Aiding test case generation in temporally constrained state based systems using genetic algorithms*

Karnig Derderian[1], Mercedes G. Merayo[2],
Robert M. Hierons[1], Manuel Núñez[2]

[1] Department of Information Systems and Computing, Brunel University
Uxbridge, Middlesex, UB8 3PH United Kingdom,
derderian@karnig.co.uk, rob.hierons@brunel.ac.uk
[2] Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Madrid, Spain,
mgmerayo@fdi.ucm.es, mn@sip.ucm.es

Abstract. Generating test data is computationally expensive. This paper improves a framework that addresses this issue by representing the test data generation problem as an optimisation problem and uses heuristics to help generate test cases. The paper considers the temporal constraints and behaviour of a certain class of (timed) finite state machines. A very simple fitness function is defined that can be used with several evolutionary search techniques and automated test case generation tools.

1 Introduction

Testing is an important part of the system development process that aims to increase the reliability of the implementation. Unfortunately, testing can be very expensive and difficult. In particular, the problem of generating test sequences for real-time systems is not trivial. One of the main problems to overcome is that it is not enough to test that the implementation system is doing what it is supposed to do, but it is necessary also to test that it is also taking the specified time to complete. In addition, the tests applied to implementations have to consider *when* to test [2]. This paper addresses the issues related to generating test sequences for temporally constrained FSM based systems. It focuses on generating transition paths with specific properties that can, in turn, be used to generate test inputs. Taking as a first step our previous work [3], where genetic algorithms (GAs) were used to aid the test sequence generation of FSMs, the problem of generating these paths is represented as a search problem and GAs are used to help automate test data generation. Other approaches to generate test data using GAs are reported in [4, 5]

* An extended version of this paper, including a case study, can be found in [1]. Research supported by the Spanish projects WEST/FAST (TIN2006-15578-C02-01) and MATES (CCG08-UCM/TIC-4124).

Ordinary FSMs are not powerful enough for some applications where extended finite state machines (EFSMs) are used instead. EFSMs have been widely used in the telecommunications field, and are also now being applied to a diverse number of areas. Here we consider EFSMs with the addition of *time*: Timed EFSMs (TFSMs). In EFSMs, a transition path (TP) represents a sequence of transitions in M where every transition starts from the state where the previous transition finished. In order to execute a TP it is necessary to satisfy all of the transition guards involved, in addition to using a specific input sequence to trigger these transitions.

When FSM based systems are tested for conformance with their specification, 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). When TEFSMs are tested for conformance there are time related faults that arise when a transition within the implementation takes longer to complete than the time specified by the TEFSM. Thus, test sequence generation is more complex than it is for FSMs. In FSMs all paths are feasible since there are no guards and actions do not affect the traversal [6]. With TEFSMs, however, in order to trigger the 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 the test generation process for TEFSMs. One way of approaching the test sequence generation problem is to abstract away the data part of the TEFSM and consider it as an FSM on its own. However, a transition sequence for the underlying FSM of a TEFSM is not guaranteed to be feasible for the actual TEFSM nor to satisfy the temporal constraints. This leads to the problem that an input sequence will only trigger a specific TP in a TEFSM if all the transition guards allow this and so an input sequence generated on the basis of the FSM may not follow the required path in the TEFSM. Another approach is to expand a TEFSM to an FSM and then use the techniques used for FSMs. However this can lead to a combinatorial explosion.

The problem of TP feasibility in EFSMs was considered in [7]. This paper uses the TP feasibility approach proposed in that work and extends it to consider TEFSMs and problems associated to testing the compliance of an implementation to its temporal aspects of the specification. In addition to estimating the feasibility of a TP, in this paper we examine how to consider the temporal properties of a TP and help in related test case generation. In order to generate a test case for a TEFSM M we can first consider the properties of the TP that this test case is expected to trigger. The general problem of finding a (an arbitrary) feasible transition sequence for a TEFSM is uncomputable, as is generating the necessary input sequence to trigger such a transition sequence. The task of finding a transition sequence with particular temporal conditions complicates the problem even further.

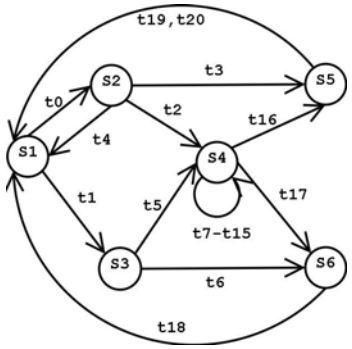

Fig. 1. Class 2 transport protocol TEFSM M_1. The transition table is on Fig. 2.

2 Timed extended finite state machine (TEFSM) model

To represent our timed EFSM assume that the number of different variables is m. If we assume that each variable x_i belongs to a domain D_i thus the values of all variables at a given point of time can be represented by a tuple belonging to the cartesian product of $D_1 \times D_2 \times ... \times D_m = \Delta$. Regarding the domain to represent time we define that time values belong to a certain domain **Time**.

Definition 1. *A TEFSM M can be defined as $(S, s_0, V, \sigma_0, P, I, O, T, C)$ 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 and C is such that $C \in \Delta$.*

*A transition $t \in T$ is defined by $(s_s, g_I, g_D, g_C, op, s_f)$ 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 be 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 $V' \subseteq V$; $g_C : \Delta \rightarrow$ **Time** is the time the transition needs to take to complete; op is the sequential operation which is made of simple output and assignment statements; and s_f is the final state of t.*

A TEFSM M is deterministic if any pair of transitions t and t' initiating from the same state s that share the same input x have mutually exclusive guards. A TEFSM M is strongly connected if for every ordered pair of states (s, s') there is some feasible path from s to s'. A configuration for a TEFSM M is a combination of state and values of the internal variables V of M.

We assume that any TEFSM considered is deterministic and strongly connected. For example, consider the Class 2 transport protocol [8] represented as a TEFSM in Figure 1. There are two transitions, t_2 and t_3, initiating from state

S_2 with the same input declaration. However, they have mutually exclusive conditions: $opt\_ind \leq opt$ and $opt\_ind > opt$, respectively. For example, consider again Figure 1. There is a TP from the initial state S_1 to every other state.

The label of a transition in a TEFSM 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 executed g_I, the guard for inputs from the environment 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 executed and affect the output of M. The input guard $(NIL, \emptyset, NIL)$ represents no input being required (spontaneous transition). g_D is the guard, or precondition, on the values of the system variables (e.g. $v > 4$, 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 (e.g. $!o$ and $u(v)$). The operation of a transition in a TEFSM has only simple statements such as output statements and assignment statements, no branching statements are allowed. In a TEFSM the time a transition takes to complete, represented by g_C, is also important. This time can depend on the current values of V' and P^i, where $V' \subseteq V$ and $P^i \subseteq P$.

We assume that none of the spontaneous transitions in a TEFSM are without any guards, $g_I = (NIL, \emptyset, NIL)$ and $g_D = NIL$, because they would be uncontrollable. When a transition in a TEFSM is executed, all the actions of the operation specified in its label are performed consecutively and only once and the transition must not take more than $g_C(C)$ time units to perform. The transition is considered to have failed (in terms of compliance to the specification or be considered void) if it is not completed within the required time. Such transitions may be considered void if they take longer than expected in some systems and in others they might still be executed even though they took longer than expected.

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

Definition 2. *A timed feasible transition path (TFTP) for state s_i to state s_j of a TEFSM 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 (configuration) of M and ends in s_j.*

In a transition path for FSMs each transition can be identified, and thus represented, by its start state and input (s_s, i). However, with TEFSMs this information is not sufficient because there can be more than one transition sharing the same start state and input due to having mutually exclusive guards. Instead, a transition t in a TEFSM M can be identified from its start state,

t	input	output	feasibility rank	temporal rank
t_0	ICONreq	!CR	0	0
t_1	CC	!ICONconf	0	0
t_2	T_expired	!CR	2	0
t_3	T_expired	!IDISind	1	1
t_4	IDATreq	DT	0	0
t_5	AK		6	1
t_6	AK		6	1
t_7	AK	DT	5	0
t_8	AK	!IDISind	4	0
t_9	T_expired	DT	3	0
t_{10}	T_expired	!IDISind	2	1
t_{11}	DR	!IDISind	0	2
t_{12}	DR	!IDISind	0	2
t_{13}	DR	!IDISind	0	2
t_{14}	DR	!IDISind	0	2

Fig. 2. Temporal constraint ranking and feasibility ranking for all transitions in M_1

input declaration, input parameter, the input parameter guard and the domain guard $(s_s, i, P^i, g_{P^i}, g_D)$.

A transition in a transition path of a TEFSM can be identified by a tuple (s_s, i, g_{P^i}, g_D) in which s_i 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.

Since not all transitions in TEFSMs have input parameter guards and domain guards, they can be categorised in the following way: *simple transitions* are those transitions that have *no* input parameter guard and *no* domain guard, that is, $g_{P^i} = NIL$ and $g_D = NIL$; g_{P^i} *transitions* are those transitions that *have* input parameter guard but *not* a domain guard, that is, $g_{P^i} \neq NIL$ and $g_D = NIL$; g_D *transitions* are those transitions that *have* a domain guard but *not* an input parameter guard, that is, $g_D \neq NIL$ 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, that is, $g_{P^i} \neq NIL$ and $g_D \neq NIL$. Next we introduce some concepts that will be used later.

Definition 3. *An input sequence (IS) is a sequence of input declarations $i \in I$ with associated input parameters $P^i \subseteq P$ of a TEFSM M.*

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.

An abstract input sequence (AIS) for M represents an input declaration sequence with associated PBs that triggers a TP in the abstracted M.

PBs can label conditional transitions and be used to help simulate the behaviour of a potential input sequence for a TEFSM without the feasibility restrictions. The advantages of using AIS and simulating the execution of a TEFSM is

that the configuration of the TEFSM is not considered. Hence, transition traversal evaluations that can be used to estimate the characteristics of a TP can be done without complex computation.

3 Using genetic algorithms to aid test case generation

In order to achieve our objectives we require an easy to compute fitness function that estimates the feasibility of a TP but also helps us to test our temporal constraints. Computing the actual feasibility of a transition path is computationally expensive, so we need a method to estimate this.

Some transition paths consist of transitions with difficult to satisfy guards. It is always possible to execute 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. However the conditions of this guard are more likely to be satisfiable than domain guards because the values of the input parameters $P^i \subseteq P$ can be chosen. When these conditions depend also on some internal variables $V' \subseteq V$ then such g_{P^i} transitions might not be 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 in a TP. In this paper we use the same feasibility estimation framework as in [7].

We may consider the temporal constraints of the transitions in a TP. For example a test sequence may be needed to stress test the temporal aspects of the system. Adequate test cases should be feasible (in order to be useful) and may focus on transitions with complex temporal constraints. However since the temporal constraints for every transition of M are dependant on the configuration of M (the current values if the internal variables) then it is difficult to know the exact temporal constraints without executing the system and verifying the configuration of M. If the temporal constraints for M are listed in a table then we can analyse the constraints and categorise different transitions in a similar way as we classified them according to their guards above. However if the temporal constraints are represented by one or more formulas the complexity of analysing all the possibilities the problem may become intractable.

We may try to estimate the different temporal constraints associated with each transition according to how the temporal constraint is defined. Note that the same transition may have different temporal constraints depending on the values of the internal variables of M. Some transitions may not have temporal constraints at all, while others might have fixed temporal constraints that are not dependant on the configuration of M. Other transitions may have temporal constraints that are expressed using tables while some may have the temporal constraints represented using formulas.

Based on these observations we may classify the transitions in a TEFSM M in the following way: *no-time transitions* are those transitions that have *no* temporal constraints; *fixed-time transitions* are those transitions that *have* temporal constraints that *are not* effected by the values of the internal variables V of M; *known-time transitions* are those transitions that *have* temporal constraints that *are* effected by the values of the internal variables V of M, but presented in an easy to analyse way; *variable-time transitions* are transitions that *have* temporal constraints that *are* effected by the values of the internal variables V of M, but are presented by one or more formulas and the temporal constraints are not easy to analyse without considering a subset of all the configurations of M.

A *transition ranking* process is completed before the fitness function can be used. This process first 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 (i.e. lowest amount of penalty points), a g_{P^i} transition is ranked next, etc. Transitions that have the same number of penalty points get the same rank. This algorithm in essence sorts $|T|$ elements and has complexity $O(|T| \cdot log|T|)$ where $|T|$ is the number of transitions in M. Then the process ranks each transition according to its temporal constraint category. In our case, if we are attempting to stress test the implementation then we can argue that $variable-time$ transitions can potentially have the most complex temporal constrains hence be ranked highest (i.e. lowest amount of penalty points), $known-time$ transitions can be ranked next as they still depend on the internal variables of M, and so on. The order can be reversed if the aim is to find a test sequence that will most likely work. Such test cases may be used in early stages of software development.

Definition 4. *The fitness is a function that given a TP of a TEFSM M, sums the penalty points (assigned through the transition ranking process for M and the temporal constrain ranking for M) for each of the transition of the TP.*

In our approach the two rankings are given equal weight before being combined, following the conclusions of a similar experiment in [7] where different weights were used for a similar multi-optimisation problem. The fitness algorithm used in this work can be used to reward a potential solution to a TP generation problem according to the combined ranks of the transitions in the sequence. This function reflects the belief that the fewer constraints a sequence contains, the more likely it is be feasible and the less we can analyse the temporal constraints of a transition, the more likely it is that they are more complex.

Estimating the feasibility of a TP is just the first part of the more difficult problem of generating actual IS for a TFTP that 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 our approach we focus on evaluating our feasibility and complex temporal conditions estimation fitness function.

As reported in the extended version of this paper [1], the two studied GAs performed well and generated very similar results. This indicates that the fitness

function and the TP representation represent the problem of TFTP generation reasonably well.

4 Conclusions and Future work

This paper defines a computationally inexpensive method to address an important problem in test data generation for TEFSMs. It extends some work on feasibility of EFSMs [7] and considers temporal constraints. We defined the problem of finding a transition sequence that is likely to be feasible and to satisfy some temporal criteria as a search problem. We defined a computationally efficient fitness function that is used to guide a GA. In the extended version of the paper [1] a case study of a communication protocol shows how the defined fitness function yields some positive results when GA search is applied to the problem.

Future work may focus on refining the fitness function to take into account loops and other difficulties to estimate transitions. Further analysis of the conditions might also help. Further evaluation of the fitness function on other larger TEFSMs may also be beneficial to evaluate further how well the method scales. Different TEFSMs may present the need for alternative temporal constraint classification, which will be very interesting to investigate.

References

1. Derderian, K., Merayo, M.G., Hierons, R.M., Núñez, M.: Aiding test case generation in temporally constrained state based systems using genetic algorithms. Available at: `http://kimba.mat.ucm.es/manolo/papers/aiding-extended.pdf` (2009)
2. Merayo, M., Núñez, M., Rodríguez, I.: Generation of optimal finite test suites for timed systems. In: 1st IEEE & IFIP Int. Symposium on Theoretical Aspects of Software Engineering, TASE'07, IEEE Computer Society Pres (2007) 149–158
3. Derderian, K., Hierons, R.M., Harman, M., Guo, Q.: Automated Unique Input Output Sequence Generation for Conformance Testing of FSMs. The Computer Journal **49**(3) (2006) 331–344
4. Jones, B.F., Eyres, D.E., Sthamer, H.H.: A strategy for using genetic algorithms to automate branch and fault-based testing. The Computer Journal **41**(2) (1998) 98–107
5. Michael, C.C., McGraw, G., Schatz, M.A.: Generating software test data by evolution. IEEE Transactions on Software Engineering **27**(12) (2001) 1085–1110
6. Duale, A.Y., Uyar, M.Ü.: 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, Kluwer, B.V. (2000) 91–112
7. Derderian, K.: Automated test sequence generation for Finite State Machines using Genetic Algorithms. PhD thesis, Brunel University (2006)
8. Ramalingom, T., Thulasiraman, K., Das, A.: Context independent unique state identification sequences for testing communication protocols modelled as extended finite state machines. Computer Communications **26**(14) (2003) 1622–1633