the start state S_s, input declaration AK and predicate branch label PB.

Definition 8 An abstract input sequence (AIS) for M represents an input declaration sequence with associated PBs that defines a TP in M.

Consider the sequence of transitions t_0, t_1, t_4, t_7 and t_5 initiating from state S_d in Figure 1. The AIS for this TP is $(ICONreq, PB_{s_d,1})$; $(CC, PB_{s_u,1})$; $(IDATreq, PB_{s_c,1})$; $(AK, PB_{s_s,4})$; $(AK, PB_{s_s,1})$. Generating an IS for a TP from an AIS involves two stages. The input declarations are mapped across from the AIS to the IS and then we need to find input parameters that trigger the PBs. This can lead to conditions on the values of some of the internal variables of the EFSM. For the TP presented earlier, the internal variable *counter* must be smaller than 4 so that transition t_5 is triggered.

3 Feasibility estimation algorithm

In order to achieve our objective we require an easy to compute fitness function that estimates the feasibility of a TP. It is always possible to trigger a *simple* transition in a transition path since there are no guards to be satisfied. The presence of g_{P^i} transitions could render a transition path infeasible because of its input predicate guard. The values of the input parameters $P^i \subseteq P$ are chosen by the tester. However, when these conditions also depend on some internal variables $V' \subseteq V$ (see, for example, the g_{P^i} of t_6 in M_1; $num = number$) then such g_{P^i} transitions might be no easier to trigger than g_D transitions. In some cases the execution of a g_D transitions could require reaching its start state through a specific transition path. The feasibility of g_{P^i}-g_D transitions depends on both issues outlined above for g_{P^i} transitions and g_D transitions.

Since the presence of g_D transitions and g_{P^i}-g_D transitions seem to increase the chance of a transition path being infeasible such transitions can be penalised and *simple* transitions rewarded.

One important issue to consider is how to weight the fitness of transitions of different types. A table with the associated penalties for every transition in a transition path is presented in Figure 4. The penalty values shown are by no means definitive, but they aim to aid the estimation according to the arguments presented above. It has also been assumed that $\neq$ is the easiest type of comparison operator to be satisfied while $=$ is the most difficult. Naturally, for some EFSMs exactly the opposite might be true in certain cases. A more detailed analysis of the EFSM might lead to better guidance but also might lead to a computationally more expensive fitness function.

When there is more than one condition in a guard the penalty value associated with the guard depends on the logical operator between these conditions.

Operator	*simple*	g_{Pi}	g_D	g_{Pi}-g_D
$\neq$	0	1	2	4
$\geq$ or $\leq$	0	3	4	8
$>$ or $<$	0	4	5	10
$=$	0	6	7	14

Fig. 4 Penalties for each condition in a transition

With the AND operator the sum of the penalties is used. For the OR operator only the lowest penalty is used.

A *transition ranking* process is completed first before the fitness function can be used. This process ranks each transition of the EFSM according to how many penalty points are assigned to the transition guards. A *simple* transition gets the highest rank (lowest penalty), an g_{Pi} transition with one condition ($\neq$) is ranked next etc. Transitions that have the same number of penalty points get the same rank. The ranks are represented by integers starting at 0 (for *simple* transitions). This algorithm in essence sorts $|T|$ elements and has complexity $O(|T|.log|T|)$ where $|T|$ is the number of transitions in M. This is a preprocessing step used to define the fitness function and does not have to be repeated when the fitness function is used.

Definition 9 The fitness is a function that given a TP of an EFSM M, sums the rank values (assigned through the transition ranking process for M) for each of the transitions of the TP.

The proposed fitness algorithm can be used to reward a potential solution to a TP generation problem according to the ranks of the transitions in the sequence. The fitness function reflects the belief that the fewer constraints a sequence contains, the more likely it is to be feasible. When there are only *simple* transitions, then the TP must be an FTP.

Estimating the feasibility of a TP is just the first part of the problem of generating actual input sequences to cover parts of an EFSM. Such input sequences do not always represent the shortest path between two states. Also there may be other computationally inexpensive analysis of a TP that can be added to the existing fitness functions to make it more accurate. In this work we focus on evaluating our feasibility estimation fitness function.

4 The Experiments

In this section we outline a set of experiments that evaluated the effectiveness of the proposed feasibility estimation function and present the results.

Two EFSMs were used in the experiments: The Inres protocol M_1 and a Class 2 transport protocol M_2 given in Figure 5. A breadth first search (BFS) algorithm was used to generate a set of TPs for each EFSM. The fitness function was then applied to each TP and we separately estimated the ease of executing that TP by applying the following *sampling* technique: Randomly generate 1000 sequences of input parameters for the TP and count how many

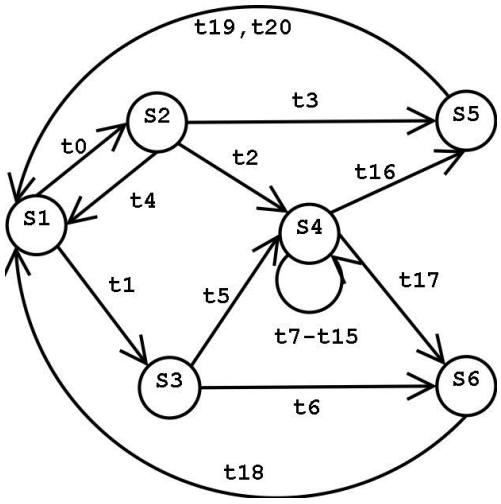

Fig. 5 Class 2 transport protocol EFSM M_2

of these trigger the TP. The value returned by sampling was called the quality (feasibility) measure of the path. This sampling process is described below in detail. The fitness function aims to estimate the feasibility of a TP in a computationally inexpensive manner and so we compared the fitness value of a TP with its quality measure.

The sampling process simply attempted to trigger a TP in question from the initial configuration of the EFSM. The TP defines an input sequence and thus it was sufficient to randomly generate values for the corresponding input parameters and these were generated within appropriate (preset) bounds. There were two conditions under which a TP was considered to have been successfully triggered:

1. Under *relaxed verification* we required that the start state and the end state of the triggered TP match that of the expected start and end state;
2. Under *strong verification* we required that all of the transitions of the TP are correctly followed in the order given in the TP.

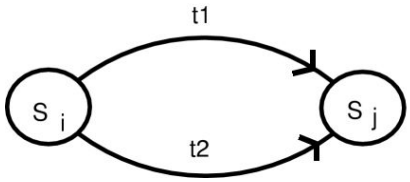

Fig. 6 Example transitions within a TP

To see the difference between relaxed verification with strong verification consider the example in Figure 6. The two transitions t_1 and t_2 both share

the same input declaration, start and end states. Imagine we have selected t_1 for our TP because it has a g_D $v' > 0$, $v' \in V$ whilst t_2 has a g_D $v' = 0$. As discussed previously a predicate with $>$ is considered to be more easily satisfiable than one with $=$. However for this particular EFSM $v' > 0$ might be more difficult to satisfy if, for example, the value of v is set to a positive value at only one transition in the EFSM. If we only care about the end states of the TP, this fact may not affect the result. This is because we might have chosen a particular TP that follws t_1 from s_i to s_j but the fact that t_2 is actually followed instead does not affect the end state of the TP. Since this paper is concerned with particular paths through an EFSM strong verification is most suitable, but in the experiments we also tested with relaxed verification.

We use M's initial state for all TPs since M is in the initial configuration (start state and values of the internal variables) after a simple reset. The problem of placing M in a given configuration before input sequence execution, other than its initial configuration, is not a focus of this work.

In the experiments presented in this paper the fitness function attempts to estimate how easily a TP can be triggered (hence how easily the input for this TP can be generated using search). The evaluation of the quality factor of an FTP involved 1000 attempts to trigger that TP with random input parameters. A negative statistical correlation is expected between the fitness values and the TP quality factor values. Correlation factor below 0.4 generally indicates lack of correlation while correlation factor of 0.6 and above indicates fair correlation between two sets of values considered [23]. A correlation factor between 0.4 and 0.6 is considered as some correlation. These limits vary for applications in different fields.

If a TP was triggered in the experiments then we know that it is feasible. However, we might fail to trigger a feasible path. We therefore checked each such path and manually confirmed that all TP that were not triggered in the evaluation of the quality factor actually were infeasible.

4.1 Class 2 Transport Protocol results

The Class 2 transport protocol M_2 is presented in Figure 5 and the corresponding transition table is shown in Figure 7 and Figure 8. M_2 has more states, transitions and is more complex than M_1. M_2 is a major module (based on the AP-module [2]) of a simplified version of a class 2 transport protocol [25]. M_2 represents only the core transitions of that EFSM, as used in [25].

1083 TPs were generated and plotted in Figure 9 that represent all possible TPs, starting at the initial state, with length of up to 5 transitions. 479 of these TPs have positive FTP quality factor (i.e. were successfully triggered at least once in 1000 attempts). All the 604 TPs with 0 quality factor are drawn in Figure 9 using lightly shaded squares. It transpired that the TPs with 0 quality factor were infeasible. Note that all TPs with 0 quality factor have a comparatively high fitness value.

t	s_{start}	s_{end}	i	Output	Ranking
t_0	s_1	s_2	U?TCONreq	N!TrCR	0
t_1	s_1	s_3	N?TrCR	U!TCONind	0
t_2	s_2	s_4	N?TrCC	U!TCONconf	3
t_3	s_2	s_5	N?TrCC	U!TDISind N!TrDR	4
t_4	s_2	s_1	N?TrDR	U!TDISind N!terminated	0
t_5	s_3	s_4	U?TCONresp	N!TrCC	1
t_6	s_3	s_6	U?TDISreq	N!TrDR	0
t_7	s_4	s_4	U?TDATAreq	N!TrDT	2
t_8	s_4	s_4	N?TrDT	U!DATAind N!TrAK	3
t_9	s_4	s_4	N?TrDT	U!error N!error	3
t_{10}	s_4	s_4	U?U_READY	N!TrAK	0
t_{11}	s_4	s_4	N?TrAK		6
t_{12}	s_4	s_4	N?TrAK	U!error N!error	4
t_{13}	s_4	s_4	N?TrAK		7
t_{14}	s_4	s_4	N?TrAK	U!error N!error	5
t_{15}	s_4	s_4	N?Ready	U!Ready	2
t_{16}	s_4	s_5	U?TDISreq	N!TrDR	0
t_{17}	s_4	s_6	N?TrDR	U!TDISind N!TrDC	0
t_{18}	s_6	s_0	N?terminated	U!TDISconf	0
t_{19}	s_5	s_0	N?TrDC	N!terminated U!TDISconf	0
t_{20}	s_5	s_0	N?TrDR	N!terminated	0

Fig. 7 Transition table for M_2 excluding transition guards

t	g_{P_i} and g_D	Ranking
t_0		0
t_1		0
t_2	$opt\_ind \leq opt$	3
t_3	$opt\_ind > opt$	4
t_4		0
t_5	$accpt\_ind \leq opt$	1
t_6		0
t_7	$S\_credit > 0$	2
t_8	$R\_credit \neq 0 \wedge Send\_sq = TRsq$	3
t_9	$R\_credit = 0 \wedge Send\_sq \neq TRsq$	3
t_{10}		0
t_{11}	$TSsq \geq XpSsq \wedge$ $cr + XpSsq - TSsq \geq 0 \wedge cr + XpSsq - TSsq \leq 15$	6
t_{12}	$TSsq \geq XpSsq \wedge$ $(cr + XpSsq - TSsq < 0 \vee cr + XpSsq - TSsq > 15)$	4
t_{13}	$TSsq < XpSsq \wedge$ $cr + XpSsq - TSsq - 128 \geq 0 \wedge cr + XpSsq - TSsq - 128 \leq 15$	7
t_{14}	$TSsq < XpSsq \wedge$ $(cr + XpSsq - TSsq - 128 < 0 \vee cr + XpSsq - TSsq - 128 > 15)$	5
t_{15}	$S\_credit > 0$	2
t_{16}		0
t_{17}		0
t_{18}		0
t_{19}		0
t_{20}		0

Fig. 8 Transition guards for M_2

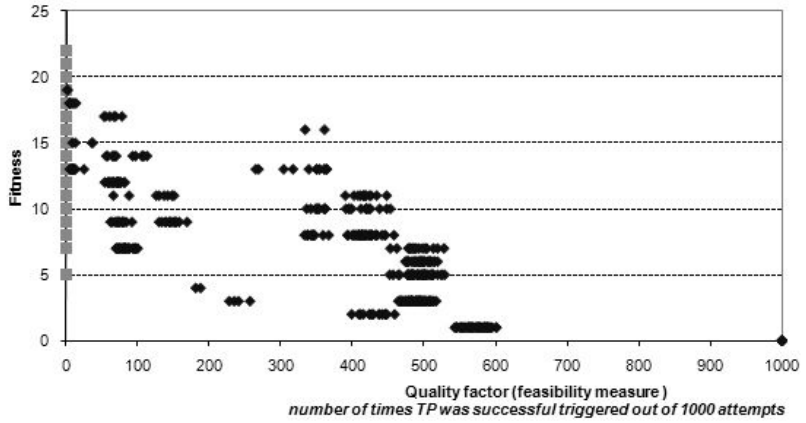

Fig. 9 All TPs generated using BFS algorithm for M_2 with 1-5 transitions (correlation factor 0.72, when excluding the 0 quality factor TPs the correlation factor is 0.76). Light shaded squares represent zero-quality TPs. The dark diamonds represent FTPs.

This high proportion of TPs with 0 quality factor indicates that some initial TP conflict resolution could be used to improve the results. Conflict resolution for EFSM test sequences has been addressed in [7] for EFSMs with linear operators and guards and the use of such methods remains a topic for future work.

Some interesting FTPs from Figure 9 and their respective fitness values and quality factors are listed in Figure 10.

	TP	fitness	quality factor (feasibility)
1	1;6;18;0;4;	0	1000/1000
2	1;5;16;20;1;	1	603/1000
3	0;2;7;	3	481/1000
4	0;2;11;11;11;	19	2/1000
5	1;5;14;11;11;	18	5/1000
6	1;5;15;14;11;	14	50/1000
7	0;2;10;11;10;	7	73/1000
8	0;4;0;2;11;	7	83/1000
9	1;5;10;15;11;	9	65/1000
10	0;2;10;14;16;	6	473/1000
11	0;2;10;11;16;	7	69/1000
12	0;2;14;14;14;	16	337/1000
13	0;2;7;14;14;	13	240/1000
14	1;5;14;14;10;	11	436/1000

Fig. 10 Example TPs for the Class 2 transport protocol in transition notation.

The fitness function (best fitness value is 0, and positive fitness values indicate less desirable solutions) correctly identified all TPs that were made of only *simple* transitions. Such TP had a fitness value of 0 and a quality factor of 1000/1000 (for example, TP 1 in Figure 10). TPs that contain mostly *simple*

transitions but also a few conditional transitions have fitness value slightly above 0 and corresponding quality factor of about 500/1000 (for example, TP 2 and 3 in Figure 10). A disproportionate drop in the TP quality factor as soon as conditional transitions are considered is observed in Figure 9. The introduction of a single conditional transition to a TP can generally be expected in theory (all other things being equal) to halve the quality factor. Every additional condition in theory should further reduce the remaining quality factor. This could explain the two clusters of points between the 0-200/1000 and 400-600/1000 quality factor values.

Most TPs with quality factor below 200/1000 have high fitness values. In these cases the fitness algorithms has correctly estimated that the TPs are not easy to trigger. TPs 4, 5 and 6 in Figure 10 are such examples.

There are some paths that the fitness algorithm estimated as not too hard to trigger (i.e. with relatively low fitness value) that have surprisingly low quality factor values (for example, TPs 7, 8 and 9 in Figure 10). It was found that transition $t11$ of M_2 is in all those TPs. To illustrate the effect of this transition in a TP consider TPs 10 and 11 in Figure 10. The only difference between them is that the fourth transition of TP 11 is $t11$. Their fitness values differ only by one point, indicating that $t11$ is estimated to be more difficult to trigger than $t14$ (used in TP 10 instead of $t11$). However the sharp contrast in the quality factor values indicates that $t11$ is harder to trigger than the fitness function has estimated (relative to the other transitions) for this transition sequence. A more detailed analysis of the predicates before they are ranked could help prevent this. Although this remains a topic of future work a few potential improvement of the fitness function are briefly noted later.

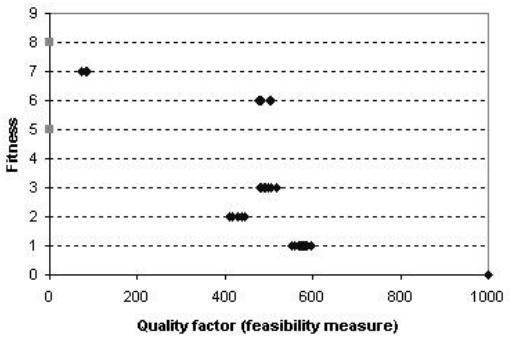

Fig. 11 All FTPs generated using BFS algorithm for M_2 with 1-5 transitions ending at s_1 (correlation factor 0.83, when excluding the 0 quality factor TPs the correlation factor is 0.77). Light shaded squares represent unfeasible TPs. The dark diamonds represent FTPs.

There were some TPs with high fitness values (estimated to be hard to trigger) that have relatively high feasibility ratio. Such examples included TPs 12, 13 and 14 in Figure 10. All TPs with such characteristics seem to contain

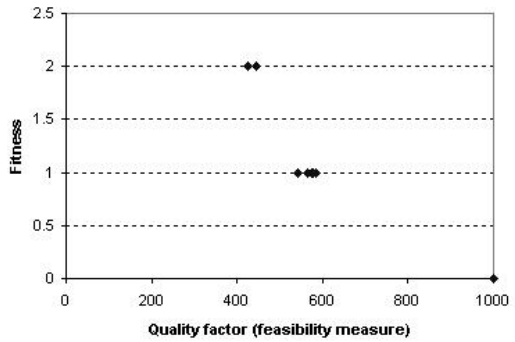

Fig. 12 All FTPs generated using BFS algorithm for M_2 with 1-5 transitions ending at s_2 (correlation factor 0.95). The dark diamonds represent FTPs. No zero-quality TPs were found.

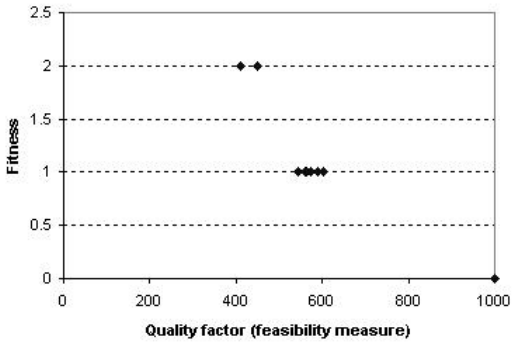

Fig. 13 All FTPs generated using BFS algorithm for M_2 with 1-5 transitions ending at s_3 (correlation factor 0.69, when excluding the 0 quality factor TPs the correlation factor is 0.76). Light shaded squares represent zero-quality TPs. The dark diamonds represent FTPs.

$t14$. Here the fitness function has not discovered that even though $t14$ has a number of complex conditions, they are easy to satisfy for this transition sequence. There is potential for a more complex analysis of the transitions to consider other factors that relate to the context in which the transition lies. The results of such an analysis might be used in the transition ranking phase and thus fitness definition or we might use a more complex fitness function that considers the previous transitions before determining the fitness contribution of a transition t. Such alternative methods might attempt to more accurately estimate the feasibility of TPs that include transitions like $t11$ and $t14$ but are a topic for future work.

The fitness function seems to correctly estimate the feasibility of most of the 1083 TPs. There is a negative correlation factor of 0.72 between the fitness function and the quality factor illustrated on Figure 9. If we only consider the 479 FTPs the correlation factor is 0.76. The FTP results (including unsuc-

cessful TPs) are classified in six sets according to the end state of the FTPs. The results for s_1, s_2, s_3, s_4, s_5 and s_6 are presented in Figures 11, 12, 13, 14, 15 and 16 accordingly. The cumulative results have negative correlation factors of 0.83, 0.95, 0.95, 0.69, 0.71 and 0.76 for TPs of s_1, s_2, s_3, s_4, s_5 and s_6 accordingly and negative correlation factors of 0.77, 0.87, 0.87, 0.75, 0.70 and 0.78 for the 479 FTPs accordingly.

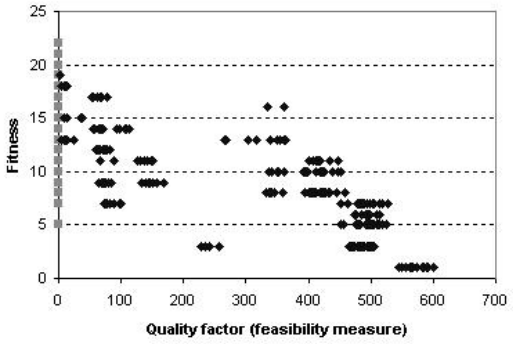

Fig. 14 All FTPs generated using BFS algorithm for M_2 with 1-5 transitions ending at s_4 (correlation factor 0.69, when excluding the 0 quality factor TPs the correlation factor is 0.75). Light shaded squares represent zero-quality TPs. The dark diamonds represent FTPs.

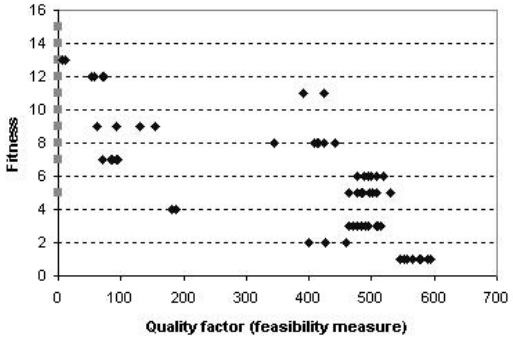

Fig. 15 All FTPs generated using BFS algorithm for M_2 with 1-5 transitions ending at s_5 (correlation factor 0.71, when excluding the 0 quality factor TPs the correlation factor is 0.70). Light shaded squares represent zero-quality TPs. The dark diamonds represent FTPs.

This relatively strong correlation shows that even though the fitness function underestimated TPs with $t11$ and overestimated TPs with $t14$, it can reasonably estimate the feasibility of a TP without executing it.

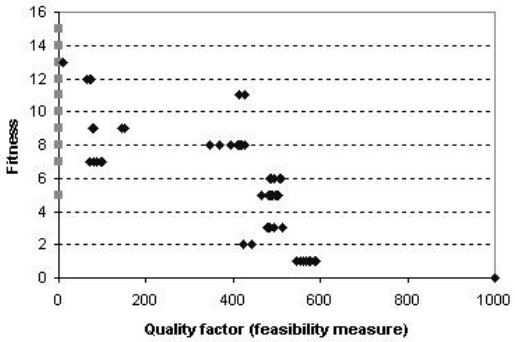

Fig. 16 All FTPs generated using BFS algorithm for M_2 with 1-5 transitions ending at s_6 (correlation factor 0.76, when excluding the 0 quality factor TPs the correlation factor is 0.78). Light shaded squares represent zero-quality TPs. The dark diamonds represent FTPs.

The results presented use strong FTP verification (the exact TP has to be followed). When the same experiment was done using relaxed FTP verification where only the start and end states of the TP are of importance the results generated a negative correlation factor of only 0.13 and when only considering FTPs - 0.22. This shows that the fitness is better at assessing feasibility of a given path than whether the corresponding IS can take us to a particular state. However, this is unsurprising since the fitness function only estimates the given path. Although the relaxed FTP verification might be useful in cases like the one shown on Figure 6, strong FTP verification seems to perform much better (at least for this EFSM).

4.2 Inres Protocol results

The EFSM M_1 in Figure 1 represents the Inres protocol that is simpler than the Class 2 transport protocol M_2 previously examined. The transition table for M_1 is presented in Figure 17.

The BFS algorithms was used to generate 257 TPs for M_1 where only 7 TPs had a 0 quality factor. Figure 18 illustrates those TPs that represent all possible TPs with length of up to 6 transitions. Again TPs with quality factor of 0 are drawn with lighter shaded squares. However, there are considerably fewer TPs with 0 quality factor for M_1. This could indicate that the FTPs are easier to generate.

Some interesting FTPs from Figure 18 and their respective fitness values and quality factors are listed in Figure 19.

Naturally, the fitness function correctly identified all TPs that were made of only *simple* transitions just as it did for M_2. Such TPs had a fitness value of 0 and a quality factor 1000/1000 (for example, TP 1,2 and 9 in Figure 19). TPs containing mostly *simple* transitions tend to have a high quality factor and low fitness value (for example, TP 8 in Figure 19 has quality factor 860/1000 and

t	s_{start}	s_{end}	i	Output	g_{P_i} and g_D	Ranking
t_0	s_d	s_w	ICONreq	!CR		0
t_1	s_w	s_c	CC	!ICONconf		0
t_2	s_w	s_w	T_expired	!CR	$counter < 4$	2
t_3	s_w	s_d	T_expired	!IDISind	$counter \geq 4$	1
t_4	s_c	s_s	IDATreq	DT		0
t_5	s_s	s_c	AK		$num = number \wedge number = 0$	6
t_6	s_s	s_c	AK		$num = number \wedge number = 1$	6
t_7	s_s	s_s	AK	DT	$num \neq number \wedge couter < 4$	5
t_8	s_s	s_d	AK	!IDSind	$num \neq number \wedge couter \geq 4$	4
t_9	s_s	s_s	T_expired	DT	$counter < 4$	3
t_{10}	s_s	s_d	T_expired	!IDSind	$counter \geq 4$	2
t_{11}	s_d	s_d	DR	!IDSind		0
t_{12}	s_w	s_d	DR	!IDSind		0
t_{13}	s_c	s_d	DR	!IDSind		0
t_{14}	s_s	s_d	DR	!IDSind		0

Fig. 17 Transition table for the Inres protocol on Figure 1.

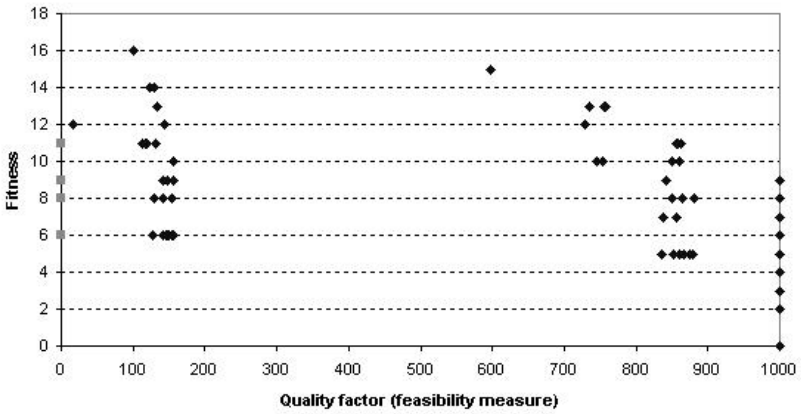

Fig. 18 All FTPs generated using BFS algorithm for M_1 with 1-6 transitions (correlation factor 0.62, when excluding the 0 quality factor TPs the correlation factor is 0.60). Light shaded squares represent zero-quality TPs. The dark diamonds represent FTPs.

fitness value of 5). Different to the results for M_2 the introduction of guarded transitions in the TPs generated for M_1 does not result in a halving of the quality factor. Even though a condition was introduced in the TP, the TP quality factor seems to remain high. This could indicate that the introduced condition is easy to satisfy. Here the fitness function seems to be penalising t_7, the only conditional transition in TP 8, slightly more than necessary. To illustrate the effect of this transition in a TP consider TPs 8 and 9 in Figure 19. The only difference between them is that the sixth transition of TP 8 is t_7. Their fitness values differ by 5 points but they both have high quality factors. This indicates that t_7 is estimated to be much more difficult to trigger than t_{14} (used in TP 9 instead of t_7). However the sharp contrast in the fitness values indicates that t_7 is slightly easier to trigger than the fitness function

	TP	fitness	quality factor (feasibility)
1	0;1;4;14;	0	1000/1000
2	20;20;0;4;12;	0	1000/1000
3	0;1;4;7;7;6;	16	101/1000
4	0;2;1;4;7;6;	13	134/1000
5	0;1;4;9;7;6;	12	144/1000
6	11;11;0;1;4;6;	6	127/1000
7	0;1;4;6;4;14;	6	142/1000
8	11;11;0;4;7;	5	860/1000
9	11;11;0;4;14;	0	1000/1000
10	0;1;4;6;4;	6	149/1000
11	0;1;4;6;4;5;	12	16/1000
12	0;1;4;9;9;9;	9	1000/1000
13	0;2;2;2;2;3;	9	1000/1000
14	0;2;1;4;9;7;	10	860/1000
15	0;1;4;7;7;	10	745/1000

Fig. 19 Example TPs for the Inres protocol in transition notation.

has estimated for this transition sequence. A more detailed analysis of the predicates before they are ranked could help prevent this, as discussed before. As mentioned in Section 4.1, in the context of t_{11} and t_{14} in M_2, analysing transitions like t_7 in more detail can help achieve better fitness results.

A large number of TPs with quality factor below 200/1000 have high fitness values. In those cases the fitness algorithms has correctly estimated these TPs as not being easy to trigger. TPs 3, 4 and 11 in Figure 19 are such examples.

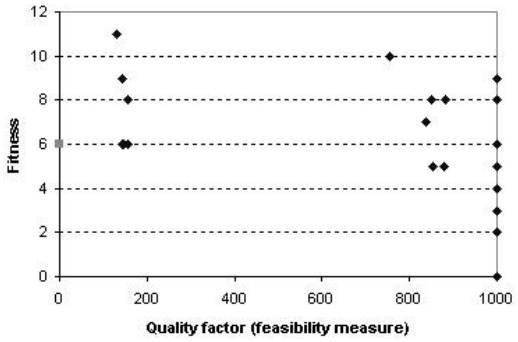

Fig. 20 All FTPs generated using BFS algorithm for M_1 with 1-6 transitions ending at s_d (correlation factor 0.62, when excluding the 0 quality factor TPs the correlation factor is 0.61). Light shaded squares represent zero-quality TPs. The dark diamonds represent FTPs.

There as some paths that the fitness algorithm estimated as not too hard to trigger (i.e. with relatively low fitness value) that have surprisingly low quality factor values (for example, TPs 6 and 7 in Figure 19) and transition t_6 appears to be in all those TPs. To illustrate the effect of this and a similar transition t_5 in a TP consider TPs 10 and 11 in Figure 19. The only difference between them

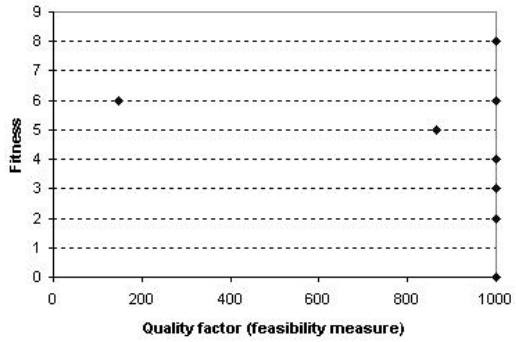

Fig. 21 All FTPs generated using BFS algorithm for M_1 with 1-6 transitions ending at s_w (correlation factor 0.54, when excluding the 0 quality factor TPs the correlation factor is 0.5). The dark diamonds represent FTPs. No zero-quality TPs were found.

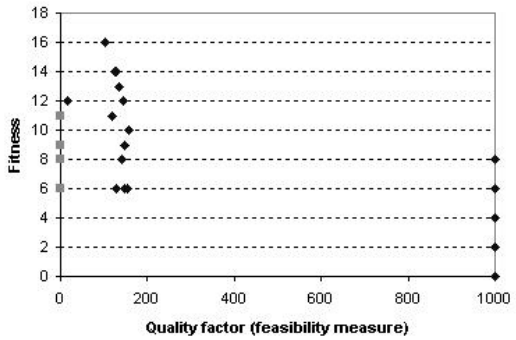

Fig. 22 All FTPs generated using BFS algorithm for M_1 with 1-6 transitions ending at s_c (correlation factor 0.85, when excluding the 0 quality factor TPs the correlation factor is 0.87). Light shaded squares represent zero-quality TPs. The dark diamonds represent FTPs.

is that the sixth transition of TP 11 is t_5. t_6 is the only conditional transition in TP 10 but the fitness value is only 6. This suggests that executing t_6 is more difficult than the fitness function estimated. Similarly t_5 seems to have similar properties as TP 11 has a quality factor of only 16/1000. A more detailed analysis of the predicates before they are ranked could be useful in this example as well.

There were some TPs with high fitness values (estimated to be hard to trigger) that have surprisingly high quality factors. Such examples included TPs 12, 13, 14 and 15 in Figure 19. All TPs with such characteristics seem to contain t_2, t_7 or t_9. We have already seen that t_7 seems to be penalised too harshly. Consider t_2 and t_3 in Figure 17. t_3 is estimated to be easier to trigger than t_2 due to the difference in the comparison operators of their g_D. However, when the EFSM is closely examined t_2 appears to always be

feasible for four consecutive executions and only then does t_3 become feasible for a single execution. t_2 represents a counter loop and t_3 is the loop exit. After four consecutive executions of t_2, it becomes infeasible and the loop exit t_3 is triggered. Similarly t_9 represents a counter loop and t_{10} is the loop exit. The dynamic behaviour of these transitions is difficult to estimate using a simple ranking process. So, identification of such loops could be beneficial. Detailed analysis of the predicates involved in this example could have strongly benefited the fitness function.

The fitness function seems to correctly estimate how easy it is to trigger each of the 257 TPs. There is a negative correlation factor of 0.62 between the fitness function and the quality factor illustrated in Figure 18. If we only consider the 479 FTPs the correlation factor is 0.6. The FTP results are classified in four sets according to the end state of the FTPs. The results for s_d, s_w, s_c and s_s are presented in Figures 20, 21, 22 and 23 accordingly. The cumulative results have negative correlation factors of 0.62, 0.54, 0.85 and 0.49 for FTPs of s_d, s_w, s_c and s_s accordingly. The correlation factor seems to be lowest for the two states s_w and s_s, target states for t_2 and t_9 accordingly and negative correlation factors of 0.61, 0.5, 0.87 and 0.54 for the 479 FTPs accordingly.

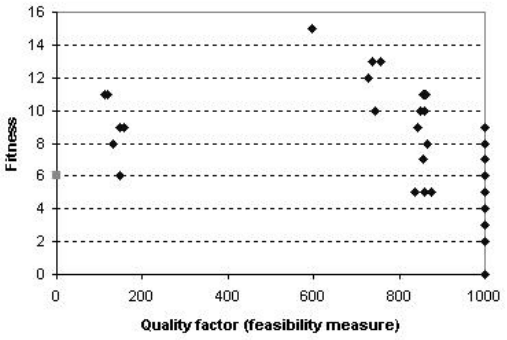

Fig. 23 All FTPs generated using BFS algorithm for M_1 with 1-6 transitions ending at s_s (correlation factor 0.49, when excluding the 0 quality factor TPs the correlation factor is 0.54). Light shaded squares represent zero-quality TPs. The dark diamonds represent FTPs.

The correlation between the fitness function and the TP quality factor is not as strong as that for M_2, however, it still seems to correctly estimate the feasibility for much more than half the FTPs. The small size of the EFSM and the complex dynamic behaviour of some of the transitions (loops of fixed number of iterations) seem to contribute towards this result.

The results presented use strong FTP verification (the TP has to be followed). Similar results were also generated using the relaxed FTP verification where only the start and end states of the FTP are of importance. Surprisingly those results generated negative correlation factors of 0.62 for all TPs

and 0.61 for FTPs. Again the small size of the EFSM is the likely cause for similar correlation values.

5 Conclusions and future work

This paper defined a computationally inexpensive method to address an important problem in test data generation for EFSMs. An EFSM abstraction method was used as the basis of a fitness function that estimated the feasibility of a transition path (TP). The fitness function was evaluated using two EFSMs - Inres protocol and a Class 2 transport protocol. A breadth first search algorithm was used to generate a set of TPs for each of the EFSMs. The fitness function was calculated for each TP and compared to a separately evaluated ease of execution estimation measure, produced using sampling.

The fitness evaluation results suggest that the fitness function estimates the feasibility of a given TP with reasonable accuracy. The 0.72 and 0.62 correlation factors suggest a good correlation between the fitness algorithm and estimated feasibility for the TPs for the two EFSMs used in the experiments. Hence this computationally inexpensive TP feasibility function may be used to aid the generation of paths through an EFSM that satisfy a test criterion and which are likely to be feasible. Other methods such as heuristic search and constraint satisfaction, can then be used to produce input sequences to trigger these paths.

Future work will investigate refining the fitness function to take into account loops and other difficult to estimate transitions. There is likely to be a trade-off between the sophistication/precision of the fitness function and the cost of computing it. Interestingly, recent work has extended the fitness function proposed in this paper by adding some dataflow information [17] but it is unclear how this will affect the accuracy of the fitness function. Further evaluation of the fitness function on other EFSMs may also be beneficial. A final area of future work is the use of the fitness function in methods that generate particular types of input sequences from an EFSM. For example, we might combine it with the UIO sequence generation work in [5].

References

1. Aho, A., A. Dahbura, D. Lee, and M. U. Uyar: 1991, 'An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tour'. *IEEE Transactions on Communications* **39**, 1604–1615.
2. Bochmann, G. V.: 1990, 'Specifications of a simplified transport protocol using different formal description techniques'. *Comput. Netw. ISDN Syst.* **18**(5), 335–377.
3. Chow, T. S.: 1978, 'Testing software design modelled by Finite State Machines'. *IEEE Transactions on Software Engineering* **4**, 178–187.
4. Derderian, K., R. M. Hierons, M. Harman, and Q. Guo: 2004, 'Input Sequence Generation for Testing of Communicating Finite State Machines CFSMs.'. In: *LNCS vol. 3103*. pp. 1429–1430.
5. Derderian, K., R. M. Hierons, M. Harman, and Q. Guo: 2006, 'Automated Unique Input Output Sequence Generation for Conformance Testing of FSMs'. *The Computer Journal* **49**(3), 331–344.

6. Duale, A. Y. and M. Ü. Uyar: 2000, 'Generation of Feasible Test Sequences for EFSM Models'. In: *TestCom '00: Proceedings of the IFIP TC6/WG6.1 13th International Conference on Testing Communicating Systems*. Deventer, The Netherlands, The Netherlands, p. 91.
7. Duale, A. Y. and M. Ü. Uyar: 2004, 'A Method Enabling Feasible Conformance Test Sequence Generation for EFSM Models.'. *IEEE Transactions Computers* **53**(5), 614–627.
8. Guo, Q., R. M. Hierons, M. Harman, and K. Derderian: 2004, 'Computing Unique Input/Output Sequences using Genetic Algorithms'. In: *Formal Approaches to Software Testing: Third International Workshop, FATES 2003, LNCS vol. 2931*. pp. 169–184.
9. Harel, D. and M. Politi: 1998, *Modeling reactive systems with statecharts: the STATEMATE approach*. McGraw-Hill.
10. Hennie, F. C.: 1964, 'Fault–detecting experiments for sequential circuits'. In: *Proceedings of Fifth Annual Symposium on Switching Circuit Theory and Logical Design*. Princeton, New Jersey, pp. 95–110.
11. Hierons, R. M. and H. Ural: 2002, 'Reduced Length Checking Sequences'. *IEEE Transactions on Computers* **51**(9), 1111–1117.
12. Hierons, R. M. and H. Ural: 2006, 'Optimizing the Length of Checking Sequences'. *IEEE Transactions on Computers* **55**(5), 618–629.
13. Hogrefe, D.: 1991, 'OSI formal specification case study: the Inres protocol and service.'. Technical Report 5, University of Bern.
14. Lee, D. and M. Yannakakis: 1994, 'Testing Finite State Machines: State Identification and Verification'. *IEEE Transactions on Computers* **43**, 306–320.
15. Lefticaru, R. and F. Ipate: 2008, 'Functional Search-based Testing from State Machines'. *First IEEE Int. Conf. on Software Testing, Verification, and Validation*, 525–528.
16. Jones, B. F., D. E. Eyres and H.- H. Sthamer: 1998, 'A strategy for using genetic algorithms to automate branch and fault-based testing'. *The Computer Journal* **41**(2), 98–107.
17. Kalaji A.S., R. M. Hierons, and S. Swift: 2008, 'Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM)'. *Second IEEE International Conference on Software Testing, Verification and Validation (ICST 2009)*.
18. Michael C. C., G. McGraw and M. A. Schatz:2001, 'Generating Software Test Data by Evolution'. *IEEE Transactions on Software Engineering* **27**(12), 1085–1110.
19. Miller, R. E. and S. Paul: 1993, 'On the generation of minimal-length conformance tests for communication protocols'. *IEEE/ACM Trans. Netw.* **1**(1), 116–129.
20. Offutt, A. J., S. Liu, A. Abdurazik and P. Ammann: 2003, 'Generating test data from state-based specifications'. *Softw. Test., Verif. Reliab.* **13**(1), 25–53.
21. Pargas R. P., M. J. Harrold, and R. R. Peck: 1999, 'Test-data generation using genetic algorithms'. *Softw. Test., Verif. Reliab.* **9**(4), 263–282.
22. Lee, D. and M. Yannakakis: 1996, 'Principles and methods of testing finite state machines - a survey'. *Proceedings of the IEEE* **84**, 1090–1123.
23. Pearson, S. W. and J. E. Bailey: 1979, 'Measurement of Computer User Satisfaction.'. In: *Int. CMG Conference*. pp. 49–58.
24. Petrenko, A., S. Boroday, R. Gorz: 2004, 'Confirming Configurations in EFSMs'. *IEEE Transactions on Software Engineering* **30**(1), 29–42.
25. Ramalingom, T., K. Thulasiraman, and A. Das: 2003, 'Context independent unique state identification sequences for testing communication protocols modelled as extended finite state machines.'. *Computer Communications* **26**(14), 1622–1633.
26. Sidhu, D. P. and T. K. Leung: 1989, 'Formal Methods for Protocol Testing: A Detailed Study'. *IEEE Transactions on Software Engineering* **15**, 413–426.
27. Shen, X. and G. Li: 1992, 'A new protocol conformance test generation method and experimental results'. In: *SAC '92: Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing*. pp. 75–84.
28. Shen, Y., F. Lombardi, and T. Dahbura: 1989, 'Protocol conformance testing using multiple UIO sequences'. In: *IFIP WG6.1 9th Int. Symp. on Protocol Specification Testing and Verification*. Amsterdam, Holland, pp. 131–144.
29. Tanenbaum, A. S.: 1996, *Computer Networks*. Prentice Hall, Upper Saddle River, NJ, USA. 3rd edition.
30. Yang, B. and H. Ural: 1990, 'Protocol conformance test generation using multiple UIO sequences with overlapping'. In: *SIGCOMM '90: Proceedings of the ACM symposium on Communications architectures & protocols*. pp. 118–125.