Balanced FSM Generation for Empirical Studies

Sarah Salahuddin
Dept. of Computer Science
University of Sheffield, UK
s.salahuddin@dcs.shef.ac.uk

Kirill Bogdanov
Dept. of Computer Science
University of Sheffield, UK
k.bogdanov@dcs.shef.ac.uk

Neil Walkinshaw
Dept. of Computer Science
University of Sheffield, UK
n.walkinshaw@dcs.shef.ac.uk

Abstract

The experiments conducted to compare and analyse different FSM-based testing methods do not always provide the details of FSM generation. This paper focuses on automated generation of data, which can be used to run FSM- and X-machine test generation experiments. The novelty of the work is generation of what we call "balanced", state machines where the number of incoming and outgoing transitions from each state is a parameter specified by a user. It is also possible to build machines where a user-defined proportion of states have a few times higher number of incoming/outgoing transitions.

1. Introduction

Model Based Testing (MBT) uses a model of an application under test to derive test cases and evaluate test results. Empirical work in model based testing focuses on specific case studies using specific model types. There is no established repository of models that could be used to evaluate a testing method or models characterizing a variety of realistic software. A possible reason is that there are various types of models that can be used to model systems and generate test sets. These different models include finite state machines, state charts, unified modelling language models, markov chains, petri nets, grammars and others including decision trees, decision tables and program design language. In this paper we focus on the empirical work using finite state machines.

Evaluation of different testing methods can focus on their applicability to different kinds of software, whether different people using the same program and method get comparable results or even whether one can reuse a test set for a modified program without a substantial rework. When tests are generated from a model, one may be interested to see whether a method is useful for models of a specific kind or whether tests generated often exhibit specific properties.

The experiments in FSM-based testing methods [1-3] focus on the results and the data analysis with little focus on the FSMs being used for experiments. As pointed out by [4], there are three important aspects that characterize the maturity of the experiments in a study.

The first aspect is the rigour of the experiment to get results beyond mere data analysis. If the experiment is difficult to describe, it will also be difficult to replicate and thus will not be useful in future. The second aspect is the use of meaningful and relevant programs to experiment on and use of realistic techniques. The third aspect is to coordinate empirical research across the community and ensure that the results obtained by the experiments are ratified and complemented by each other.

In this paper we focus on the automated LTS (labelled transition system)/ FSM generation used in experiments such as those to analyse the impact of change on test sets. Labelled transition systems are state transition diagrams where each transition carries a single label rather than an input/output pair as in the case of FSMs. We implemented LTS because X-machine test methods [1] rely on it but the generation algorithm presented below can be easily modified to produce machines with inputs/outputs instead of labels. This will particularly address the first aspect of [4] mentioned above, providing the means to develop libraries of state machines that can be used as benchmarks for experiments with FSMs. It has already been used in the work [5] to evaluate how resistant a Vasilevski/Chow [6, 7] test method is to changes.

The rest of the paper is structured as follows: Section 2 describes how FSMs have been generated in a number of papers, section 3 describes the approach taken for LTS/FSM generation illustrated by an example, section 4 summarises the results obtained in [5] and section 5 concludes the paper.

2. Related Work

Much work has been done on experiments using FSMs, but there is little literature available on the generation of FSMs to be used in experiments. FSM generation can be seen as random graph generation with certain specific constraints on the graphs. There should be no equivalent states in the graph (two states are equivalent if they exhibit the same behaviour) and all states must have a path leading to them from the initial state.

Dorofeeva et al [8] have tried to address the problem of lack of research in FSM generation. The random machines generated by [8] to evaluate the FSM-based testing methods are complete FSMs with varying number of states and inputs. The number of inputs chosen is much less than the number of states. Though there has been no reason provided, a logical reason is that the work is based on models developed for protocol specifications, which have been used for FSM testing [9].

In another experimental evaluation of FSM-based testing methods, Adenilso et al [10] outline the details of the random FSMs used and also give the detail of the generation steps involved in random FSM generation. The authors generate random FSMs in a two-phase procedure. In the first phase an initially connected FSM is generated. This is achieved by selecting a state as the initial state and marking it as reachable, then for each state, which is not reachable, a reachable state is chosen and a transition is added with an input and an output between the pair of a reachable and an unreachable state. The previously unreachable state is then marked as reachable and the process continues until all states are reachable thus generating an initially connected FSM and completing the first phase. In the second phase, more transitions are added until the required numbers of transitions have been added. In this phase the choice of states is made randomly since all states are now reachable. One drawback to this approach is that the machine could have states with very low or very high numbers of incoming/outgoing transitions.

The use of random graph generation algorithms could be a possible way to generate FSMs with the necessary constraints since FSMs are directed graphs. The JUNG framework [11]could be used for this purpose. It is a free open source library that allows generation, manipulation, analysis and visualisation of graphs.

Of particular interest are the algorithms of randomly generated graphs in the JUNG network. The Erdös and Réyni [12] random graph generation algorithm produces graphs where every pair of vertices may be connected to each other with a probability specified by a user. The Barabassi-Albert [12] algorithm generates undirected graphs. The generation starts with a small number of vertices, and at each time step, a new vertex is created and is connected to existing vertices according to the principle of "preferential attachment", where vertices with higher degree have a greater probability of being chosen.

It appears hard to control the topology of a network generated with the described random generation algorithms in order to avoid unreachable and equivalent states. For this reason one has to generate numerous FSMs and discard those not satisfying those properties. Because of this it was not realistic to use these network algorithms.

An important statistic, used in graph network analysis, is the degree of a vertex. In-degree of a state is the number of transitions leading into it and out-degree of a state is the number of transitions leaving it. We use the term “degree of completeness” to calculate the in/out degree for states; it is a ratio of in/out degree to a number of different labels which may be placed on transitions. Since the machines generated have the same in/out degree,

In/out degree = No. of labels * degree of completeness

The total number of transitions can then be calculated using the in/out degree,

No. of transitions = No. of states * in/out degree

Varying in/out degrees for each of the states in a machine can lead to machines with an uneven distribution of transitions as identified in [10]. One possible solution to this problem is generating machines with all states having the same in/out degree. The algorithm to do this is described in section 3.1. Based on the evaluation of a few models, it appears that software mostly satisfies this property except for a few states where the in/out degree is very high and which have been addressed in section 3.2.

Another factor important in FSM generation is the number of labels; software models tend to show higher number of labels whereas the FSM experiments in the literature tend to focus on using FSM based testing methods for protocol testing which involves fewer labels, and as few as two. Therefore, in our FSM generation, machines generated had higher number of labels to be more representative of the software system models.

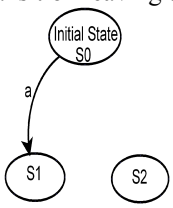
3. FSM Generation Algorithm

3.1 Balanced FSM Generation

Using the knowledge of random graph generation algorithms and the necessary constraints for LTS/FSM generation we have developed a generator, which allows generating machines with given number of states and labels that have same in/out degrees for all states. Unlike the random graph generation algorithms discussed earlier these are "balanced" graphs and not entirely random in their nature.

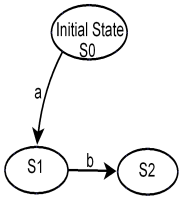
The machine generation algorithm is illustrated below using the example of a machine with 3 states and 3 labels.

Initially, as shown in figure 1 all 3 states are added to the machine and two of them are randomly chosen for creation of the first transition between these two states (S0 and S1) with a randomly chosen label *a*. The table next to the figure shows the in/out degree for each state and whether the state is reachable or not. For the next transition, the source state is chosen from the states that are reachable and have the lowest out-degree; therefore only S1 can be chosen as the source state. The state with lowest in-degree is chosen as a destination state, both S0 and S2 are candidates for the destination state and in this case either can be chosen randomly. A label for the new transition is chosen randomly as long as that label is not used on any transition leaving the source state.



States	*S0*	*S1*	*S2*
In-Deg	0	1	0
Out-Deg	1	0	0
Reachable	Y	Y	N

Figure 1: The initial step in FSM generation



States	*S0*	*S1*	*S2*
In-Deg	0	1	1
Out-Deg	1	1	0
Reachable	Y	Y	Y

Figure 2: Adding a transition

After a transition has been added from S1 to S2 the table is updated and source/destination states for the next transition are chosen using in/out degrees. For the destination state of the next transition there is only one choice i.e. the initial state S0, and the only choice for the source state is S2. Therefore, the third transition is added between S2 and S0. The described way to choose source and target states ensures that all states are given an equal chance of being selected thus eliminating the factor of randomness in state selection. The process of adding transitions continues until the desired numbers of transitions have been added.

After each selection of a pair of states and the transition label between them, the graph with this transition is checked for equivalent states and a new choice of source/destination states and a label is made if any two states become equivalent once a new transition is added. Figure 3a shows a machine with four transitions. The next transition possible could be from S1 to S1 using a label *a* shown in Figure 3b, but this would cause S0 and S1 to become equivalent. FSM-based testing methods require all pairs of states in the machine to be distinguishable, and therefore no equivalent states can exist. W set is the set of sequences that allows all pairs of states to be distinguished and in the course of testing this set is used for state identification.

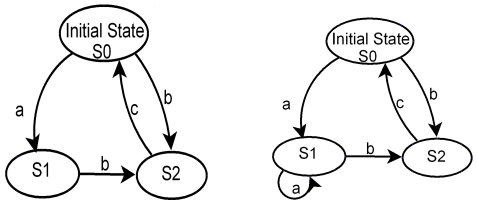


Figure 3a: Machine with 4 transitions
Figure 3b: Adding a transition that leads to an FSM with equivalent states (S0, S1)

3.2 FSM with Sun-Like States

Although balanced machines generated cover the threat to validity mentioned in [10] regarding the uneven distribution of transitions, balancing the in/out degree can be a limitation as it may not be most representative of realistic software systems. To address this limitation, a variation of the balanced FSM was developed. Two additional parameters were introduced, (a) a ratio of states with more in/out transitions than the rest of the states and (b) the ratio of transitions from those states to the degree of completeness. This variation of the FSM generation was a result of studying several FSM software models, where states representing menus have a higher in/out degree than the other states.

For example, a machine with 10 states and 20 labels with a degree of completeness 0.2 would have a total of 40 transitions i.e. each state with an in/out degree of 20*0.2= 4. A variation of this machine could be created with the proportion of sun-like states being 0.3 i.e 3 states having more transitions than the other 7 states. If the number of transitions from sun-like states is chosen

to be 4 times the number of transitions from/to other states, this would result in a machine with 3 states with in/out degree of 16 and the remaining 7 states with an in/out degree of 4. Usually, one would want to compare results of experiments on balanced graphs to those using sun-like states and the same total number of transitions. Such a normalisation would yield 3 states with in/out degree of 8 and 7 states with in/out degree 2 or 3.

4. Experiments Using the Proposed Algorithm

There is a lack of literature on the generation of realistic FSM models of software, and the focus of this paper is on the generation of such FSMs. We will not go into the details of all the results of the experiments conducted [5] using the described generator. Instead, to demonstrate the usefulness of the FSM generator we mention two sets of experiments where the FSM generator was used to generate the experimental data.

The generated FSMs were used in experiments analysing the impact of change on test sets. Test sets were generated for the machines and then changes were made and test sets were generated for the changed machines. Knowing the details of the FSMs allowed analysing the changes since all possible changes could be calculated.

(1) The W sets for the generated machines turned out to comprise of mostly singleton sequences, which is a characteristic representative of software models where the number of labels is quite large. (2) In addition, a W set generator that does not aim to reduce the size of W was producing W sets which were more resistant to changes than smaller W sets.

Besides testing, our FSMs have also been used to evaluate the performance of a grammar inference-based technique [13] to reverse engineer software systems. As with conventional testing techniques, grammar inference techniques tend to only be evaluated with respect to (quasi-) random graphs that do not tend to represent state machines of realistic software systems. The use of our synthetic machines enabled a more controlled evaluation, by systematically varying the number of states and transitions to gain an overview of the accuracy and scalability of the algorithm.

5. Conclusions

In this paper we have described an approach to generate data for LTS/FSM experiments. The generator has been developed with consideration of the points mentioned in section 3. In future, we aim to identify other variants of FSMs that could be representative of different types of software.

6. References

1. Bogdanov, K., Holcombe,M., Ipate,F., Seed,L., and Vanak,S., *Testing methods for X-machines: a review.* Formal Aspects of Computing, 2006. **18**(1): p. 3-30.
2. Chan, W.Y.L., Vuong,C. T., and Otp, M. R., *An improved protocol test generation procedure based on UIOS.* ACM SIGCOMM Computer Communication Review , Symposium proceedings on Communications architectures & protocols, 1989. **19**(4): p. 283-294.
3. Fujiwara, S., Bochmann,G., Khendek,F., Amalou,M. and Ghedamsi,A., *Test Selection Based on Finite State Models.* IEEE Transactions on Software Engineering, 1991. **17**(6): p. 591-603.
4. Juristo, N., M,Moreno., and Vegas,S., *Reviewing 25 years of testing technique experiments.* Empirical Software Engineering, 2004. **9**: p. 7-44.
5. Salahuddin, S., Bogdanov, K. *Analysing the Impact of Change on State Machine Test Sets*. *Submitted to A-MOST 2008*.
6. Chow, T.S., *Testing software design modelled by finite-state machines.* IEEE Transactions on Software Engineering, 1978. **4**(3): p. 178-187.
7. Vasilevskii.M.P., *Failure diagnosis of automata. Kibernetika. (Transl.),*. July-Aug.1973. **4**: p. 98-108.
8. Dorofeeva, R., Yevtushenko,N., El-Fakih,K., and Cavalli, A.,R. *Experimental Evaluation of FSM-Based Testing Methods* in *Proceedings of the Third IEEE international Conference on Software Engineering and Formal Methods* 2005. Washington: IEEE Computer Society.
9. Sidhu, D.P. and Leung,T.,K., *Formal Methods for Protocol Testing: A detailed study.* Transactions on IEEE Software Engineering, 1989. **15**(4): p. 413-426.
10. Simao, A., Petrenko,A., and Maldonado,J.C., *Experimental Evaluation of Coverage Criteria for FSM-based Testing*. in *Proceedings of XXI Brazilian Symposium on Software Engineering (SBES 2007)*. 2007. Joao Pessoa, Brazil.
11. Madadhain, J.O., Fisher,D.,Smyth.,S., White,S., and Boey,Y.B., *Analysis and Visualisation of Network Data using JUNG.* PrePrint, Journal of Statistical Software.
12. Barabasi, A.L., *Emergence of Scaling in Complex Networks.* Handbook of Graphs and Networks, 2004: p. 69-84.
13. Walkinshaw, N., Bogdanov,K., Holcombe,M., and Salahuddin,S., *Reverse Engineering State Machines by Interactive Grammar Inference* in *14th Working Conference on Reverse Engineering (WCRE 2007)*. 2007. Canada.