Reduced Checking Sequences Using Unreliable Reset

Guy-Vincent Jourdan[a], Hasan Ural[a], Hüsnü Yenigün[b,*]

[a] *University of Ottawa, Ottawa, Ontario, Canada*
[b] *Sabancı University, Tuzla, Istanbul, Turkey*

Abstract

The length of a checking sequence (CS) generated from a deterministic, minimal, and completely specified finite state machine model M of a system under test which does not have a reliable reset feature, is exponential when M does not have a distinguishing sequence. This is due to the exponential length locating sequences that need to be used in such a CS. In this work, we propose a method to decrease the number of locating sequences used in CS, by first verifying the reset input r and then using r as a reliable reset. When such a reset input is not available, a synchronizing sequence can be used as a compound reset input, which makes the proposed method applicable to a wide range of systems.

Keywords: Model-Based Testing, Finite State Machines, Checking Sequences, Synchronizing Sequences

1. Introduction

Testing from a Finite State Machine (FSM) model of systems in a variety of application areas has been used in Model-Based testing [1]. Much of the work in Model-Based testing from an FSM has been done on specific classes of FSMs, mostly on FSMs that have a Distinguishing Sequence (DS) [2], that is, a sequence of inputs that produces a unique output sequence for each state of the FSM [3]. A DS in this context is used to identify the states of the FSM. However,

*Corresponding author

Email addresses: gvj@eecs.uottawa.ca (Guy-Vincent Jourdan), ural@eecs.uottawa.ca (Hasan Ural), yenigun@sabanciuniv.edu (Hüsnü Yenigün)

not every FSM has a DS. A more general approach to state identification is to use a *characterization set*, commonly known as W-set [4], which is a set of input sequences producing a unique set of output sequences for each state of the FSM. Such a set exists for each deterministic, minimal, and completely specified FSM.

The common aim of all of the approaches for testing from an FSM is to construct a *checking sequence* (CS) from an FSM model M of an implemented system N which determines whether N is a correct implementation of M for some fault model. Correctness of N with respect to M is established when the output sequence produced by N in response to the application of the CS is the same as the expected output from M. Without a fault model, construction of a CS is not possible, because for a given CS C for an FSM M, a faulty implementation of M producing the expected output sequence from M for C can be constructed. A fault model places a set of assumptions on the possible implementations of M that will be tested by C to be constructed from M. Common assumptions made by approaches for testing from an FSM M include that the implementation N has at most the same number of states as M and does not change during testing.

CS construction methods often differ in the type of state identification sequences that they use, such as DS, W-sets and Unique Input-Output sequences [5], or the assumption that a *reliable reset* (a reset input for the implementation N that is known to work correctly) can be used. When a reliable reset is available in N, or a DS exists in M, a CS that is polynomial in the size of M can be constructed [6, 4, 7].

Otherwise (when there is no reliable reset and there is no DS), a CS can still be constructed using a W-set [8, 9, 10, 11] which exists for every minimal, completely specified FSM. All these methods utilize *Locating Sequences* which assure that every element of the W-set is applied to the same state. However, as pointed out in [10], when using W-sets for state identification the resulting checking sequence is "of exponential length in general", because the length of locating sequences grows exponentially with the size of the W-sets.

It should be noted that the major factor determining the length of the resulting checking sequence is the number of locating sequences used in the checking sequence. Although [9, 11] give sufficient conditions for the construction of a checking sequence of minimum length, both methods use the same number of locating sequences in forming the checking sequence, without making any attempt to reduce this number.

In this paper, we investigate when the number of locating sequences can be reduced in forming checking sequences. We consider FSMs without a DS and demonstrate that for the case of the FSMs with an *unreliable* reset and the FSMs with a synchronizing sequence, one can obtain a CS which is shorter than the one obtained by [9].

2. Background

For an integer $k \geq 1$, let $[k]$ denote the set $\{1, 2, \ldots, k\}$. An FSM M with input alphabet X and output alphabet Y is a set of state transitions among a set S of n states where a transition from state $s_i \in S$ to state $s_j \in S$ is triggered by an input $x \in X$ and produces an output $y \in Y$. For an input sequence $w \in X^\star$ and a state s, $\delta(s, w)$ denotes the state that is reached when w is applied at state s. As in [9], we consider M to be deterministic, completely specified, and minimal. M is *deterministic and completely specified*, if for each state s in S, there is exactly one transition in M outgoing from s defined for each input in X. M is *minimal* if for each distinct pair of states s and s', there exists an input sequence for which s and s' produce different output sequences.

A checking sequence for an FSM M is typically constructed in two phases (which may be interleaved): a *state identification* phase, and a *transition verification* phase. The state identification phase includes input sequences to verify that an implementation N of M has n states, and to recognize each state of N as corresponding to a state of M. This is achieved by bringing N to a state s (for each $s \in S$) and applying the state identification procedure there. The transition verification phase includes input sequences to verify that each transition of M is correctly implemented in N. This is achieved by bringing N to the state s_i (for each transition from s_i to s_j with input x and output y), by applying x at s_i, observing the output y, and by applying the state identification procedure at $\delta(s_i, x)$.

When a W-set is used to construct a checking sequence, the state identification procedure demands that every element of the W-set is applied to the same implementation state in order to identify that state of N as one of the states of M. In the absence of a reliable reset, the methods in [8, 3, 10] use a *Locating Sequence* L_{s_i}, for each $s_i \in S$, which meets this demand. That is, L_{s_i} assures that every element of the W-set is applied to the same implementation state, which corresponds to the specification state s_i.

Locating sequences are constructed in the following way [9]: Let $W = \{w_1, w_2, \ldots, w_k\}$ be a W-set. Given a state s_i and an input sequence $w_p \in W$, $p \in [k]$, let U_p^i be an input sequence such that w_p is a prefix of U_p^i and $\delta(s_i, U_p^i) = s_i$. The *locating sequence* L_{s_i} for state s_i is defined as the sequence $L_{s_i} = F_{k-2}(U_1^i, U_2^i, \ldots, U_k^i)$, where $F_0(a_1, a_2)$ is defined as the concatenation of $n+1$ copies of a_1 followed by a_2, and for $p > 0$, $F_p(a_1, a_2, \ldots, a_{p+2})$ is defined as the concatenation of $n+1$ copies of $F_{p-1}(a_1, a_2, \ldots, a_{p+1})$ followed by $F_{p-1}(a_1, a_2, \ldots, a_p, a_{p+2})$.

As can be seen from the definition, locating sequences grow exponentially quickly with the number of states n and the number of elements in the W-set k, with elements of W-set being applied in the order of n^k times.

In [9], state identification is achieved by applying at each state s_i a sequence $\alpha_i = L_{s_i} w_1 I_i^1 L_{s_i} w_2 I_i^2 \ldots L_{s_i} w_k I_i^k L_{s_i}$, where I_i^p is a *transfer sequence* of inputs from $\delta(s_i, w_p)$ to s_i. The verification of a transition from a state s_i to a state s_j with an input x is performed by applying at the state recognized as s_i the set of input sequences $\beta_{i,x,p} = x w_p L_{\delta(s_j, w_p)}$, for $p \in [k]$. Each transition thus yields k input sequences. Given that we are dealing with complete FSMs, there are exactly $q.n$ transitions, where q is the size of the input alphabet. The method forms a checking sequence by concatenating the set of α_i and $\beta_{i,x,p}$ sequences, using appropriate transfer sequences for concatenation whenever necessary. In terms of cost, defined as the total number of inputs in the CS, the main factor in [9] comes from the applications of locating sequences: each α_i ($i \in [n]$) requires the application of $k+1$ locating sequences, and each $\beta_{i,x,p}$ ($i \in [n], x \in X, p \in [k]$) contains one locating sequence. Therefore, in [9], the total number of application of locating sequences is $Q_{[9]} = n(k+1+qk)$.

3. Proposed Improvements

In this paper, we show that under some conditions, we can generate a CS that has fewer than $Q_{[9]}$ applications of locating sequences. The first such condition is when the specification FSM M has a *reset feature* where at every state of M the reset input r brings M to state s_1 and when this feature is not considered reliable in an implementation N of M. In this case, we propose to use locating sequences to verify the reliability of the reset, and use the *verified* reset instead of the locating sequence for the transition verification phase of the CS. The CS constructed using this approach will therefore have three phases: state identification, reset verification, and transition verification.

The state verification phase is performed almost exactly as in [9]. However, we do not need to ensure that the implementation ends at the state recognized as s_i after the application of α_i, so we can remove the last L_{s_i} from α_i, which yields a sequence $\alpha'_i = L_{s_i} w_1 I_i^1 L_{s_i} w_2 I_i^2 \dots L_{s_i} w_k$.

In order to verify the reset input r at each state, a CS needs to show that applying r from every state of N resets N to a state corresponding to state s_1 of M. For this reason, we include the sequences $\gamma_{i,p} = L_{s_i} r w_p, i \in [n], p \in [k]$ in CS for the reset verification phase. The transition verification phase will then proceed as in the case of the methods that use reliable reset (see for example [4]). Similar to these methods, suppose that a spanning tree for M rooted at state s_1 is constructed, where the edges are only labeled by the input symbols of the corresponding transitions. Let t_i be the label of the path from the root to the node labeled by s_i in this spanning tree. The transition verification phase includes sequences of the form $\beta'_{i,x,p} = r t_i x w_p, i \in [n], x \in X, p \in [k]$. Note that $\beta'_{i,x,p}$ is exactly the same as the transition verification phase of CS construction methods that are used when a reliable reset is available [8] and it does not incorporate the use of locating sequences.

In terms of cost, the main factor remains the applications of locating sequences. For the state identification phase, each α'_i yields k applications of L_{s_i}, and there are n such α'_i. For the additional phase of reset verification, from each state we include one locating sequence for each of the k sequences verifying the reset from that state, which yields nk applications of the locating sequences. Thus, the overall number of applications of locating sequences is $Q_{\text{new}} = 2nk < Q_{[9]}$.

3.1. Correctness of the method

In order to construct a checking sequence, the sequences α'_i, $\gamma_{i,p}$, and $\beta'_{i,x,p}$ explained above need to be concatenated to form a single sequence. Note that, each one of these sequences needs to be applied at a state of the implementation N of FSM M that will necessarily be recognized as a state s_i of M. Therefore, appropriate transfer sequences need to be used to concatenate these sequences when necessary. Let CS be an input sequence that is constructed from an FSM M in this way.

Lemma 1. *If N produces the expected output to CS then r is a reliable reset for N.*

Proof. CS includes L_{s_i} for each $i \in [n]$. Having the expected outputs from N for L_{s_i} sequences assures the existence of n states in N, and also the state q_i in N right before the application of the last element U^i_k in L_{s_i} is recognized as s_i of M [8]. The state reached in N after an application of L_{s_i} is also recognized as the state s_i of M due the inclusion of α'_i in CS. Therefore, the state of N at the end of an application of L_{s_i} is q_i as well.

In each sequence $\gamma_{i,p}$, for $p \in [k]$, after L_{s_i} is applied N is at q_i again, by the argument above. The reset input r is applied from q_i and the state reached after the application of r is recognized by applying the elements of the W-set as s_1 of M. $\square$

Lemma 2. *Every transition of M is verified by CS.*

Proof. The proof is by induction on the length of t_i in $\beta'_{i,x,p}$.

Base step ($|t_i| = 0$): This means that $s_i = s_1$. In this case, in $\beta'_{i,x,p}$, x is applied after r. Using Lemma 1, x is applied to the state recognized as s_1. The state reached in N after the application of the input x will be recognized as the state $\delta(s_1, x)$ by the application of the elements of the W-set.

Inductive step: Let s_j be the parent of s_i in the spanning tree used, and x be the label of the edge from s_j to s_i in the spanning tree. By the induction hypothesis, the transition from s_j to s_i with input x is verified, hence it is known that the sequence rt_i brings N to the state q_i that is recognized as s_i. Since x is applied after rt_i in $\beta'_{i,x,p}$, x is applied to the state q_i recognized as s_i. Similarly, the state reached in N after the application of the input x will be recognized as the state $\delta(s_i, x)$ by the application of the elements of the W-set. □

Theorem 1. *CS is a checking sequence for M.*

Proof. CS is a checking sequence for M if every transition of M is verified by CS (see Theorem 1 in [9]). Using Lemma 2 the result follows. □

3.2. A generalization of the method

The method explained so far depends on the existence of a reset input r in the implementation. However, the method can be directly adopted to FSMs without such a reset input, but with a synchronizing sequence. A synchronizing sequence R is an input sequence such that $\delta(s, R) = \delta(s', R)$ for every pair of states s and s'. In other words R resets M to a particular state. Therefore, a synchronizing sequence is in fact effectively a *compound* reset input.

In order to adopt our method to use a synchronizing sequence R instead of a reset input r, the only modifications needed are to replace the occurrence of r in $\gamma_{i,p}$ and $\beta'_{i,x,p}$ with R. This does not change the number of application of locating sequences, which is also $Q_{\text{new}} = 2nk < Q_{[9]}$ in this case.

4. Discussion

The method proposed in this paper will reduce the number of application of locating sequences by a factor greater than $2/(1+q)$, where q is the size of the input alphabet. Given that systems will typically have at least two inputs, and usually more, it means that our method will apply at most two-third of the locating sequences of the previous method, usually less. We thus expect our method to yield significantly shorter CSs.

A necessary condition to use our method is to have some sort of reset capability, which might seem as a limitation. However, being able to use a synchronizing sequence for reset means that our method is in fact widely applicable. Berlinkov [12] claims that a random FSM with n states and q inputs has a synchronizing sequence with probability $1-\Theta(1/n^{0.5\times q})$. This claim was experimentally supported by Kisielewicz et. al [13]. Therefore, our method becomes applicable for virtually all FSMs as the size of the FSM increases.

One optimization that is used for reducing the length of checking sequences when a W-set is used is the following. Based on the knowledge that the given W-set for M is also a W-set for the implementation N, further recognitions for a state s_i can be performed by using a subset $W_i = \{w^i_1, w^i_2, \ldots, w^i_{|W_i|}\}$ of W (see for example [14]). The set W_i, called *an identification set for state* s_i, has the following property: The set of responses of s_i to the sequences in W_i is unique and no subset of W_i has this property. Applying L_{s_i}, $i \in [n]$, guarantees that W is a W-set for the implementation as well. Thus, we can avoid using the entire W-set elsewhere, and simply use the sequences in each W_i's at different places. For the state recognition sequences, each state s_i can be recognized using the sequence $\alpha'_i = L_{s_i} w^i_1 I^1_i L_{s_i} w^i_2 I^2_i \ldots L_{s_i} w^i_{|W_i|}$.

For the reset verification, if the reset input r resets the FSM to state s_1, and if W_1 is the identification set for s_1, then the number of sequences for reset verification can be reduced by using $\gamma'_{i,p} = L_{s_i} r w^1_p, i \in [n], p \in [|W_1|]$ instead of $\gamma_{i,p}$. Although this would in most case further reduce the number of applications of the locating sequence, in the worst case this number remains unchanged.

The idea of verifying resets or synchronizing sequences as an intermediate step in the CS generation algorithm can also be used in the case of FSM having distinguishing sequences. However, in that case, well known methods do generate CS that do not have a particular element which is order of magnitude longer than the other elements of the sequence (i.e. nothing like a locating sequence). Thus, we do not expect that the added cost of a separated reset/synchronizing sequence verification will be justified and the resulting CS would probably be of comparable length, if not longer.

Finally, we note that Lee and Yannakakis introduce the idea of constructing a reliable reset and then using it to build the CS [10]. However, their suggestion was to simulate a reset by transferring from the current state back to the initial state s_1 and then applying the locating sequence L_1. This approach would not reduce the overall length of the CS, since the number of applications of locating sequences would not decrease. In another paper, the same authors suggest a randomized algorithm for FSMs with unreliable reset [15]. However as they point out, the generated sequence may not necessarily be a checking sequence. In contrast our method produces a checking sequence. If one does not require such a guarantee, the approach given in [15] is a reasonable alternative.

References

[1] M. Broy, B. Jonsson, J.-P. Katoen, Model-Based Testing of Reactive Systems: Advanced Lectures (LNCS), Springer, 2005.

[2] A. da Silva Simão, A. Petrenko, Generating checking sequences for partial reduced finite state machines, in: TestCom/FATES, 2008, pp. 153–168.

[3] Z. Kohavi, Switching and Finite State Automata Theory, McGraw-Hill, New York, 1978.

[4] T. S. Chow, Testing software design modelled by finite state machines, IEEE Transactions on Software Engineering 4 (1978) 178–187.

[5] K. Sabnani, A. Dahbura, A protocol test generation procedure, Computer Networks 15 (4) (1988) 285–297.

[6] M. P. Vasilevskii, Failure diagnosis of automata, Cybernetics and Systems Analysis 9 (1973) 653–665, 10.1007/BF01068590.

[7] W. Y. L. Chan, C. T. Vuong, M. R. Otp, An improved protocol test generation procedure based on uios, SIGCOMM Comput. Commun. Rev. 19 (4) (1989) 283–294. `doi:10.1145/75247.75274`.

[8] F. C. Hennie, Fault-detecting experiments for sequential circuits, in: Proceedings of Fifth Annual Symposium on Switching Circuit Theory and Logical Design, Princeton, New Jersey, 1964, pp. 95–110.

[9] A. Rezaki, H. Ural, Construction of checking sequences based on characterization sets, Computer Communications 18 (12) (1995) 911–920.

[10] D. Lee, M. Yannakakis, Testing finite-state machines: State identification and verification, IEEE Transactions on Computers 43 (3) (1994) 306–320.

[11] K. Inan, H. Ural, Efficient checking sequences for testing finite state machines, Information and Software Technology 41 (11–12) (1999) 799–812.

[12] M. V. Berlinkov, On the probability of being synchronizable, CoRR abs/1304.5774.

[13] A. Kisielewicz, J. Kowalski, M. Szykula, Computing the shortest reset words of synchronizing automata, Journal of Combinatorial Optimization (2013) 1–37`doi:10.1007/s10878-013-9682-0`.

[14] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, A. Ghedamsi, Test selection based on finite state models, IEEE Transactions on Software Engineering 17 (6) (1991) 591–603.

[15] M. Yannakakis, D. Lee, Testing finite state machines: Fault detection, J. Comput. Syst. Sci. 50 (2) (1995) 209–227.