Generating Checking Sequences for User Defined Fault Models

Alexandre Petrenko, Adenilso Simao

Generating Checking Sequences for User Defined Fault Models

Alexandre Petrenko

CRIM, Centre de recherche informatique de Montréal
405 Ogilvy Avenue, Suite 101, Montréal (Québec) H3N 1M3, Canada
petrenko@crim.ca

Adenilso Simao

Instituto de Ciencias Matematicas e de Computacao, Universidade de Sao Paulo
Sao Carlos/Sao Paulo, Brazil
adenilso@icmc.usp.br

Abstract. In this paper, we investigate how a checking sequence can be generated from a Finite State Machine, with respect to a user-defined set of faults, modeled as a nondeterministic FSM, called Mutation Machine (MM). We propose an algorithm for generating a checking sequence in this scenario and demonstrate its correctness.

Keywords. FSM testing, fault models, checking sequence, mutation machine

1 Introduction

Generation of checking sequence (CS) from a Finite State Machine (FSM) is a relevant problem, when the implementation may not be reset or when reset operation it prohibitively costly. There are methods which, given a distinguishing sequence, can generate a checking sequence in polynomial time [2] [3]. Other methods generate checking sequence from characterization sets instead of a distinguishing sequence [1], since the former is available for any minimal machine, while the latter may not exist. Those methods, however, rely on the repetition of the sequences in the characterization sets, resulting in an exponentially long sequence. These methods also consider the classical fault domain where the implementation may have arbitrary faults, except extra states.

In this paper, we investigate how a CS can be generated from an FSM, with respect to a subset of faults. The faults of interest are modeled as a nondeterministic FSM, called Mutation Machine (MM), such that any implementation is assumed to be a deterministic submachine of the MM. We propose an algorithm for generating a CS in this scenario. After demonstrating the correctness of the algorithm, we illustrate its application on a simple example.

2 Checking Sequence Construction

An FSM is a tuple $M = (S, S_0, X, O, h)$, where S is the set of states, $S_0 \subseteq S$ is the set of initial states, X is the set of inputs, O is the set of outputs, which satisfy the condition $I \cap O = \varnothing$, and $h \subseteq (S \times X \times O \times S)$ is the set of transitions. For state s and input x, let $h(s, x)$ be the set of transitions from state s with input x. The FSM M is *initialized* if $|S_0| = 1$ and is *deterministic* if for each $(s, x) \in S \times X$, $|h(s, x)| \leq 1$. For an initialized FSM, where $|S_0| = 1$, write s_0, instead of $\{s_0\}$.

The machine M is *completely specified* (complete FSM) if $|h(s, x)| \geq 1$ for each $(s, x) \in S \times X$; otherwise, it is *partially specified* (partial FSM).

A *path* of the FSM $M = (S, S_0, X, O, h)$ from state $s \in S$ is a sequence of transitions $(s_1, x_1, o_1, s_2)\ (s_2, x_2, o_2, s_3)\ \ldots\ (s_k, x_k, o_k, s_{k+1})$, such that $(s_i, x_i, o_i, s_{i+1}) \in h$, for $1 \leq i \leq k$. Notice that we also allow a path to be empty, represented by ε. The machine is *strongly connected*, if it has a path from each state to any other state. The input projection (output projection) of the path is $x_1x_2...x_k$ ($o_1o_2\ldots o_k$). Input sequence $\beta \in I^*$ is a *defined* input sequence in state s of M if it is an input projection of a path from state s. We use $\Omega(s)$ to denote the set of all input sequences defined in state s and $\Omega(M)$ for the states in S_0, i.e., for M. $\Omega(M) = X^*$ holds for any complete machine M, while for a partial FSM $\Omega(M) \subset X^*$.

Given a path p, let $trav(p)$ be the set of transitions of M which appear in p. For state s and input x, let $trans(p, (s, x))$ be the set of transitions from state s with input x in $trav(p)$, i.e., $trans(p, (s, x)) = trav(p) \cap h(s, x)$. For the FSM $M = (S, S_0, X, O, h)$, given a set of states $S' \subseteq S$ and an input sequence α, let $path(S', \alpha)$ be the set of paths of M from states of S' with input projection α. We denote $path(S_0, \alpha)$ by $path_M(\alpha)$. Let $\lambda(S', \alpha)$ be the set of output projections of the paths in $path(S', \alpha)$; we denote $\lambda(S_0, \alpha)$ by $\lambda_M(\alpha)$. Unless stated otherwise, paths are assumed to be from an initial state.

Given states s, $t \in S$ of the deterministic FSM $M = (S, S_0, X, O, h)$, t is *quasi-equivalent* to s, if $\Omega(t) \supseteq \Omega(s)$ and $\lambda(t, \alpha) = \lambda(s, \alpha)$ for all $\alpha \in \Omega(s)$; moreover, in case $\Omega(t) = \Omega(s)$, states are *equivalent*. States s, $t \in S$ are *distinguishable*, if $\lambda(t, \alpha) \neq \lambda(s, \alpha)$ for some $\alpha \in \Omega(t) \cap \Omega(s)$. The machine is *reduced*, if any two states are distinguishable. The quasi-equivalence (equivalence) of two deterministic FSMs is the corresponding relation of their initial states.

Spec $= (S, s_0, X, O, h)$ is an initialized deterministic FSM specification. We assume that it is strongly-connected machine, not necessarily complete and reduced.

Given an FSM $M = (S, s_0, X, O, h)$ and $s \in S$, let M/s be the FSM (S, s, X, O, h), i.e., M initialized in state s. We let s-after-α denote the set of states reached by input sequence α from state s; if α is applied to the initial state of M then we write M-after-α instead of s_0-after-α; for deterministic machines, we write s-after-$\alpha = s'$ instead of s-after-$\alpha = \{s'\}$

We use a so-called *mutation machine* $MM = (S', S'_0, X, O, h')$ which is a completely specified possibly nondeterministic FSM.

FSM $M = (S, s_0, X, O, h)$ is a *submachine* of $MM = (S', S'_0, X, O, h')$ iff $S \subseteq S'$, $s_0 \in S'_0$ and $h \subseteq h'$. Any complete deterministic submachine of MM is one of the *mutants* of *Spec*. The number of mutants is $|S'_0|\ \Pi_{(s, x) \in S \times X} |h'(s, x)|$. For the sets of states S,

inputs X and outputs O, we define the machine $Chaos(S, X, O) = (S, s_0, X, O, (S \times X \times O \times S))$ representing the universe of all FSMs with $|S|$ states.

Let *Prod* be the product of *Spec* and $MM = (S', S'_0, X, O, h')$; the states of *Prod* is a subset of $(S \cup \{\Delta\}) \times S'$. A state (Δ, s) is a Δ-state. The product $Prod = (P, P_0, X, O, H)$, where $P_0 = \{(s_0, s') | \ s' \in S'_0\}$ is such that P and H are the smallest sets satisfying the following rules:

1. If $(s, s') \in P$, $(s, x, o, t) \in h$, $(s', x, o', t') \in h'$, and $o = o'$, then $(t, t') \in P$ and $((s, s'), x, o, (t, t')) \in H$.
2. If $(s, s') \in P$, $(s, x, o, t) \in h$, $(s', x, o', t') \in h'$, and $o \neq o'$, then $(\Delta, t') \in P$ and $((s, s'), x, o', (\Delta, t')) \in H$.

Notice that Δ-states are sink states. If the product has no Δ-states, then any mutant of *MM* is quasi-equivalent to *Spec*.

An input sequence $\omega \in \Omega(Spec)$ is a *checking sequence* for *Spec* w.r.t. *MM*, if for each deterministic submachine N of *MM*, if $\lambda_N(\omega) = \lambda_{Spec}(\omega)$, then N is quasi-equivalent to *Spec*/s, where $s \in S$.

Given a path $p = ((s_1, m_1), x_1, o_1, (s_2, m_2))\ ((s_2, m_2), x_2, o_2, (s_3, m_3))\ ...\ ((s_k, m_k), x_k, o_k, (s_{k+1}, m_{k+1}))$ of the product *Prod* of *Spec* and *MM*, let $p{\downarrow_{MM}}$ be the corresponding path in *MM*, i.e., $p{\downarrow_{MM}} = (m_1, x_1, o_1, m_2)\ (m_2, x_2, o_2, m_3)\ ...\ (m_k, x_k, o_k, m_{k+1})$.

A path of the product *Prod* is *deterministic* (w.r.t. *MM*) if for every state s and input x, $|trans(p{\downarrow_{MM}}, (s, x))| \leq 1$. Given a set of paths Q of *Prod*, let $det(Q)$ be the set of paths of Q which are deterministic (w.r.t. *MM*) and $\Delta(Q)$ be the set of deterministic paths which leads to a Δ-state.

Algorithm for generating a CS for *Spec* w.r.t. *MM*
Input: *Spec* and *MM*
Output: A CS for *Spec* w.r.t. *MM*
$\omega := \varepsilon$
Compute the product *Prod* of *Spec* and *MM*.
While there exists a nonempty shortest input sequence α, such that $\Delta(det(path_{MM}(\omega\alpha))) \neq \varnothing$ **do**
    $\omega := \omega\alpha$
End While
Return ω

Lemma 1. Let ω be an input sequence such that for each input sequence α, we have that $\Delta(det(path_{MM}(\omega\alpha))) = \varnothing$. Then, ω is a checking sequence for *Spec* w.r.t. *MM*.

Proof. Assume that ω is not a checking sequence for *Spec* w.r.t. *MM*, but for each input sequence α, we have that $\Delta(det(path_{MM}(\omega\alpha))) = \varnothing$.

Thus, there exists a deterministic submachine N of *MM*, such that $\lambda_N(\omega) = \lambda_{Spec}(\omega)$, and for each $s \in S$, we have that N is not quasi-equivalent to *Spec*/s. This implies that state N-after-ω is not quasi-equivalent to any state *Spec*/s-after-ω. Then for each $s \in S$, there exists an input sequence $\beta \in \Omega(s)$ such that $\lambda_{N/N\text{-after-}\omega}(\beta) \neq \lambda_{Spec/s\text{-after-}\omega}(\beta)$.

Let $p_{\omega\beta}$ be the path in N which has $\omega\beta$ as the input projection. It follows that $p_{\omega\beta} \in \Delta(det(path_{MM}(\omega\beta)))$, since N is deterministic; moreover, it leads to a Δ-state, since $\lambda_{NN\text{-after-}\omega}(\beta) \neq \lambda_{Spec/s\text{-after-}\omega}(\beta)$,thus, $\Delta(det(path_{MM}(\omega\beta))) \neq \varnothing$, a contradiction. □

Thus, by Lemma 1, if the algorithm stops, the resulting sequence ω is indeed a checking sequence. It remains to show that it will always stop for any specification and mutation machine.

Lemma 2. After a finite number of steps, the algorithm terminates.

Proof. First, notice that for a given deterministic submachine of *MM*, there is exactly one deterministic path with a given input sequence projection (many submachines can share the same path). Thus, the number of paths in $det(path_{MM}(\omega))$ is limited by the number of deterministic submachines of *MM*; as there are finitely many such submachines, there are finitely many paths in $det(path_{MM}(\omega))$. Let Sub_ω be the set of deterministic submachines for which correspond the paths in $det(path_{MM}(\omega))$. At least one path in $det(path_{MM}(\omega))$ leads to a Δ-state, since the algorithm updated ω in the previous iteration to $\omega\alpha$ and $\Delta(det(path_{MM}(\omega\alpha))) \neq \varnothing$.

Let α be a nonempty input sequence, such that $\Delta(det(path_{MM}(\omega\alpha))) \neq \varnothing$. Let $Sub_{\omega\alpha}$ be the set of deterministic submachines each which has a path in $det(path_{MM}(\omega\alpha))$. As Δ-states are sink states, any submachine in Sub_ω with a path to a Δ-state is not in $Sub_{\omega\alpha}$. Thus, there exists at least one submachine which is in Sub_ω but not in $Sub_{\omega\alpha}$. The set of submachines with paths in $det(path_{MM}(\omega))$ is thus reduced each time ω is updated by the algorithm. As the set of submachines is finite, eventually after a finite number of steps the set Sub_ω has no more machines distinguishable from the specification machine *Spec*, which means that for any input sequence α, it holds that $\Delta(det(path_{MM}(\omega\alpha))) = \varnothing$, and the algorithm terminates. □

We now illustrate the application of the algorithm. Consider the FSM in Figure 1a. Observe first that it has no distinguishing sequence. In Figure 1b, we include a mutation machine for which we will generate a checking sequence.

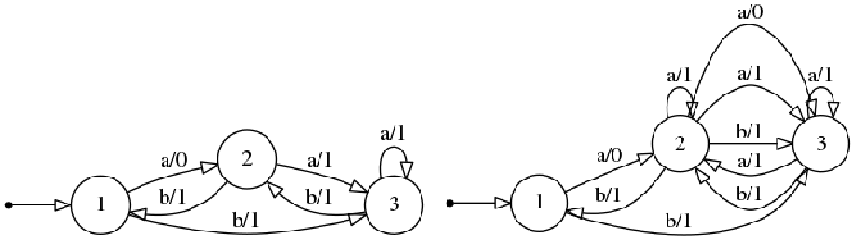

Figure 1. (a) Specification FSM. (b) the Mutation Machine *MM*

Notice that there are twelve deterministic complete submachines of *MM*. One possibility to obtain a checking sequence for *Spec* is to use any of the applicable methods [2] [3], ignoring *MM*. However, the resulting checking sequence would be unnecessarily long.

The algorithm starts by building the product of *Spec* and *MM*, as well as initializing ω with the empty sequence. The nonempty input sequence $\alpha = aa$ is such that $\omega\alpha$ reaches a Δ-state in the product, since $det(path_{MM}(\omega\alpha)) = \{((1, a, 0, 2), (2, a, 1, 3)), ((1, a, 0, 2) (2, a, 0, \Delta)), ((1, a, 0, 2), (2, a, 1, 2))\}$, i.e., $\Delta(det(path_{MM}(\omega\alpha))) \neq \varnothing$. We

append α to ω, so that now ω = *aa*. In the next iteration, the nonempty input sequence α = *ba* is selected, since $det(path_{MM}(\omega\alpha))$ = {((1, *a*, 0, 2), (2, *a*, 1, 3), (3, *b*, 1, 2), (2, *a*, 1, 3)), ((1, *a*, 0, 2), (2, *a*, 1, 2), (2, *b*, 1, 1), (1, *a*, 0, Δ)), ((1, *a*, 0, 2), (2, *a*, 1, 2), (2, *b*, 1, 3), (3, *a*, 1, 3)), ((1, *a*, 0, 2), (2, *a*, 1, 2), (2, *b*, 1, 3), (3, *a*, 1, 2))}, i.e., $\Delta(det(path_{MM}(\omega\alpha))) \neq \emptyset$. We append α to ω, so that now ω = *aaba*. In the next iteration, the nonempty input sequence α = *aba* is selected, since $det(path_{MM}(\omega\alpha))$ = {((1, *a*, 0, 2), (2, *a*, 1, 3), (3, *b*, 1, 2), (2, *a*, 1, 3), (3, *a*, 1, 3), (3, *b*, 1, 2), (2, *a*, 1, 3)), ((1, *a*, 0, 2), (2, *a*, 1, 3), (3, *b*, 1, 2), (2, *a*, 1, 3), (3, *a*, 1, 2), (2, *b*, 1, 1), (1, *a*, 0, Δ)), ((1, *a*, 0, 2), (2, *a*, 1, 3), (3, *b*, 1, 2), (2, *a*, 1, 3), (3, *a*, 1, 2), (2, *b*, 1, 3), (3, *a*, 1, 2)), ((1, *a*, 0, 2), (2, *a*, 1, 2), (2, *b*, 1, 3), (3, *a*, 1, 3), (3, *a*, 1, 3), (3, *b*, 1, 2), (2, *a*, 1, 2)), ((1, *a*, 0, 2), (2, *a*, 1, 2), (2, *b*, 1, 3), (3, *a*, 1, 2), (2, *a*, 1, 2), (2, *b*, 1, 3), (3, *a*, 1, 2))} , i.e., $\Delta(det(path_{MM}(\omega\alpha))) \neq \emptyset$. We append α to ω, so that now ω = *aabaaba*. In the next iteration, the nonempty input sequence α = *bba* is selected, since $det(path_{MM}(\omega\alpha))$ = {((1, *a*, 0, 2), (2, *a*, 1, 3), (3, *b*, 1, 2), (2, *a*, 1, 3), (3, *a*, 1, 3), (3, *b*, 1, 2), (2, *a*, 1, 3), (3, *b*, 1, 2), (2, *b*, 1, 1), (1, *a*, 0, 2)), ((1, *a*, 0, 2), (2, *a*, 1, 3), (3, *b*, 1, 2), (2, *a*, 1, 3), (3, *a*, 1, 3), (3, *b*, 1, 2), (2, *a*, 1, 3), (3, *b*, 1, 2), (2, *b*, 1, 3), (3, *a*, 1, Δ)), ((1, *a*, 0, 2), (2, *a*, 1, 3), (3, *b*, 1, 2), (2, *a*, 1, 3), (3, *a*, 1, 2), (2, *b*, 1, 3), (3, *a*, 1, 2), (2, *b*, 1, 3), (3, *b*, 1, 2), (2, *a*, 1, Δ)), ((1, *a*, 0, 2), (2, *a*, 1, 2), (2, *b*, 1, 3), (3, *a*, 1, 3), (3, *a*, 1, 3), (3, *b*, 1, 2), (2, *a*, 1, 2), (2, *b*, 1, 3), (3, *b*, 1, 2), (2, *a*, 1, Δ)), ((1, *a*, 0, 2), (2, *a*, 1, 2), (2, *b*, 1, 3), (3, *a*, 1, 2), (2, *a*, 1, 2), (2, *b*, 1, 3), (3, *a*, 1, 2), (2, *b*, 1, 3), (3, *b*, 1, 2), (2, *a*, 1, Δ))}, i.e., $\Delta(det(path_{MM}(\omega\alpha))) \neq \emptyset$. We append α to ω, so that ω = *aabaababbba*. There is no nonempty input sequence such that $\Delta(det(path_{MM}(\omega\alpha))) \neq \emptyset$. Thus, by Lemma 1, *aabaababbba* is a checking sequence for *Spec* with respect to *MM*.

Consider now the *Spec* in Figure 1(a) and the corresponding *Chaos*(*S*, *X*, *O*) which represents a traditional fault domain, the universe of all FSMs with up to three states. The algorithm we propose in this paper generates the checking sequence *aaaabaababababbabbabbbbba*, of length 24. On the other hand, the algorithm proposed in [1], generates a checking sequence of length 130.

3 Experimental Results

In this section we present some preliminary experimental results on the length of the checking sequence obtained for various size of a mutation machine. The experiments are set up as follows. For each run, a random complete deterministic FSM *Spec* with 5 states, 2 inputs and 2 outputs is generated, as proposed in [4]. Then, increasingly bigger mutation machines are generated from *Spec* by adding transitions to it. The size of the mutation machine is the number of its transitions; the smallest mutation machine is the specification itself, which the biggest one is the Chaos machine with that a given number of states, inputs and outputs. We executed 30 runs and collected the length of the obtained checking sequence. Figure 2 shows the result of the experiments. We note that, as expected, the length of the checking sequence increases with the size of the mutation machine. However, the increment tends to be smaller, as the number of transitions approaches the maximum.

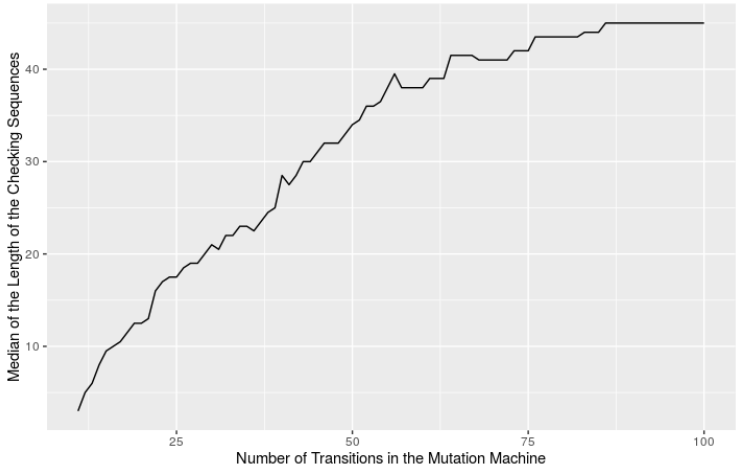

Figure 2. Variation of the Length of the Checking Sequence with respect to the Number of Transitions in the Mutation Machine.

4 Conclusion

In this paper, we proposed an algorithm for generating a checking sequence with respect to a user-defined fault model. In the forthcoming steps of this research, we plan to characterize scenarios when the algorithm can be effectively applied as well as its scalability.

Acknowledgement.

This work was partially supported by MESI (Ministère de l'Économie, Science et Innovation) of Gouvernement du Québec and NSERC of Canada, and by Brazilian Funding Agency FAPESP, Grant 2013/07375-0.

References

1. Ali Rezaki, Hasan Ural: Construction of checking sequences based on characterization sets. Computer Communications 18(12): 911-920 (1995).
2. Adenilso da Silva Simão, Alexandre Petrenko: Generating checking sequences for partial reduced finite state machines. TestCom/FATES 2008: 153-168
3. Robert M. Hierons, Hasan Ural: Optimizing the length of checking sequences. IEEE Trans. Computers 55(5): 618-629 (2006)
4. Adenilso da Silva Simão, Alexandre Petrenko: Checking completeness of tests for finite state machines. IEEE Trans. Computers 59(8): 1023-1032 (2010)