Using Adaptive Distinguishing Sequences in Checking Sequence Constructions

Robert M. Hierons
School of Information Systems, Computing and Mathematics
Brunel University
Uxbridge, Middlesex, UK
rob.hierons@brunel.ac.uk

Guy-Vincent Jourdan Hasan Ural
School of Information Technology and Engineering
University of Ottawa
Ottawa, Canada
{gvj, ural}@site.uottawa.ca

Husnu Yenigun
Faculty of Engineering and Natural Sciences
Sabanci University
Istanbul, Turkey
yenigun@sabanciuniv.edu

ABSTRACT
A number of methods have been published to construct checking sequences for testing from Finite State Machine-based specifications. Many of these methods require the existence of a preset distinguishing sequence in the model. In this paper, we show that usually an adaptive distinguishing sequence is sufficient for these methods to work. This result is significant because adaptive distinguishing sequences are strictly more common and up to exponentially shorter than preset ones.

Categories and Subject Descriptors
D.2.4 [**Software Engineering**] Software & Program Verification – *formal methods, validation*; D.2.5 [**Software Engineering**] Testing and Debugging – *testing tools (data generators, coverage testing)*

General Terms
Reliability, Theory, Verification.

Keywords
Model-based testing, finite state machines, checking sequence construction, adaptive distinguishing sequences, preset distinguishing sequences.

1. INTRODUCTION
State-based systems are often specified using Finite State Machine (FSM) model or its extensions such as Specification and Description Language (SDL) or State-Charts. As a model-based testing approach, testing from an FSM has been utilized in a number of application domains such as telecommunications systems, communications protocols, embedded systems using state-based languages, object-oriented systems, web services, pattern matching and machine learning. When testing from an FSM M it is common to consider a fault model $\Phi(M)$ for M representing the types of faults that can occur and an implementation of FSM M to be tested is taken as an unknown FSM N in $\Phi(M)$. In order to determine whether a given implementation N of M is faulty with respect to M, a *checking sequence* (an input sequence constructed from M by taking into consideration $\Phi(M)$ the fault model for M) is applied to N. Any discrepancy between the expected output sequence and the actual output sequence produced by N in response to the application of the checking sequence is sufficient to determine that N is a faulty implementation of M [1, 2].

The methods reported in the literature for constructing a checking sequence from an FSM M use a distinguishing sequence [1, 3], a set of characterizing sequences [4, 5], or a set of unique input/output (UIO) sequences [6, 7]. These methods recognize each distinct state in N as a distinct state of M and verify that each transition of M is correctly implemented in N [8]. The methods that use a set of characterizing sequences or UIO sequences either require that there is a reset that is known to have been implemented correctly or produce a checking sequence whose length is at least exponential in terms of the number of states of M and the lengths of the sequences used.

Among these previous methods, we will consider the ones using a *distinguishing sequence D* of M which is an input sequence for which the response of each state of M is distinct. Such methods [9, 10, 11, 12, 13] achieve recognition of a state of N as a state of M by applying DT (i.e. the distinguishing sequence D followed by a transfer sequence T) at that state of N and perform the verification of a transition of M from state s_i to state s_j under input x in N by 1) transferring N to the state recognized as state s_i of M; 2) checking the output produced by N in response to x to be as specified in M (to detect an *output fault*); and 3) recognizing the state reached by N after the application of x as state s_j of M (to detect a *transfer fault*). Step 3) is realized by applying a DT at the state reached by N after the application of x. Step 1) is realized indirectly by making sure that the state reached by the application of a DT at each state is recognized by applying another D at some point in the checking sequence. This facilitates the use of a DT at a state such that the state reached by the application of DT is the state to which N needs to be transferred (i.e., state recognized as the starting state of a transition to be verified).

For the construction of a checking sequence of an FSM, these methods utilize the digraph representation of the FSM and first form three sets of paths: set A of *state recognition paths* which are used to recognize each state of the FSM (e.g., α-sequences, α'-sequences or α-elements in different methods); set B of *transition verification paths* which are used to verify each transition of the FSM (e.g., test segments); set C of transfer paths which are used to concatenate paths in A and B. Checking sequence generation methods place various constraints on the selection of C. Earlier methods use some predefined strategies to reduce the length of transfer paths [2, 14]. These strategies do not guarantee that transfer paths found yield minimized checking sequences. An optimization model has been proposed to solve this problem [9] and it is adopted by successive checking sequence generation methods [10, 11, 13].

These methods utilize a *preset* distinguishing sequence, that is, a sequence of inputs which is the same for every state of the FSM M. There exists another type of distinguishing sequences, the *adaptive distinguishing sequences* that are rooted decision trees where each root to leaf path represents an input sequence that is specific to the state represented by the leaf. It has been reported that there may not be a preset or adaptive distinguishing sequence for every FSM and that to determine the existence of a preset distinguishing sequence for a given FSM is PSPACE-complete while that of an adaptive distinguishing sequence is of polynomial complexity [3]. It has been shown that if an FSM has a preset distinguishing sequence, it also has an adaptive one, but the converse is not true (see, for example, [3]). In this paper, we note that often adaptive distinguishing sequences can be used instead of preset ones. We then show that checking sequence construction methods utilizing preset distinguishing sequences can easily be adapted to work with adaptive distinguishing sequences leading to methods that yield shorter checking sequences and more widely useable than when used with preset distinguishing sequences.

The rest of the paper is organized as follows: Section 2 briefly reviews the terminology. Section 3 explains the use of adaptive distinguishing sequences for constructing a checking sequence, and presents the advantages of their use with respect to using preset ones. In Section 4, we review several specific checking sequence construction methods and show that they can be easily adapted to use adaptive distinguishing sequences instead of preset ones. Section 5 concludes the paper.

2. PRELIMINARIES

We represent a deterministic *finite state machine* (FSM) M as $(S, X, Y, \delta, \lambda)$, where S is a finite set of states with $n = |S|$, $s_1 \in S$ is the *initial state*, X is a finite set of inputs with $p = |X|$, Y is a finite set of outputs with $q = |Y|$, δ is a state transition function that maps $S \times X$ to S and λ is an output function that maps $S \times X$ to Y. These two functions are extended to input sequences $I \in X^*$ in the usual manner. A transition from state s_i to state s_j in M is denoted $t_{ij} = (s_i, s_j; x / y)$.

An FSM M is considered *minimal* if, for every pair of states $s_i, s_j \in S$, $i \neq j$, there is an input sequence $I \in X^*$ such that $\lambda(s_i, I) \neq \lambda(s_j, I)$. M is considered *completely specified*, if for each input $x \in X$ and for each state $s_i \in S$, $\delta(s_i, x)$ is defined. M can be represented by a *digraph* $G = (V, E)$ where a set of vertices V represents the set S of states of M, and a set of directed edges E represents all specified transitions of M. Each edge $e = (v_i, v_j; x / y) \in E$, is a state transition from state s_i to state s_j with input $x \in X$ and output $y \in Y$, where v_i and v_j are the *starting* and *terminating* vertices of e (states of t), and input/output (i.e., *i/o* pair) x/y is the label of e, denoted by *label*(e). $G = (V, E)$ is strongly connected, if for each pair of vertices $v_i, v_j \in V$, there exists a path from v_i to v_j.

A *path* $P = (n_1,n_2; x_1/y_1)(n_2,n_3; x_2/y_2)\ldots(n_{r-1},n_r; x_{r-1}/y_{r-1})$, $r > 1$, of $G = (V, E)$ is a finite sequence of adjacent (not necessarily distinct) edges in E, where each node n_i, $1 \leq i \leq r$, represents a vertex of V; n_1 and n_r are called *starting* and *terminating* nodes of P, and the input/output sequence $(x_1/y_1)(x_2/y_2)\ldots(x_{r-1}/y_{r-1})$ is called *label* of P. P is represented by $(n_1,n_r; I/O)$, where I/O is the label of P, $I = x_1x_2\ldots x_{r-1}$ is called the *input portion* of I/O, $O= y_1y_2\ldots y_{r-1}$ is called the *output portion* of I/O. The input portion I of the label I/O of path $(n_1, n_r; I/O)$ will be called the *transfer sequence T* (from n_1 to n_r). The *length* of an *input sequence I* (or *input/output sequence I/O*) is its number of inputs, denoted by $|I|$ (or $|I/O|$). A sequence $i_1i_2\ldots i_k$ is a *subsequence* of $x_1x_2\ldots x_m$ if there exists Δ, $0 \leq \Delta \leq m-k$, such that for all j, $1 \leq j \leq k$, $i_j = x_{j+\Delta}$. Subsequence $i_1i_2\ldots i_k$ is a prefix of $x_1x_2\ldots x_m$ if $\Delta = 0$. Subsequence $i_1i_2\ldots i_k$ is a suffix of $x_1x_2\ldots x_m$ if $\Delta = m-k$.

Let $M = (S, X, Y, \delta, \lambda)$ denote a completely specified, minimal and deterministic FSM, which is represented by a strongly connected digraph $G = (V, E)$. Let the fault model for M, $\Phi(M)$, be the set of FSMs each of which has at most $n = |S|$ states and the same input and output sets as M. Let N be an FSM of $\Phi(M)$. N is *isomorphic* to M if there is a one-to-one and onto function f on the state sets of M and N such that for any state transition $(s_i, s_j; x / y)$ of M, $(f(s_i), f(s_j); x / y)$ is a transition of N. A *checking sequence* of M is an input sequence starting at the initial state s_1 of M that distinguishes M from any N of $\Phi(M)$ that is not isomorphic to M. (i.e., the output sequence produced by any such N of $\Phi(M)$ is different from the output sequence produced by M). In the context of testing, this means that in response to this input sequence, any faulty implementation N from $\Phi(M)$ will produce an output sequence different from the expected output sequence, thereby indicate the presence of one or more faults.

Note that under this definition of a checking sequence we do not care about the initial state of N since isomorphism does not require that the initial states of M and N correspond. Since we wish to apply the checking sequence in the state of N that corresponds to the initial state of M we can precede it by a process that takes M to its initial state irrespective of its current state. Such a process can be produced by starting with a homing sequence [8], whose output identifies the current state, and then moving to the initial state. If we also wish to find initialization faults, where N starts in a different state to M, we should start the checking sequence with a distinguishing sequence and a number of methods do this (see, for example, [13]).

As stated earlier, the recognition of each distinct state in N as a distinct state of M and verification of whether each transition of M is correctly implemented in N are based on distinguishing sequences for the methods considered in this paper. A preset *distinguishing sequence D* of M is an input sequence such that the output sequence produced by M in response to D is different for each state of M (i.e., $\forall s_i, s_j \in S, s_i \neq s_j; \lambda(s_i,D) \neq \lambda(s_j,D)$). A distinguishing sequence D of an FSM M is then used as follows:

Consider a path P of G representing M and the nodes within it. Let $Q = label(P)$.

1. A node n_i of P is *recognized* in Q as state s of M if
   a) n_i is the starting node of a subpath of P whose label is $DT / \lambda(s, DT)$ for some T or

   b) $(n_q, n_i; T / \lambda(s', T))$ and $(n_j, n_k; T / \lambda(s', T))$ are subpaths of P, n_q and n_j are recognized in Q as state s' of M, and node n_k is recognized in Q as state s of M.
2. A transition $t = (s_p, s_q; x / y)$ is *verified* in Q if there is a

subpath (n_i, n_{i+1}; x_i / y_i) of P such that n_i is recognized as s_p in Q, n_{i+1} is recognized as s_q in Q, x_i/y_i = x / y.

A subpath used to recognize a state is called a *state recognition path* for that state. The set of all state recognition paths for a given M and a D of M is denoted by P_S. A subpath used to verify a transition is called a *transition verification path* for that transition. The set of all transition verification paths for a given M and a D of M is denoted by P_T. Paths used to concatenate recognition/verification paths are called *transfer paths* and the input portions of their labels are called *transfer sequences*. For the methods considered in this paper, if this path starts from s_1, recognizes all states of M and verifies all transitions of M, the input portion of this path's label is a checking sequence of M. Several checking sequence generation algorithms are based on the following result [9]:

Theorem 1. Let Q be an IO-sequence of M starting at v_1 (i.e. the label of a path P=(n_1, n_r; Q) of G). If every edge of G is verified in Q then Q is a checking sequence of M.

3. CHECKING SEQUENCES

A preset distinguishing sequence is an input sequence that can be used to distinguish each state of the model. An *adaptive* distinguishing sequence is not really a sequence but a decision tree where each root to leaf path represents an input sequence that is specific to the state represented by the leaf. Formally, an *adaptive distinguishing sequence* is a rooted tree θ with exactly n leaves; the internal nodes are labeled with input symbols, the edges are labeled with output symbols, and the leaves are uniquely labeled with states of the FSM such that: 1) edges emanating from a common node have distinct output symbols, and 2) for every leaf of θ, if D_i and y_i are the input and output strings respectively formed by the node and edge labels on the path from the root to the leaf labeled by state s_i of the FSM then $y_i = \lambda(s_i, D_i)$. We call D_i as the *(adaptive) distinguishing sequence of state s_i*. The length of the sequence is the depth of the tree [3].

Adaptive distinguishing sequences have several core advantages over preset ones, while achieving the same purpose of state identification. The first and most obvious one is that a preset distinguishing sequence is also an adaptive one (for which every path from the root to the leaf has the same input portion on its label), while the reverse is not true. In other words, if an FSM has a preset distinguishing sequence, it has an adaptive distinguishing sequence as well, but there are FSMs that have adaptive distinguishing sequences but do not have a preset distinguishing sequence. That simple fact is important because it shows that if we can use adaptive distinguishing sequences instead of preset ones in a checking sequence generation method, then the method can be used on a strictly larger set of FSMs.

The second advantage of adaptive distinguishing sequences over preset ones is deciding their existence. Current algorithms that determine whether a given FSM has a preset distinguishing sequence require exponential time [1]. Moreover, this is probably unavoidable since it is PSPACE-complete to test whether a given FSM has a preset distinguishing sequence [3]. On the other hand, to decide whether a given FSM has an adaptive distinguishing sequence can be achieved in $O(pn \log n)$ [3]. In other words, deciding whether a checking sequence generation method can be used on a given FSM is at least PSPACE-complete when the method is based on preset distinguishing sequences, while it can take polynomial time[1] when the same method is adapted to use adaptive distinguishing sequences (this does not speak about the complexity of the method itself).

The third advantage is the length of the resulting sequence. It is known that there are FSMs for which the shortest preset distinguishing sequence is of exponential length [3], while an FSM that has an adaptive distinguishing sequence has one of length $O(n^2)$ [15]. What is more, Lee and Yannakakis have proposed an $O(pn^2)$ algorithm to produce an adaptive distinguishing sequence of length at most $n(n-1)/2$ if there is one [3]. Hence using adaptive distinguishing sequences can also provide up to exponential reduction on the length of the checking sequences.

These advantages imply that if it is possible to adapt a checking sequence construction method to use adaptive distinguishing sequences instead of preset ones, then

1. the method will be useable on a strictly larger set of FSMs
2. deciding whether the method can be used on a given FSM will take polynomial time, while it necessarily uses exponential time in the worst case with the preset distinguishing sequences
3. constructing a checking sequence will require polynomial time in the worst case, while it required exponential time in the worst case with preset distinguishing sequences
4. the method will yield a checking sequence that is no longer than the one obtained when using preset distinguishing sequence (since at worst the preset distinguishing sequence can still be used) and can in the best case yield checking sequences that are exponentially shorter than the one produced by the same method used with preset distinguishing sequences.

This shows that if a checking sequence generation method can be altered to use adaptive distinguishing sequences, then it should be adapted and used in this way. In the following, we show that most methods can actually be adapted to use adaptive distinguishing sequences, and thus should be used this way.

3.1 Use of Adaptive Distinguishing Sequences

The intuition behind the use of preset distinguishing sequences is that they are "easier" to use because the input sequence is the same regardless of the state the implementation is in. However, this is only useful when there are several possibilities for the current state of the implementation and we want to identify which state it is actually in. When building a checking sequence, it is usually not required: the checking sequence should trigger the expected output from the implementation, that is, the checking sequence assumes that the implementation is at a known state at all times (and checks that the implementation indeed reacts as if it is in that state). Under these conditions, it is easy to see that when building the checking sequence, it is usually possible to replace any portion of the sequence that uses a (preset) distinguishing sequence with the adaptive distinguishing sequence of the state the implementation is supposed to be in.

[1] It will take polynomial time unless some other characteristics of the FSM required by the method cannot be decided in polynomial time.

We can summarize this fact by stating that any checking sequence construction method that is based on distinguishing sequence and that does not use this distinguishing sequence for another purpose than state verification can be adapted to use *adaptive* distinguishing sequences instead of preset ones, because the checking sequence is built *a priori* and thus must necessarily anticipate the state the implementation should be in. Note that the checking sequence itself is still a preset sequence even if it uses adaptive distinguishing sequences.

3.2 A Sufficient Condition

Before discussing checking sequence generation using adaptive distinguishing sequences, it is important to provide an appropriate formal basis. Since several checking sequence generation algorithms are based on Theorem 1 from [9], one possibility is to extend this result to the use of adaptive distinguishing sequences and we do this in this section. First we extend the notion of recognizing nodes to the case where we are using an adaptive distinguishing sequence.

Consider a path P of G representing M and the nodes within it and let θ be an adaptive distinguishing sequence for M. Let $Q = label(P)$.

1. A node n_i of P is θ-*recognized* in Q as state s_m of M if
   a) n_i is the starting node of a subpath of P whose label is $D_mT / \lambda(s_m, D_mT)$ and we say that $depth(n_i) = 0$; or
   b) $(n_q, n_i; T / \lambda(s', T))$ and $(n_j, n_k; T / \lambda(s', T))$ are subpaths of P, n_q and n_j are θ-recognized in Q as state s' of M, and node n_k is θ-recognized in Q as state s_m of M. Here $depth(n_i) = 1 + \max\{depth(n_q), depth(n_j), depth(n_k)\}$
2. A transition $t=(s_p, s_q; x / y)$ is θ-*verified* in Q if there is a subpath $(n_i, n_{i+1}; x_i / y_i)$ of P such that n_i is θ-recognized as s_p in Q, n_{i+1} is θ-recognized as s_q in Q, $x_i/y_i = x/y$.

We now adapt results from [9] to prove that it is sufficient to θ-verify transitions in generating a checking sequence.

Proposition 1. Let Q be an IO-sequence of M starting at v_1 and let θ be an adaptive distinguishing sequence of M. If every edge of G is θ-verified in Q then for every vertex v_j of G we have that $D_j/\lambda(s_j,D_j)$ is a subsequence of Q.

Proof. By definition, Q is the label of a path $P = (n_1, n_r; Q)$. Given state s_j of M, let n_p denote a node of P with minimum depth that is θ-recognized as s_j. Note that there must be at least one such node since G is strongly connected (hence there is at least one transition leaving s_j) and every edge of G is θ-verified. If $depth(n_p) > 0$ then there exist subpaths $(n_q, n_p; T / \lambda(s', T))$ and $(n_i, n_k; T / \lambda(s', T))$ of P such that n_q and n_i are θ-recognized as state s' of M, n_k is θ-recognized as s_j and $depth(n_p) > depth(n_k)$, contradicting the minimality of $depth(n_p)$. Thus, $depth(n_p) = 0$ and so in P the node n_p is followed by a subpath with label $D_j/\lambda(s_j,D_j)$ as required.

Proposition 2. Let Q be an IO-sequence of M starting at v_1 (i.e. the label of a path $P=(n_1, n_r; Q)$ of G) and let θ be an adaptive distinguishing sequence of M. Let $x_1 \dots x_{r-1}$ be the input portion of Q. Suppose that every edge of G is θ-verified in Q and Q is also an IO-sequence of $M^* = (S^*, X, Y, \delta^*, \lambda^*)$ from $\Phi(M)$ starting at v_1^*. If node n_i is θ-recognized in Q as state s_j of M then $\lambda^*(\delta^*(v_1^*, x_1 \dots x_{i-1}), D_j) = \lambda(\delta(v_1, x_1 \dots x_{i-1}), D_j) = \lambda(s_j, D_j)$.

Proof. The proof will proceed by induction on $depth(n_i)$. The base case is $depth(n_i) = 0$. Here, n_i is followed by a subpath with label $\lambda(s_j,D_j)$ and so the result follows. Inductive hypothesis: the result holds for every node of depth less than l and let $depth(n_i) = l$. Since $depth(n_i) > 0$ there exist subpaths $(n_q, n_i; T / \lambda(s_m, T))$ and $(n_p, n_k; T / \lambda(s_m, T))$ of P such that n_q and n_p are θ-recognized as s_m, n_k is θ-recognized as s_j and $depth(n_i)$ is greater than $depth(n_k)$, $depth(n_p)$, and $depth(n_q)$.

By Proposition 1 we know that M^* has n states and θ is an adaptive distinguishing sequence for M^*. By the inductive hypothesis we know that $\lambda^*(\delta^*(v_1^*, x_1 \dots x_{q-1}), D_m) = \lambda(\delta(v_1, x_1 \dots x_{q-1}), D_m) = \lambda(s_m, D_m)$ and $\lambda^*(\delta^*(v_1^*, x_1 \dots x_{p-1}), D_m) = \lambda(\delta(v_1, x_1 \dots x_{p-1}), D_m) = \lambda(s_m, D_m)$ and so $\delta^*(v_1^*, x_1 \dots x_{q-1}) = \delta^*(v_1^*, x_1 \dots x_{p-1})$ and $\delta(v_1, x_1 \dots x_{q-1}) = \delta(v_1, x_1 \dots x_{p-1})$. Thus, $\delta^*(v_1^*, x_1 \dots x_{i-1}) = \delta^*(v_1^*, x_1 \dots x_{k-1})$ and $\delta(v_1, x_1 \dots x_{i-1}) = \delta(v_1, x_1 \dots x_{k-1})$. By the inductive hypothesis, we have that $\lambda^*(\delta^*(v_1^*, x_1 \dots x_{i-1}), D_j) = \lambda(\delta(v_1, x_1 \dots x_{i-1}), D_j) = \lambda(s_j, D_j)$ and so the result follows.

Theorem 2. Let Q be an IO-sequence of M starting at v_1 (i.e. the label of a path $P = (n_1, n_r; Q)$ of G) and let θ be an adaptive distinguishing sequence of M. If every edge of G is θ-verified in Q then Q is a checking sequence of M.

Proof. Let M^* be an element of $\Phi(M)$ with initial state represented by v_1^* such that Q is an IO-sequence of M^* and let $x_1 \dots x_r$ denote the input portion of Q. By Proposition 1 we know that M^* has n states $s_1^*, \dots, s_n^*$ such that for all $1 \le i \le n$ we have that $\lambda^*(s_i^*, D_i) = \lambda(s_i, D_i)$. It is sufficient to prove that under these conditions we have that for every transition $(s_i, s_j; x/y)$ of M we have that $(s_i^*, s_j^*; x/y)$ is a transition of M^*; from this we can deduce that M and M^* are isomorphic. Given transition $(s_i, s_j; x/y)$ of M there is a corresponding edge $(n_p, n_{p+1}; x/y)$ of P such that n_p is θ-recognized as s_i in Q and n_{p+1} is θ-recognized as s_j in Q. By Proposition 2 we know that $\lambda^*(\delta^*(v_1^*, x_1 \dots x_{p-1}), D_i) = \lambda(\delta(v_1, x_1 \dots x_{p-1}), D_i) = \lambda(s_i, D_i)$ and $\lambda^*(\delta^*(v_1^*, x_1 \dots x_p), D_j) = \lambda(\delta(v_1, x_1 \dots x_p), D_j) = \lambda(s_j, D_j)$ and so $(s_i^*, s_j^*; x/y)$ is a transition of M^* as required.

We have proved a result equivalent to Theorem 1 from [9] that gives a sufficient condition for a test sequence produced using adaptive distinguishing sequence to be a checking sequence. We can thus extend checking sequence generation algorithms that are based on this result to allow adaptive distinguishing sequences and in Section 4 we explain how this can be done.

4. THE PROPOSED METHODS

4.1 The HEN64 Method

In the method proposed by Hennie [2], henceforth called HEN64, the states are recognized in a specific order which takes a permutation of states s_1 to s_n such that a preset distinguishing sequence D followed by a (possibly empty) transfer sequence T is applied at s_i and $s_{i+1} = \delta(s_i, DT)$, for all i, $1 \le i \le n-1$, and for $s_{n+1} = s_1$, D is applied. Thus, one state recognition path is formed as the only element of P_S. Then, the transition verification path for each $(s_j, s_k; x/y)$ is formed by applying a (possibly empty) transfer sequence T_1 from a state s_i to state s_{j-1} and then applying D followed by a (possibly empty) transfer sequence T_2 to reach s_j and finally applying xD where s_i is the terminating state of the only path in P_S or a transition verification path in P_T.

This method can be adapted to adaptive distinguishing sequences [3]. When an adaptive distinguishing sequence is used, then the path that will be used to recognize the states will be formed exactly in the same manner. The only difference will be that instead of DT, D_iT will be applied at each state s_i, where D_i is the adaptive distinguishing sequence of s_i. The use of D within the transition verification paths will also be replaced by adaptive distinguishing sequences. More explicitly, the transition verification path for a transition $(s_j, s_k; x/y)$ will be formed by applying a (possibly empty) transfer sequence T_1 from a state s_i to state s_{j-1} and then applying D_{j-1} followed by a (possibly empty) transfer sequence T_2 to reach s_j and finally applying xD_t where s_i is the terminating state of the only path in P_S or a transition verification path in P_T, and D_t is the adaptive distinguishing sequence of the state $s_t = \delta(s_i, x)$.

4.2 The UWZ97 Method

The method proposed by Ural et al. [9], henceforth called UWZ97, first forms so called state recognition path candidates as concatenations of the application of D followed by a T at each state until the application of the last D is a replication of an earlier application of D at the same state in the concatenated path. Some of these n state recognition path candidates can be a suffix of other state recognition path candidates. In fact, a state recognition path candidate which is not suffix of another state recognition path candidate is taken to be a state recognition path (i.e., an element of P_S), and is called an α-*sequence* in [9]. The number of α-sequences is therefore $k \le n$. Transition verification paths are formed by applying D after the transition's input. UWZ97 finds a shortest sequence containing all α-sequences and transition verification paths connected (possibly) by transfer paths.

When adaptive distinguishing sequences are used, the state recognition path candidates will be formed by using the adaptive distinguishing sequences of the corresponding states instead of the preset distinguishing sequence. In other words, the state recognition path candidate for state s_i will have the label $D_{i1}T_{i1}D_{i2}T_{i2}D_{i3}\ldots D_{ik}$, where $1 \le i_j \le n$, $i_1 = i$, $1 \le j \le k$, such that $s_{ij+1} = \delta(s_{ii}, D_{ij}T_{ij})$. If a state recognition path candidate is a suffix of another candidate, then it will be eliminated and the remaining candidates will similarly be called as α-sequences. The transition verification path of a transition $(s_j, s_k; x/y)$ will have the label xD_iT_i where $s_i = \delta(s_j, x)$.

4.3 The HIU06 Method

The method proposed by Hierons and Ural [13], henceforth called HIU06, is an enhanced version of the method in [9]. There are three main differences between HIU06 and UWZ97. The first one is that, while forming state recognition path candidates as concatenations of the application of D at each state, HIU06 permits the application of the last D in the concatenation to be a replication of an earlier application of D at the same state not necessarily in the same concatenated path. Therefore, some of the n state recognition path candidates can terminate once the last D in the concatenation is found to be a replication of an earlier application of D at the same state in another concatenated path, yielding a shorter state recognition path as an element of P_S. Similar to UWZ97, some state recognition path candidates can be a suffix of other state recognition path candidates and hence can be dropped when forming P_S. The elements of P_S formed in this manner are called α'-*sequences* in [13]. The number of α'-sequences is $k \le n$. The second difference relates to the optimization algorithm used and involves a change that allows optimization to occur over a larger set of checking sequences.

Using adaptive sequences does not have any complicating effect on forming α'-sequences. Similar to the case of UWZ97, the state recognition path candidate for state s_i will have the label $D_{i1}T_{i1}D_{i2}T_{i2}D_{i3}\ldots D_{ik}$, where $1 \le i_j \le n$, $i_1 = i$, $1 \le j \le k$, such that $s_{ij+1} = \delta(s_{ii}, D_{ij}T_{ij})$. α'-sequences will be obtained by eliminating those candidates that are a suffix of another candidate.

The third difference between UWZ97 and HiU06 is that, although a transition verification path is similarly formed by applying a D after the transition's input, a state recognition path is allowed to overlap a transition verification path, as long as the overlap is on the entire length of D. The method decides whether this overlapping should be used or not while forming the checking sequence. When adaptive distinguishing sequences are used, the transition verification path of a transition $(s_j, s_k; x/y)$ will have the label xD_iT_i where $s_i = \delta(s_j, x)$. If there is an α'-sequence that starts from s_i, the occurrence of D_i at the beginning of this α'-sequence may or may not be overlapped with the D_i at the end of the transition verification path of $(s_j, s_k; x/y)$. The method will decide this as it tries to optimize the length of the checking sequence produced.

5. CONCLUSIONS

A checking sequence generated from an FSM M is guaranteed to lead to failures if the implementation FSM N has no more states than M. Many checking sequence generation methods are based on the use of a preset distinguishing sequence D that distinguishes the states of M, despite the negative computational complexity results regarding distinguishing sequences.

This paper has investigated the use of adaptive distinguishing sequences. One of the benefits of using an adaptive distinguishing sequence, rather than a preset distinguishing sequence, is that there are FSMs for which there exists an adaptive distinguishing sequence but no preset distinguishing sequence and the converse is not the case. Further, in contrast to preset distinguishing sequences, there are polynomial time algorithms that decide whether M has an adaptive distinguishing sequence and, if it does, generates such an adaptive distinguishing sequence.

We have argued that when a checking sequence is being produced, adaptive distinguishing sequences can be used in place of preset distinguishing sequences. In addition, recent checking sequence generation algorithms are based on a sufficient condition by Ural et al. [9]. We have proved that the corresponding result holds for adaptive distinguishing sequences. We have also explained how several checking sequence generation algorithms can be altered so that they use adaptive distinguishing sequences.

Future work will consider other checking sequence generation algorithms. It will also involve implementing these new algorithms, that use adaptive distinguishing sequences, and evaluating them on FSMs.

6. ACKNOWLEDGMENTS

This work has been supported in part by grants from the Natural Sciences and Engineering Research Council of Canada, the Ontario Centers of Excellence, and Sabanci University.

7. REFERENCES

[1] Gill, A. *Introduction to the Theory of Finite-State Machines*, McGraw-Hill, NewYork, 1962.

[2] Hennie, F.C. Fault detecting experiments for sequential circuits*. Proc. 5th. Symp. Switching Circuit Theory and Logical Design,* Princeton, N.J., 1964, 95-110.

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

[4] Chow, T. Testing software design modeled by finite-state machines. *IEEE Trans. on Software Eng.* SE-4, (1978), 178-187.

[5] Fujiwara, S., Bochmann, Gv., Khendek, F., Amalou, M., Ghedamsi, A. Test selection based on finite state models. *IEEE Trans. on Software Eng.* 17, 6 (1991), 591-603.

[6] Sabnani, K.K., Dahbura, A.T. A protocol test generation procedure. *Computer Networks* 15, 4 (1988), 285-297.

[7] Dahbura, A.T., Sabnani K.K., Uyar, M.U. Formal methods for generating protocol conformance test sequences. *Proceedings of the IEEE,* 78 (1990), 1317-1325.

[8] Lee, D., Yannakakis, M. Principles and methods of testing finite state machines – a survey. *Proceedings of the IEEE*, 84, 8 (1996), 1089–1123.

[9] Ural, H., Wu, X., Zhang, F. On minimizing the length of checking sequence*. IEEE Trans. on Computers,* 46 (1997), 93-99.

[10] Hierons, R.M., Ural, H. Reduced length checking sequences. *IEEE Trans. On Computers,* 51, 9 (2002), 1111-1117.

[11] Chen, J., Hierons, R.M., Ural, H., Yenigun, H. Eliminating redundant tests in checking sequences. In *Proc. of IFIP TestCom'05*, Montreal, Quebec, 2005, 146-158.

[12] Tekle, K.T., Ural, H., Yalcin, C.M., Yenigun, H. Generalizing redundancy elimination in checking sequences. *Proc. of ISCIS'05*, Istanbul, Turkey, 2005, 915-926.

[13] Hierons, R.M., Ural, H. Optimizing the length of checking sequences. *IEEE Trans. on Computers,* 55, 5 (2006), 618-629.

[14] Gonenc, G. A method for the design of fault detection experiments. *IEEE Trans. on Computers,* 19 (June 1970), 551-558.

[15] Sokolovskii, M.N. Diagnostic experiments with automata. *Kibernetika,* 6 (1971), 44-49.