2009 Seventh IEEE International Conference on Software Engineering and Formal Methods

Checking Sequence Construction Using Adaptive and Preset Distinguishing Sequences

Robert M. Hierons*, Guy-Vincent Jourdan†, Hasan Ural† and Husnu Yenigun‡
*School of Inf. Systems, Comp. and Mathematics
Brunel University, Uxbridge, Middlesex, UK
rob.hierons@brunel.ac.uk
†School of Information Tech. and Engineering
University of Ottawa, 800 King Edward Avenue
Ottawa, Ontario, CANADA K1N 6N5
{gvj,ural}@site.uottawa.ca
‡Faculty of Engineering and Natural Sciences
Sabanci University, Istanbul 34956, TURKEY
yenigun@sabanciuniv.edu

***Abstract*—Methods for testing from Finite State Machine-based specifications often require the existence of a preset distinguishing sequence for constructing checking sequences. It has been shown that an adaptive distinguishing sequence is sufficient for these methods. This result is significant because adaptive distinguishing sequences are strictly more common and up to exponentially shorter than preset ones. However, there has been no study on the actual effect of using adaptive distinguishing sequences on the length of checking sequences. This paper describes experiments that show that checking sequences constructed using adaptive distinguishing sequences are almost consistently shorter than those based on preset distinguishing sequences. This is investigated for three different checking sequence generation methods and the results obtained from an extensive experimental study are given.**

I. Introduction

The Finite State Machine (FSM) model and its extensions such as Specification and Description Language (SDL) or State-Charts are often used to specify state-based systems. As a result, testing from an FSM has been utilized in a number of application domains such as sequential circuits, telecommunications systems, communications protocols, embedded systems, object-oriented systems, web services, pattern matching and machine learning. When testing from an FSM M, a fault model $\Phi(M)$ for M is used to represent the types of faults that can occur in a potential implementation of M. A particular implementation to be tested is then assumed to be functionally equivalent to an unknown FSM N in $\Phi(M)$. In order to determine whether N is a correct implementation of M, a test sequence (a sequence of input/expected output pairs constructed from M) is applied to N [18]. This test sequence is a checking sequence if it has the following property: if $N \in \Phi(M)$ then N passes the test sequence if and only if N is equivalent to M [10], [12].

Several methods have been reported in the literature for constructing a checking sequence from an FSM M. These methods use a distinguishing sequence [10], [17], a set of characterizing sequences [5], [8], or a set of unique input/output (UIO) sequences [19], [7]. 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 by N 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. The methods that use a *distinguishing sequence*[1] D of M yield checking sequences whose lengths are polynomial in the number of states and the length of the distinguishing sequence that is used [22], [13], [4], [21], [14]. They recognize a state of N as a state of M by applying D at that state of N and verify 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 that the output produced by N in response to x is 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 1. is realized indirectly by making sure that each state s is reached by the application of a DI, for some input sequence I, in a given state s' and this is followed by another D in the checking sequence. Step 3. is realized by applying D at the state reached by N after the application of x.

These methods make use of the graphical representation of the FSM and first form three sets of paths: set A of state recognition paths 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 used to verify each transition of the FSM (e.g., test segments); set C of transfer paths used to concatenate paths in A and B. They place various constraints on the selection of C. Earlier methods use some predefined strategies to reduce the length of transfer paths [12], [11]. These strategies do not guarantee that transfer paths found yield minimized checking sequences. An optimization model has been proposed to solve this problem [22] and it is adopted by successive checking sequence generation

[1] an input sequence for which the response of each state of M is distinct

978-0-7695-3870-9/09 $26.00 © 2009 IEEE
DOI 10.1109/SEFM.2009.12

methods [13], [4], [14].

All of these methods utilized a preset distinguishing sequence (henceforth called a *PDS*), that is, a sequence of inputs which is the same for every state of the given FSM. There exists another type of distinguishing sequence called an adaptive distinguishing sequence (henceforth called an *ADS*) that is a rooted decision tree where each root to leaf path represents an input sequence that is specific to the state represented by the leaf. It was first conjectured in [16] that ADSs can be used for checking sequence construction. Later, Boute proved in [2], [3] that ADSs (which are called *distinguishing sets* in [2], [3]) can really be used to construct checking sequences. There are also some recent methods making use of ADSs [6].

Lee and Yannakakis have reported a polynomial time algorithm to check the existence and to construct an ADS for an FSM, whereas checking the existence and constructing a PDS is PSPACE–complete [17]. It is also known that there are FSMs with ADSs for which no PDS exist. These results already make it advantageous to use ADSs rather than PDSs. Furthermore, the least upper bound of the length of PDSs is known to be exponential whereas there is a lower bound of the length of ADSs that is quadratic in the number of states [20], [17]. For a given FSM it is known that the shortest ADS is no longer than the shortest PDS, if both exist. This suggests that using ADSs will lead to shorter checking sequences. However, no practical study on this has been conducted.

In this paper, first we show how three different checking sequence generation methods that were originally described to use PDSs can be modified to use ADSs instead.

A preliminary version of this work appeared in [15] where there were only some informal explanations on how the checking sequence construction methods can be modified in order to use ADSs. In this paper, we provide formal details of the changes needed and explicitly give the modified algorithms for each method. In addition in this paper, these algorithms are proved to generate checking sequences. Finally, in order to be able to understand the practical improvements provided by using ADSs in these methods, this paper also includes an extensive experimental study for all the methods considered.

The rest of the paper is organized as follows: Section 2 reviews the terminology used throughout the paper. Section 3 formalizes the use of ADSs for constructing a checking sequence, and presents the advantages of their use over PDSs. Section 4 reviews several checking sequence construction methods and shows that they can be adapted to use ADSs. Section 5 presents the results of experiments conducted to compare the lengths of the checking sequences constructed using ADSs with those constructed using PDSs. Section 6 gives our concluding remarks.

II. Preliminaries

A deterministic *finite state machine* (FSM) is defined as (S,X,Y,δ,λ), 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.

An FSM is *minimal* if, for every pair of states $s_i, s_j \in S$, $s_i \neq s_j$, there is an input sequence $I \in X^*$ such that $\lambda(s_i,I) \neq \lambda(s_j,I)$. An FSM is *completely specified*, if for each input $x \in X$ and state $s_i \in S$, $\delta(s_i,x)$ is defined. An FSM M can be represented by a directed graph (digraph) $G=(V,E)$ where the set V of vertices represents the set S of states of M and the set E of directed edges represents the transitions of M. An edge $e=(v_i,v_j;x/y) \in E$ represents a transition $t=(s_i,s_j;x/y)$ 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 pair x/y is the label of e.

Path $P = (n_1,n_2;x_1/y_1)(n_2,n_3;x_2/y_2)\dots(n_{r-1},n_r;x_{r-1}/y_{r-1})$, $r > 1$, 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 *starting* and *terminating* nodes of P, and the input/output sequence $(x_1/y_1)(x_2/y_2)\dots(x_{r-1}/y_{r-1})$ is the *label* of P. P is often represented by $(n_1,n_r;I/O)$, where I/O is the label of P, $I=x_1x_2\dots x_{r-1}$ is the *input portion* of I/O, and $O=y_1y_2\dots y_{r-1}$ is the *output portion* of I/O. I/O is said to *label* a path from n_1 to n_r if there is a path $(n_1,n_r;I/O)$. Also, the label I/O of path $(n_1,n_r;I/O)$ is a *transfer sequence* T (from n_1 to n_r). The length of input sequence I (or input/output sequence I/O) is its number of inputs, denoted by $|I|$ (or $|I/O|$). The length (or cost) of an edge or path is the length of its label.

$G=(V,E)$ is strongly connected, if for all $v_i, v_j \in V$, there exists a path from v_i to v_j. In $G=(V,E)$, $indegree_E(v_i)$ and $outdegree_E(v_i)$ represent the number of edges in E terminating and starting at $v_i \in V$, respectively. An Euler path of $G=(V,E)$ is a path that contains every edge in E exactly once. A rural postman path from vertex v_i to vertex v_j over $E' \subseteq E$ is a path that starts at v_i, ends at v_j, and includes all edges of E'. A rural Chinese postman path from v_i to v_j over $E' \subseteq E$ is a rural postman path of minimum-cost. A tour is a path that starts and terminates at the same vertex. Sequence $i_1i_2\dots i_k$ is a subsequence of $x_1x_2\dots x_m$ if there exists $\Delta, 0 \leq \Delta \leq m-k$, such that for all $j, 1 \leq j \leq k$, $i_j = x_{j+\Delta}$.

Let $M=(S,X,Y,\delta,\lambda)$ henceforth denote a completely specified, minimal and deterministic FSM represented by strongly connected digraph $G=(V,E)$. Let the fault model for $M, \Phi(M)$, be the set of FSMs with at most n 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 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/output 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). This means that in response to the input portion of the checking sequence, any faulty implementation N from $\Phi(M)$ will produce an output sequence different from the output portion of the checking

sequence, indicating the presence of faults.

Under this definition of a checking sequence we do not care about the initial state of N; isomorphism does not require that the initial states of M and N correspond. Since we 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 [18], 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 some methods do this (see, for example, [14]).

Recall that 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* (PDS) 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., $\forall s_i, s_j \in S, s_i \neq s_j \Rightarrow \lambda(s_i, D) \neq \lambda(s_j, D)$). A distinguishing sequence D of a given 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 $DI/\lambda(s, DI)$ for some I which is the input portion of a transfer sequence T or
   b) $(n_q, n_i; I/\lambda(s', I))$ and $(n_j, n_k; I/\lambda(s', I))$ are subpaths of P, n_q and n_j are recognized in Q as state s' of M, and n_k is recognized in Q as state s of M.
2) A transition $t = (s_j, s_k; x/y)$ 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 in Q as s_j, n_{i+1} is recognized in Q as s_k, $x_i/y_i = x/y$.

A subpath of P used to recognize a state is a *state recognition path* for that state. A subpath of P used to verify transition t is a *transition verification path* for t. Paths used to concatenate recognition/verification paths are called *transfer paths* and their labels are called *transfer sequences*. For the methods considered in this paper, if path P starts from s_1 and verifies all transitions of M, then this path's label is a checking sequence of M. Several checking sequence generation algorithms are based on the following result [22]:

Theorem 2.1: Let G be a digraph representing M and Q be the label of a path P of G starting at v_1. If every edge of G is verified in Q then Q is a checking sequence of M.

III. Checking Sequences Based on Adaptive Distinguishing Sequences

In this section, we define ADSs and recap their advantages over PDSs. We then give an intuitive idea of why ADSs can often be used instead of PDSs and present a sufficient condition for a sequence to be a checking sequence of M.

A. Advantages of Adaptive Distinguishing Sequences

While a PDS is a single input sequence that can be used to distinguish each state of M, an *adaptive distinguishing sequence* is a rooted tree where each root to leaf path represents an input sequence specific to the state represented by the leaf.

Definition 3.1 ([17]): An *adaptive distinguishing sequence (ADS)* 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 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 x, y are the input and output strings respectively formed by the node and edge labels on the path from the root to the leaf, and if the leaf is labeled by state s_i of the FSM then $y = \lambda(s_i, x)$.

The length of the sequence is the depth of the tree.

In the following, D_i denotes the *ADS* of state s_i of M. ADSs and PDSs can be used for state identification but ADSs offer several advantages:

- ADSs are strictly more general than PDSs: A PDS is an ADS where every path from the root of the tree to a leaf has the same input portion. However, the converse is not true. As a result the set of FSMs having ADSs strictly contains the set of FSMs having PDSs. So if a method based on PDSs can be adapted to use ADSs instead, then this method can be applied to a strictly larger set of FSMs.
- It is easier to find ADSs than PDSs. Current algorithms for determining whether an FSM has a PDS require exponential time [10]. Moreover, this is probably unavoidable since it is PSPACE-complete to decide whether an FSM has a PDS [17]. However, it can be decided whether an FSM has an ADS in $O(pn\log(n))$ and, if such a sequence exists, one can be constructed in $O(pn^2)$ [17]. Thus any method based on PDSs is at least PSPACE-complete, while a method based on ADSs can take polynomial time.
- ADSs can be up to exponentially shorter than PDSs. If an FSM has an ADS, it has one of length $O(n^2)$ [20] and the $O(pn^2)$ algorithm of [17] will produce one of length at most $n(n-1)/2$. There are FSMs for which the shortest PDS is of exponential length [17].

Thus, if a method to generate checking sequence based on PDSs can be modified to use ADSs instead, then

1) this method will be more generally usable (that is, will be usable on a strictly larger set of FSMs),
2) we can decide in polynomial time whether an FSM can be used with this method (instead of exponential time in the worst case when using PDSs),
3) finding the distinguishing sequence will require polynomial time in the worst case, as opposed to exponential time in the worst case for PDSs, and
4) the resulting checking sequence might be up to exponentially shorter.

In the following, we show that many methods can be adapted to use ADSs instead, with the benefits outlined above.

B. From Preset to Adaptive Distinguishing Sequences

PDSs seem easier to handle than ADSs because the input sequence is the same regardless of the current state. A checking sequence is "preset" in that it does not have any branching, so it seems to require PDSs. However, the input sequence being the same regardless of the current state is only important if there are several possibilities for the current state and this usually is not the case for checking sequences. Indeed, they typically anticipate that the implementation must be in a given state and check that the implementation reacts appropriately. When this is the case, it is possible to replace the PDS with the input of the ADS corresponding to the anticipated current state and reach the same conclusion.

Thus, in general, if a method for constructing a checking sequence is based on a PDS and uses this sequence only for state verification purposes, then this method can be adapted to use ADSs. In fact, most of the published methods are of this nature. It should also be noted that the checking sequence itself is still a preset sequence even if it uses ADSs.

C. A Sufficient Condition

Theorem 1 from [22] shows that it is sufficient to verify all the transitions of an FSM to obtain a checking sequence. This result is the basis for several of the checking sequence generation methods published. The theorem is based on PDSs. In this section, we will extend it to the case of ADSs. This will allow us to extend checking sequence generation algorithms based on this result to the case of ADSs. Section IV will explain how this can be done.

We first extend the definitions of state recognition and transition verification to the case of ADSs. Consider a path P of G representing M and the nodes within it and let θ be an ADS of M. Let $Q = label(P)$.

Definition 3.2: A node n_i of P is *θ-recognized* in Q as state s_m of M if

1) n_i is the starting node of a subpath of P whose label is $D_m I/\lambda(s_m, D_m I)$; or
2) $(n_q, n_i; I/\lambda(s', I))$ and $(n_j, n_k; I/\lambda(s', I))$ 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.

In the first case, we say that $depth(n_i) = 0$. In the second case, we say that $depth(n_i) = 1 + max\{depth(n_q), depth(n_j), depth(n_k)\}$.

Definition 3.3: A transition $t = (s_j, s_k; x/y)$ 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 in Q as s_j, n_{i+1} is θ-recognized in Q as s_k, and $x_i/y_i = x/y$.

The following propositions and theorem are adapted from [22] to prove that it is sufficient to verify all transitions in generating a checking sequence.

Proposition 3.1: Let G be a digraph representing M, Q be the label of a path P of G starting at v_1, and θ be an ADS 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: Let $P = (n_1, n_r; Q)$. Given state s_j of M, let n_p denote a node of P with minimum depth that is θ-recognized in Q 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 in Q. If $depth(n_p) > 0$ then there exist subpaths $(n_q, n_p; I/\lambda(s', I))$ and $(n_i, n_k; I/\lambda(s', I))$ of P such that n_q and n_i are θ-recognized in Q as state s' of M, n_k is θ-recognized in Q 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 3.2: Let G be a digraph representing M, Q be the label of a path P of G starting at v_1, and θ be an ADS of M. Let $x_1 \ldots x_{r-1}$ be the input portion of Q. Suppose that every edge of G is θ-verified in Q. Also let $M^* = (S^*, X, Y, \delta^*, \lambda^*)$ be a member of $\Phi(M)$ and P^* be a path of the digraph representing M^* such that P^* has label Q and starts at v_1^*, which represents the initial state s_1^* of M^*. If node n_i is θ-recognized in Q as state s_j of M then $\lambda^*(\delta^*(v_1^*, x_1 \ldots x_{i-1}), D_j) = \lambda(\delta(v_1, x_1 \ldots x_{i-1}), D_j) = \lambda(s_j, D_j)$.

Proof: The proof will proceed by induction on $depth(n_i)$ where 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, $l > 0$, and let $depth(n_i) = l$. Since $depth(n_i) > 0$ there exist subpaths $(n_q, n_i; I/\lambda(s_m, I))$ and $(n_p, n_k; I/\lambda(s_m, I))$ of P such that n_q and n_p are θ-recognized in Q as s_m, n_k is θ-recognized in Q as s_j and $depth(n_i)$ is greater than $depth(n_k), depth(n_p)$, and $depth(n_q)$.

By Proposition 3.1 we know that M^* has n states and θ is an ADS of M^*. By using the induction hypothesis $\lambda^*(\delta^*(v_1^*, x_1 \ldots x_{q-1}), D_m) = \lambda(\delta(v_1, x_1 \ldots x_{q-1}), D_m) = \lambda(s_m, D_m)$. Again by using the induction hypothesis we have that $\lambda^*(\delta^*(v_1^*, x_1 \ldots x_{p-1}), D_m) = \lambda(\delta(v_1, x_1 \ldots x_{p-1}), D_m) = \lambda(s_m, D_m)$. Therefore $\delta^*(v_1^*, x_1 \ldots x_{q-1}) = \delta^*(v_1^*, x_1 \ldots x_{p-1})$ and $\delta(v_1, x_1 \ldots x_{q-1}) = \delta(v_1, x_1 \ldots x_{p-1})$. Thus, $\delta^*(v_1^*, x_1 \ldots x_{i-1}) = \delta^*(v_1^*, x_1 \ldots x_{k-1})$ and $\delta(v_1, x_1 \ldots x_{i-1}) = \delta(v_1, x_1 \ldots x_{k-1})$. The result follows since by the inductive hypothesis, $\lambda^*(\delta^*(v_1^*, x_1 \ldots x_{i-1}), D_j) = \lambda(\delta(v_1, x_1 \ldots x_{i-1}), D_j) = \lambda(s_j, D_j)$. ■

Theorem 3.3: Let G be a digraph representing M, Q be the label of a path P of G starting at v_1, and θ be an ADS of M. If every edge of G is θ-verified in Q then Q is a checking sequence of M.

Proof: Let $M^* = (S^*, X, Y, \delta^*, \lambda^*)$ be a member of $\Phi(M)$ and P^* be a path of the digraph representing M^* such that P^* has label Q and starts at v_1^*, which represents the initial state s_1^* of M^*. Let $x_1 \ldots x_r$ denote the input portion of Q. By Proposition 3.1 we know that M^* has n states $s_1^*, \ldots, s_n^*$ such that for all $1 \leq i \leq n$ 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 in Q as s_i and n_{p+1} is θ-recognized in Q as s_j. By Proposition 3.2 we know

that $\lambda^*(\delta^*(v_1^*,x_1\ldots x_{p-1}),D_i) = \lambda(\delta(v_1,x_1\ldots x_{p-1}),D_i) = \lambda(s_i,D_i)$ and $\lambda^*(\delta^*(v_1^*,x_1\ldots x_p),D_j) = \lambda(\delta(v_1,x_1\ldots x_p),D_j) = \lambda(s_j,D_j)$ and so $(s_i^*,s_j^*;x/y)$ is a transition of M^* as required. ■

IV. Using Adaptive Distinguishing Sequences in Checking Sequence Construction

In this section we describe three methods for producing a checking sequence on the basis of a distinguishing sequence and update them to use an ADS. These methods are chosen for the following purposes. The first method considered in Section IV-A is the first method in the literature. The method considered in SectionIV-B originally formalized the sufficient condition given in Section III-C and is taken as a base by a series of other methods. Finally, the method considered in Section IV-C is the most advanced method in this series of studies mentioned above.

A. The Method of Hennie

In the method proposed by Hennie [12], henceforth called HEN64, the states are recognized in a specific order that is based on a permutation $s_1,\ldots,s_n$ of the states of M. Given this order, for each s_i we define a (possibly empty) transfer sequence T_i such that $D/\lambda(s_i,D)T_i$ labels a path with starting state s_i and terminating state s_{i+1} for $1 \le i \le n$. Since $s_1,\ldots,s_n$ is a permutation we have that s_{n+1} denotes s_1 and s_0 denotes s_n. A single state recognition sequence $D/\lambda(s_1,D)T_1$ $D/\lambda(s_2,D)T_2\ldots D/\lambda(s_n,D)T_nD/\lambda(s_1,D)$ with starting state s_1 is formed. This state recognition sequence is capable of checking that the distinguishing sequence D is also a distinguishing sequence of N, and so that the mapping between states of M and N defined by D is a bijection.

The checking sequence is generated by extending the state recognition sequence $D/\lambda(s_1,D)T_1D/\lambda(s_2,D)T_2\ldots$ $D/\lambda(s_n,D)T_nD/\lambda(s_1,D)$ with subsequences that verify the transitions. Assuming that the current sequence is the label of a path with terminating state s_i, and there remain transitions to be verified, we choose a transition $(s_j,s_k;x/y)$ not yet verified. We extend the current sequence with a transfer sequence T to s_j and then apply xD. The transfer sequence T must lead to x being applied in a state recognized as s_j and so T is a transfer sequence that ends with $D/\lambda(s_{j-1},D)T_{j-1}$. This process iterates until transition verification sequences for all transitions are included. Transition verification sequences for the transitions are added in the order in which the transitions appear in the transition table.

When an ADS is used, the path used to θ-recognize the states is formed in the same manner. We define a permutation of the states $s_1,\ldots,s_n$ with s_1 as the initial state. For each s_i, $1 \le i \le n$, we define transfer sequence T_i such that $D_i/\lambda(s_i,D_i)T_i$ labels a path with starting state s_i and terminating state s_{i+1}. We obtain state recognition sequence $D_1/\lambda(s_1,D_1)T_1D_2/\lambda(s_2,D_2)T_2\ldots D_n/\lambda(s_n,D_n)$ $T_nD_1/\lambda(s_1,D_1)$ with starting state s_1. To check transition $(s_j,s_k;x/y)$ we extend the current sequence with a transfer sequence T to s_j and apply xD_k. As before, T must lead to x being applied in a state θ-recognized as s_j and so T ends with $D_{j-1}/\lambda(s_{j-1},D_{j-1})T_{j-1}$. This iterates until transition verification sequences for all transitions are included.

This is summarized in Algorithm 1.

Algorithm 1 checking sequence generation algorithm [12]

```
Input M and ADS θ of M.
Define a permutation s_1,...,s_n of the states of M.
for all 1 ≤ i ≤ n do
    define a transfer sequence T_i such that D_i/λ(s_i,D_i)T_i
    labels a path with starting state s_i and terminating state
    s_{i+1}, where s_{n+1} denotes s_1
end for
Let CS = D_1/λ(s_1,D_1)T_1D_2/λ(s_2,D_2)T_2...D_n/λ(s_n,D_n)
    T_nD_1/λ(s_1,D_1)
Let Tr denote the transitions of M listed in the order in
    which they appear in the transition table.
while Tr ≠ ε do
    Let CS_I be the input portion of CS
    Let s_i = δ(s_1,CS_I)
    Let (s_j,s_k;x/y) = head(Tr)
    Choose some transfer sequence T that ends with
    D_{j-1}/λ(s_{j-1},D_{j-1})T_{j-1} and that is the label of a path
    with starting state s_i.
    let CS = CSTx/yD_k/λ(s_k,D_k).
    Let Tr = tail(Tr)
end while
Output CS.
```

Proposition 4.1: The I/O sequence CS returned by Algorithm 1 is a checking sequence of M.

Proof:

Let G be a digraph representing M and the label of a path P of G starting at v_1 be CS. Let $(s_j,s_k;x/y)$ denote a transition of M. It is clear that P contains a node n_j that is θ-recognized in CS as state s_j and is followed by an edge with label x/y whose terminating node is θ-recognized in CS as s_k. Thus, every edge of G is verified in CS. Further, the starting node of P is θ-recognized in CS as state s_1 by D_1. The result thus follows from Theorem 3.3. ■

The worst case time complexity of this algorithm is the same as that of HEN64, which is of $O(pn^3)$.

B. The method of Ural, Wang and Zhang

The method proposed by Ural et al. [22], henceforth called UWZ97, forms state recognition paths as concatenations of $D/\lambda(s_i,D)$ followed by a transfer sequence T_i at each state s_i until the application of the last $D/\lambda(s_i,D)T_i$ is a repeat of an earlier $D/\lambda(s_i,D)T_i$ in the path. The sequences used are defined in the following way. The first step is to choose subsets $V_k \subseteq V$ $(1 \le k \le q)$ of V whose union is V. The elements within each V_k are ordered giving $V_k = \{v_1^k,\ldots,v_{n_k}^k\}$, where the state represented by v_i^k is denoted $s_{m(i,k)}$. For each v_i^k, they obtain a sequence $D/\lambda(s_{m(i,k)},D)T_i^k$, which is the result of applying D in state $s_{m(i,k)}$ followed by a transfer sequence T_i^k whose

terminating state corresponds to v^k_{i+1} ($v^k_{n_k+1}$ can be any v^k_w, $1 \leq w \leq n_k$). For each V_k, they form a path P_k with starting state $s_{m(1,k)}$ and label $\alpha_k = D/\lambda(s_{m(1,k)},D)T^k_1 D/\lambda(s_{m(2,k)},D)$ $T^k_2 \ldots D/\lambda(s_{m(n_k,k)},D)T^k_{n_k} D/\lambda(s_{m(w,k)},D)T^k_w$, $1 \leq w \leq n_k$. The set $A = \{\alpha_1, \ldots, \alpha_q\}$ is called an α-set and each sequence $\alpha_i \in A$ is called an *α-sequence from A*. If A is clear, its members are called *α-sequences*. The transfer sequence, that follows the execution of D from state s_i, is denoted T_i.

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, possibly connected by transfer paths. A preset acyclic subset of transitions is chosen and the transfer paths are only allowed to contain transitions from this subset in addition to subpaths of the form $(s_i, s_j; D/\lambda(s_i,D)T_i)$. It is proved that any I/O sequence constructed in this way defines a checking sequence. We now describe this method in detail, showing how it can be applied using ADSs.

When ADSs are used, the state recognition paths are formed by using the ADS of the corresponding states. The α-sequences are defined in the following way. First we choose $V_k \subseteq V$ $(1 \leq k \leq q)$ whose union is V and order the elements in each V_k, giving $V_k = \{v^k_1, \ldots, v^k_{n_k}\}$, the state represented by v^k_i being called $s_{m(i,k)}$. For each v^k_i, we obtain $D_{m(i,k)}/\lambda(s_{m(i,k)},D_{m(i,k)})T^k_i$; the result of applying $D_{m(i,k)}$ in state $s_{m(i,k)}$ followed by a transfer sequence T^k_i whose terminating state corresponds to v^k_{i+1}, where $v^k_{n_k+1}$ can be any v^k_w, $1 \leq w \leq n_k$. For each V_k, we form a path P_k from v^k_1 with label $\alpha_k = D_{m(1,k)}/\lambda(s_{m(1,k)},D_{m(1,k)})T^k_1 D_{m(2,k)}/$ $\lambda(s_{m(2,k)},D_{m(2,k)})T^k_2 \ldots\ldots D_{m(n_k,k)}/\lambda(s_{m(n_k,k)},D_{m(n_k,k)})$ $T^k_{n_k} D_{m(w,k)}/\lambda(s_{m(w,k)},D_{m(w,k)})T^k_w$, $1 \leq w \leq n_k$. The set $A = \{\alpha_1, \ldots, \alpha_q\}$ is an α-set and each $\alpha_i \in A$ is an *α-sequence from A*. If A is clear, its members are simply called *α-sequences*. The transfer sequence, that follows the execution of D_i from state s_i, is denoted T_i and its input portion is denoted I_i. The transition verification path of $(s_j, s_k; x/y)$ will have label $x/yD_k/\lambda(s_k,D_k)T_k$ and starting state s_j. Again, a single sequence is constructed from the α-sequences and transition verification sequences using a preset acyclic subset of transitions and subpaths of the form $(s_i, s_j; D_i/\lambda(s_i,D_i)T_i)$ as follows.

An auxiliary digraph $G' = (V', E')$ is formed using $G = (V, E)$ in which $V' = V \cup U'$ where $U' = \{v'_i | v_i \in V\}$ and $E' = E \cup E_\alpha \cup E_T \cup E_c \cup E_\varepsilon \cup E''$ defined by the following.

1) $E_\alpha = \{(v_i, v'_j, \alpha_k) | 1 \leq k \leq q \wedge v_i$ is the starting node of $P_k \wedge v_j$ is the terminating node of $P_k\}$ is a set of edges representing the α-sequences.
2) $E_T = \{(v_i, v'_j, D_i/\lambda(s_i,D_i)T_i) | \delta(s_i, D_i I_i) = s_j\}$ is a set of edges each representing the ADS followed by the corresponding transfer sequence.
3) $E_c = \{(v'_i, v'_j, xD_l/\lambda(s_i,xD_l)T_l) | \delta(s_i,x) = s_l \wedge \delta(s_i, xD_l I_l) = s_j\}$ is a set of edges representing transition verification paths.
4) $E_\varepsilon = \{(v'_i, v_i, \varepsilon) | 1 \leq i \leq n\}$ is a set of edges that, for each state s_i, connects v'_i to v_i without introducing any input or output.
5) $E'' \subset \{(v'_i, v'_j, x/y) | \delta(s_i,x) = s_j \wedge \lambda(s_i,x) = y\}$ is a set of edges that are copies of edges of E with the property that (U', E'') is acyclic, E'' is a spanning tree for U', and G' is strongly connected.

The set E'' can be generated in the manner described in [22] and so we do not give the details of this process here.

The key point is that if we use a path P in G' that includes every edge in E_α then every edge of P that ends at a vertex in U' and is not in E'' has a terminating node that is θ-recognized in the label of P as the corresponding state. In addition, for each transition of M there is a transition verification path in E_c with a starting vertex in U'. Checking sequence generation involves finding a Rural Chinese Postman path of G' that starts at v_1 and contains every edge in $E_\alpha \cup E_c$; it is sufficient to prove that by requiring (U', E'') to be acyclic we ensure that every edge of G is verified. The above checking sequence generation process is summarized in Algorithm 2.

Algorithm 2 checking sequence generation algorithm [22]

1: Input M, ADS θ of M and α-set A with transfer sequences $T_1, \ldots, T_n$.
2: Generate the digraph G'.
3: Obtain $G^* = (V^*, E^*)$ from G' by adding a new vertex σ and new edges $(v_i, \sigma; \varepsilon), i = 1, \ldots, |V'|$, and $(\sigma, v_1; \gamma)$. Then, $V^* = V' \cup \{\sigma\}$ and $E^* = E' \cup \{(v_i, \sigma; \varepsilon), i = 1, \ldots, |V'|\} \cup \{(\sigma, v_1; \gamma)\}$.
4: Let the cost of every new edge be zero except the cost of edge $(\sigma, v_1; \gamma)$ which is very large.
5: Let $E^+ \subset E^*$ be $\{(\sigma, v_1; \gamma)\} \cup E_\alpha \cup E_c$ and find a minimum-cost tour Γ in G^* containing every edge of E^+.
6: Delete vertex σ from Γ to obtain a minimum-cost path (Rural Chinese Postman Path) P of G^* that starts at v_1 and contains every edge in $E_\alpha \cup E_c$.
7: Return the I/O sequence $Q = label(P)$.

We now prove that Algorithm 2 returns a checking sequence by adapting the proof from [22].

Theorem 4.2: The I/O sequence $Q = label(P)$ produced by Algorithm 2 is a checking sequence of M.

Proof: We will use proof by contradiction, assuming that Q is not a checking sequence. Let R denote the starting nodes of edges from E_c that are in P: by Theorem 3.3 it is sufficient to prove that each node from R is θ-recognized in Q.

Since (U', E'') is acyclic and E'' forms a spanning tree for U', the edges in E'' define a partial order $\propto$ on V by $v_i \propto v_j$ if and only if there is a path in (U', E'') from v'_i to v'_j and we can order the nodes in R according to their corresponding vertices. Thus, amongst the nodes in R that are not θ-recognized in Q we can choose a node n_i that is minimal according to $\propto$.

We can now consider the node n_{i-1}; n_i cannot be n_1 since P starts at v_1 and not a vertex in U'. Observe that the terminating node of an edge from $E_\alpha \cup E_T \cup E_c$ is θ-recognized in Q and so the edge from n_{i-1} to n_i must represent an edge from E'' that corresponds to a transition $t = (s_k, s_l; x/y)$ of M. By the

minimality of n_i we know that n_{i-1} is θ-recognized in Q as s_k. In addition, P contains an edge $(n_r, n_j, xD_l/\lambda(s_k, xD_l)T_l)$ representing a transition verification path for t. But $n_r \propto n_i$ and so by the minimality of n_i we must have that n_r is θ-recognized in Q. By the definition of a node being θ-recognized, as n_{i-1} is θ-recognized in Q and there is a transition verification path for t we must have that n_i is θ-recognized in Q and this provides a contradiction as required. ■

Given M and an ADS θ of M the worst case time complexity of this algorithm is the same as that of UWZ97, which is of $O(pn^2 \log n)$ if algorithms described in [1] for finding a Rural Chinese Postman Tour are used.

C. The method of Hierons and Ural

The method of Hierons and Ural [14], henceforth called HIU06, is an enhanced version of UWZ97. There are three main differences between HIU06 and UWZ97. The first is that, while forming state recognition paths, HIU06 permits the application of the last D in the concatenation to be a replication of an application of D at this state but not necessarily in the same concatenated path. Thus, a state recognition path can terminate once the last D in the concatenation is a replication of an application of D at the same state in another path, yielding shorter state recognition paths. The elements formed in this manner are called α'-sequences [14]. The second difference is that the optimization algorithm used allows optimization to occur over a larger set of checking sequences. The third difference is that, although a transition verification path is 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. We now update the HIU06 method to the use of an ADS θ.

The α'-sequences are defined in the following way. The first step is to partition V into $V_1, \ldots, V_q$ and to order the elements within each V_k giving $V_k = \{v_1^k, \ldots, v_{n_k}^k\}$, where $s_{m(i,k)}$ denotes the state represented by v_i^k. For each v_i^k, produce a sequence $D_{m(i,k)}/\lambda(s_{m(i,k)}, D_{m(i,k)})T_i^k$; the result of applying $D_{m(i,k)}$ in state $s_{m(i,k)}$ followed by a transfer sequence T_i^k whose terminating state corresponds to v_{i+1}^k ($v_{n_k+1}^k$ can be any v_w^j, $1 \leq j \leq q, 1 \leq w \leq n_j$). For each V_k, form a path P_k from $s_{m(1,k)}$ with label α_k where $\alpha_k = D_{m(1,k)}/\lambda(s_{m(1,k)}, D_{m(1,k)})T_1^k D_{m(2,k)}/\lambda(s_{m(2,k)}, D_{m(2,k)})T_2^k \ldots\ldots D_{m(n_k,k)}/\lambda(s_{m(n_k,k)}, D_{m(n_k,k)})T_{m_k}^k D_{m(w,j)}/\lambda(s_{m(w,j)}, D_{m(w,j)})T_w^j$ ($1 \leq j \leq q$, $1 \leq w \leq n_j$). The set $\{\alpha_1, \ldots, \alpha_q\}$ is called an α'-set. Given A, each sequence $\alpha_i \in A$ is called an *α'-sequence from A*. If A is clear, its members are simply called *α'-sequences*. The transfer sequence, that follows D_i from state s_i, is denoted T_i.

The α'-sequences play the following roles in checking sequence generation. First they verify that the ADS θ used is also an ADS of the SUT. For each state s_i they also θ-recognize the terminating state (say s_j) of the path from s_i with label $D_i/\lambda(s_i, D_i)T_i$. Finally, an α'-sequence α_k from A that has stating state s_i begins with D_i and thus its starting node is θ-recognized. Thus, an α'-sequence can be used to check the terminating state of a transition.

Each α'-sequence is represented by an edge in a set called E_α: for every $\alpha_i \in A$ with starting state s_j and terminating state s_k, E_α contains an edge from v_j to v_k with label α_i.

The problem of producing an α'-set using an ADS is almost identical to that of producing an α'-set with a PDS, a problem addressed in detail in [14]. We therefore assume that an α'-set $A = \{\alpha_1, \ldots, \alpha_q\}$ has been found for the ADS θ.

The following gives a sufficient condition for an I/O sequence to be a checking sequence of a given M and is a result from [14] changed in order to allow an ADS to be used.

Theorem 4.3: Let G be a digraph representing M, A denote an α'-set and $G_\Upsilon = (V, E \cup E_\Upsilon)$ for some E_Υ that satisfies the following properties:

1) For each transition τ, with terminating state s_j, E_Υ contains one edge representing τ followed by a path whose label is either $D_j/\lambda(s_j, D_j)T_j$ or is from A.
2) For every α'-sequence $\alpha_k \in A$, E_Υ contains one edge that represents a path with label α_k or a transition τ followed by a path with label α_k.
3) Every edge from E_Υ represents a path whose label is an α'-sequence or a transition τ, with terminating state s_j, followed by a path whose label is either a sequence from A or $D_j/\lambda(s_j, D_j)T_j$.

Let us suppose that Γ is a tour of G_Υ that contains every edge from E_Υ. Let $e \in E_\Upsilon$ represent the transition verification path for a transition τ whose terminating state is s_1. Let Γ' denote Γ with e replaced by the corresponding sequence $e_1, \ldots, e_k$ of edges from G (so e_1 represents τ) and let P denote the path formed by starting Γ' with edge e_2. Let $G[E_C]$ denote the digraph induced by the set of edges in P that are not in E_Υ and let us suppose that $G[E_C]$ is acyclic. If $Q = label(P)$ then $QD_1/\lambda(s_1, D_1)$ is a checking sequence of M.

Proof:

A proof by contradiction will be produced: assume that $QD_1/\lambda(s_1, D_1)$ does not represent a checking sequence. Then, by Theorem 3.3, some of the nodes of P are not θ-recognized in $QD_1/\lambda(s_1, D_1)$. As every node following an edge from E_Υ is θ-recognized in $QD_1/\lambda(s_1, D_1)$, any node that is not recognized must follow an edge from E_C.

It is possible to place a partial order $\propto$ on the vertices of G_Υ such that $v_i \propto v_j$ if and only if there is a path in $G[E_C]$ from v_i to v_j. This partial order can be extended to the nodes, which are ordered according to their corresponding vertices. Amongst the nodes that are not θ-recognized in $QD_1/\lambda(s_1, D_1)$, take some n_i that represents a vertex that is minimal according to $\propto$. There may be more than one such node, but any one will suffice. Clearly, i cannot be 1 as n_1 is θ-recognized in $QD_1/\lambda(s_1, D_1)$ as s_1 by D_1.

It is sufficient to look at the node n_{i-1} that precedes n_i. The edge from n_{i-1} to n_i must represent some transition t that is represented by an edge in E_C, as its terminating node is not θ-recognized in $QD_1/\lambda(s_1, D_1)$, and thus $n_{i-1} \propto n_i$. By the minimality of n_i, n_{i-1} is θ-recognized in $QD_1/\lambda(s_1, D_1)$.

The path P contains an edge e, from node n_j to n_{j+1} say, that tests t. As $n_j \propto n_i$, by the minimality of n_i the node n_j must be θ-recognized in $QD_1/\lambda(s_1,D_1)$. Thus, in e, the transition t exists within a context in which it is followed by some path with label $D_l/\lambda(s_l,D_l)T_l$ (possibly as part of an α'-sequence) and its starting node is θ-recognized in $QD_1/\lambda(s_1,D_1)$. Thus, by the definition of a node being θ-recognized, as n_{i-1} is θ-recognized in $QD_1/\lambda(s_1,D_1)$we have that n_i is θ-recognized in $QD_1/\lambda(s_1,D_1)$. This provides a contradiction as required. ■

We now investigate the problem of producing a minimal length tour satisfying these conditions. We produce a network W from $G=(V,E)$ and derive the minimum cost/maximum flow (min cost/max flow) F of W. W has vertex set $\{s,t\} \cup \{s'_1,\ldots,s'_n\} \cup \{t'_1,\ldots,t'_n\}$ with source s and sink t. A node of the form s'_i represents being in state s_i after a transition being tested and before an α'-sequence or $D_i/\lambda(s_i,D_i)T_i$ and the t'_i represent nodes before the start of a transition verification path. The edges are defined by the following.

1) For each i, there is an edge from s to s'_i with capacity $indegree_E(v_i)$ and cost 0 since each edge of G with terminating vertex v_i represents a transition that needs to be followed by a path whose label is an α'-sequence or $D_i/\lambda(s_i,D_i)T_i$.
2) For each i, there is an edge from t'_i to t with capacity $outdegree_E(v_i)$ and cost 0 as $outdegree_E(v_i)$ edges leaving v_i represent transitions to be tested.
3) For each $\alpha_k \in A$ from v_i to v_j there is an edge from s'_i to t'_j with capacity 1 and cost $|\alpha_k|$ that represents the execution of α_k for verifying a transition.
4) For states s_i and s_j that are the starting and terminating states of a path with label $D_i/\lambda(s_i,D_i)T_i$ there is an edge from s'_i to t'_j with capacity $indegree_E(v_i) - outdegree_{E_{\alpha'}}(v_i)$ and cost $|D_i/\lambda(s_i,D_i)T_i|$. This edge represents the use of the sequence $D_i/\lambda(s_i,D_i)T_i$ to θ-recognize the terminating state of a transition in a transition verification path. The capacity is the number of transitions that will be followed by a path with label $D_i/\lambda(s_i,D_i)T_i$ but not an α'-sequence in the tour.
5) For each transition from s_i to s_j there is an edge from t'_i to t'_j with infinite capacity and cost 1 representing an edge used to connect elements of E_Υ.

The execution of $(s_i,s_j,x/y)$ as part of a transition verification path is represented by flow from t'_i to t and flow from s to s'_j while the execution of $D_j/\lambda(s_j,D_j)T_j$ for verifying a transition is represented by flow from s'_j to some t'_k.

The min cost/max flow F is found and this can be produced in low order polynomial time (see, for example, [1]). As in [14], we produce a digraph $G'=(V',E')$ on the basis of F. G' has vertex set $V'=\{a_1,\ldots,a_n\} \cup \{b_1,\ldots b_n\}$ and edge set E' that is defined by the following.

1) For each transition τ from s_i to s_j in M there is an edge from b_i to a_j representing the execution of τ as part of a transition verification path.
2) Given an edge from s'_i to t'_j in W with flow f in F there are f corresponding edges from a_i to b_j, each representing the use of some α_k or $D_i/\lambda(s_i,D_i)T_i$.
3) Given an edge from t'_i to t'_j in W with flow f in F, there are f corresponding edges from b_i to b_j, representing the execution of transitions used to connect transition verification paths.

As flow is conserved at vertices the digraph G' is symmetric and so if G' is connected then it has an Euler Tour Γ [9] and from this we can produce a checking sequence of length $cost(F)+n\times p+|D|$, where $cost(F)$ denotes the cost of the flow F. If G' is not connected then a set of tours can be produced. Otherwise the tours can be connected by adding further transitions in an identical way to that described in [14].

We now choose an edge e in Γ that represents a transition verification path for a transition with terminating state s_1 and in Γ we replace e by the corresponding sequence $e_1,\ldots,e_k$ of edges from G to give a tour Γ'. We start Γ' with e_2 to form a path P with label Q. The I/O sequence $QD_1/\lambda(s_1,D_1)$ forms a checking sequence of M. Algorithm 3 summarizes this.

Algorithm 3 checking sequence generation algorithm [14]

```
Input M, ADS θ of M and α'-set A with transfer sequences T_1,...,T_n.
Produce network W and a min cost/max flow F for W.
Generate the digraph G'.
if G' is strongly connected then
    produce an Euler Tour Γ of G' and otherwise produce a set of tours and connect these tours to form a single tour Γ.
end if
Choose an edge e in Γ that represents a transition verification path for a transition with terminating state s_1.
Replace e in Γ by the sequence e_1,...,e_k of edges of G that correspond to e to form Γ'.
Let P denote the path produced by starting Γ' with e_2.
Let Q = label(P).
Return the I/O sequence QD_1/λ(s_1,D_1).
```

The proof of the following is identical to the proof of the corresponding result in [14].

Lemma 4.4: The set of edges between the t'_i, with non-zero flow in F, defines an acyclic subgraph of G.

We now prove that Algorithm 3 produces a checking sequence.

Theorem 4.5: The I/O sequence produced by Algorithm 3 is a checking sequence of M.

Proof: By Lemma 4.4 the set of edges between the t'_i, that have non-zero flow in F, define an acyclic digraph. Further, each edge from E_Υ is included in the resultant sequence. The result thus follows from Theorem 4.3. ■

The time complexity is identical to that of the algorithm given in [14], which is $O(pn^2\log n)$.

V. Experimental Study

In this section we describe the experiments carried out to compare the performance of the checking sequence gener-

ation methods when PDSs and ADSs are used. Below an "improvement" will mean an improvement that is obtained when an ADS is used instead of a PDS. Therefore, a negative improvement will mean that the method performs better when a PDS is used.

For the experiments, deterministic, minimal, completely specified FSMs with PDSs and that are represented by strongly connected digraphs were used. These FSMs were randomly generated as follows. To generate such a random FSM with n states, p inputs and q outputs, first a random digraph with n nodes, each with outdegree p, was created by randomly assigning the terminating vertex of an edge to be any one of the n vertices. We required strongly connected digraphs so the set of strongly connected components was found. If there were more than one such component, then a set of edges was picked from each component (which we call free edges) such that the removal of these edges did not stop this component being strongly connected. The terminating vertices of the free edges were reassigned randomly to vertices in other strongly connected components. This redirection of the free edges was performed iteratively until a strongly connected graph was formed. Once a strongly connected graph was formed, p input labels are randomly assigned to the p outgoing edges of each node. The output labels were randomly assigned to the edges in the graph where $q = p$. This was followed by a minimality check and then a check for the existence of a PDS. If the FSM passes both of these two tests, then it was included into the set of FSMs to be used for the experimental study.

Note that, using ADSs for checking sequence generation is obviously superior to using PDSs when FSMs without a PDS but with an ADS are considered. In order to understand how the performance of the checking sequence generation methods are improved when both PDS and ADS exist, FSMs with a PDS were used (which also implies the existence of an ADS for these FSMs). No special type of PDS or ADS was searched for and the distinguishing sequences that found first were used.

There were 10 groups of FSMs where each FSM in the same group has the same number of states n, where $n \in \{10, 20, \ldots, 100\}$. There were 800 FSMs in each group, hence there were 8000 FSMs in total.

One performance measure for the checking sequence generation methods is the time it takes to generate checking sequences. The generation of PDS and ADS can be considered as part of the checking sequence generation methods. However, it is not the aim of this work to compare the time performances of PDS and ADS generation methods. Therefore in our measurements we consider only the time for the methods to form checking sequences by using a given a PDS or an ADS.

Time requirements of the methods should not be affected noticeably by the type of the distinguishing sequence used. The experiments support this expectation. The percentage execution time differences between PDS and ADS cases are very small, and quickly approaches to 0 as the size of the FSMs increase. Figure 1 presents the results in terms of the average percentage decrease in the time it takes to construct

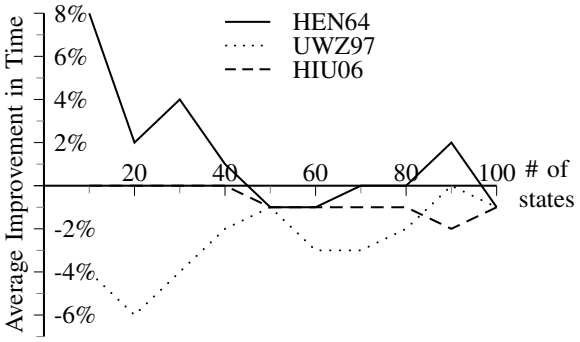

Fig. 1. Improvement in time for generating checking sequences

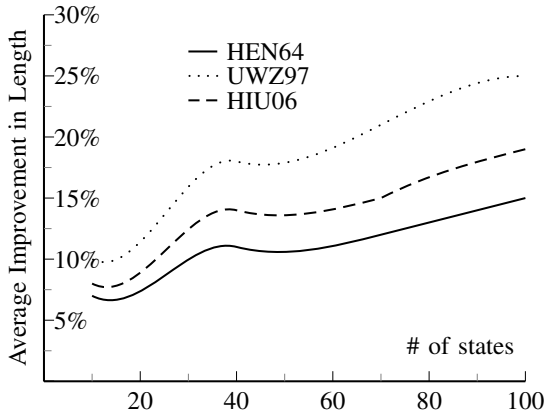

Fig. 2. Improvement in the length of checking sequences

checking sequences if ADSs are used instead of PDSs. As can be seen from the figure, using ADSs can occasionally increase the time requirement (negative values in the figure).

Another and possibly more important performance measure is the improvement on the length of the checking sequences. In other words, how much shorter will checking sequences get when ADSs are used instead of PDSs? The results of the experiments in this aspect are shown in Figure 2 and Figure 3.

As the size of the FSMs increase, an improvement of at least 10% is obtained. Depending on the method, the average improvement can be as high as 20-25%. When individual FSMs are considered, it is possible that for some FSMs the checking sequences generated by using ADSs are longer than the ones generated by using the PDSs. This happens around 20% of the cases when we consider the FSM set with 10 states only, but quickly decreases as we consider larger and larger FSMs and seen only in (less than) 1% of the cases for the FSMs with 100 states. However, on the average there is a consistent improvement.

We also explored the minimum and maximum improvements in checking sequences for each method and how these varied with the number of states. As the size of the FSMs increases, the minimum improvement on the length of the checking sequences also increases. For the methods HEN64 and HIU06, when we consider FSMs with 90 and 100 states,

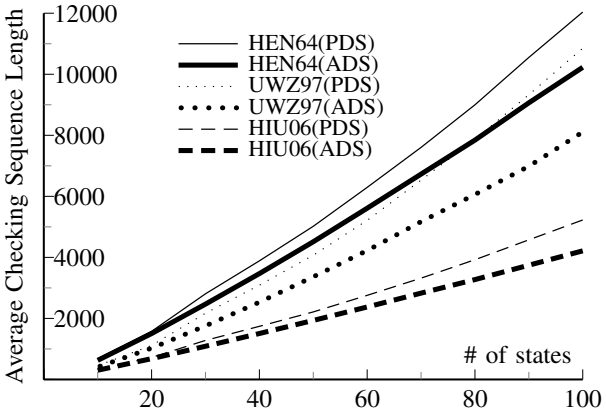

Fig. 3. Length of checking sequences

we see that minimum improvement is not negative. In other words among the 800 FSMs in these two groups there is not one single FSM for which using ADS causes the generation of a longer checking sequence than using PDS. The maximum improvement is more or less steady around 25%, 30% and 60% for the methods UWZ97, HIU06 and HEN64 respectively.

Note that although it is possible to get an exponential reduction in length by using ADS instead of PDS, this will be realized when an FSM with exponentially long PDS is used, which did not appear to happen in any one of our randomly generated test FSMs. Also note that we do not compare the performance of the methods considered here to the methods originally defined by using ADS (e.g. [3], [6]) as the main focus of this paper is to see the performance in those methods originally defined by using PDS when switched to ADS.

VI. Conclusion

A checking sequence generated from an FSM is guaranteed to lead to failures in a faulty implementation of this FSM under some commonly advocated assumptions. Many checking sequence generation methods are based on the use of a PDS that distinguishes the states of an FSM, despite the negative computational complexity results regarding the existence and length of PDSs.

This paper has investigated the use of ADSs for the construction of checking sequences. One of the benefits of using an ADS, rather than a PDS, is that there are FSMs for which there exists an ADS but no PDS and the converse is not the case. Further, in contrast to PDSs, there are polynomial time algorithms that decide whether an FSM has an ADS and, if it does, generate such an ADS.

We have shown that when a checking sequence is being produced, ADSs can be used in place of PDSs. In addition, recent checking sequence generation algorithms are based on a sufficient condition by Ural et al. [22] and we have proved that the corresponding result holds for ADSs. We have also shown how several checking sequence generation algorithms can be altered to use ADSs. Experimental results showed that the checking sequences constructed by using ADSs are almost consistently shorter than those based on PDSs.

Acknowledgment

This work was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada, the Ontario Centers of Excellence, the Marie Curie project MRTN-CT-2003-505121/TAROT, and Sabancı University.

References

[1] A. V. Aho, A. T. Dahbura, D. Lee, and M. U. Uyar, "An optimization technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman Tours," in *Protocol Specification, Testing, and Verification VIII*. Atlantic City: Elsevier (North-Holland), 1988, pp. 75–86.

[2] R. Boute, "Adaptive design methods for checking experiments," Digital Systems Laboratory, Stanford University, Tech. Rep. 30, July 1972.

[3] ——, "Distinguishing sets for optimal state identification in checking experiments," *Computers, IEEE Transactions on*, vol. C-23, no. 8, pp. 874–877, August 1974.

[4] J. Chen, R. M. Hierons, H. Ural, and H. Yenigün, "Eliminating redundant tests in a checking sequence," in *TestCom*, ser. Lecture Notes in Computer Science, F. Khendek and R. Dssouli, Eds., vol. 3502. Springer, 2005, pp. 146–158.

[5] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE Trans. Softw. Eng.*, vol. 4, no. 3, pp. 178–187, 1978.

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

[7] A. Dahbura, K. Sabnani, and M. Uyar, "Formal methods for generating protocol conformance test sequences," *Proceedings of the IEEE*, vol. 78, no. 8, pp. 1317–1326, 1990.

[8] S. Fujiwara, G. von Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test selection based on finite state models," *IEEE Trans. Softw. Eng.*, vol. 17, no. 6, pp. 591–603, 1991.

[9] A. Gibbons, *Algorithmic Graph Theory*. Cambridge University Press, 1985.

[10] A. Gill, *Introduction to the Theory of Finite-State Machines*. New-York: McGraw-Hill, 1962.

[11] G. Gonenc, "A method for the design of fault detection experiments," *IEEE Trans. Comput.*, vol. 19, no. 6, pp. 551–558, 1970.

[12] 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, November 1964, pp. 95–110.

[13] R. M. Hierons and H. Ural, "Reduced length checking sequences," *IEEE Trans. Comput.*, vol. 51, no. 9, pp. 1111–1117, 2002.

[14] ——, "Optimizing the length of checking sequences," *IEEE Trans. Comput.*, vol. 55, no. 5, pp. 618–629, 2006.

[15] R. M. Hierons, G.-V. Jourdan, H. Ural, and H. Yenigun, "Using adaptive distinguishing dequences in checking sequence constructions," in *SAC '08: Proceedings of the 2008 ACM Symposium on Applied computing*. New York, NY, USA: ACM, 2008, pp. 682–687.

[16] I. Kohavi and Z. Kohavi, "Variable-length distinguishing sequences and their application to the design of fault-detection experiments," *Computers, IEEE Transactions on*, vol. C-17, no. 8, pp. 792–795, Aug. 1968.

[17] D. Lee and M. Yannakakis, "Testing finite-state machines: State identification and verification," *IEEE Trans. Comput.*, vol. 43, no. 3, pp. 306–320, 1994.

[18] ——, "Principles and methods of testing finite state machines - A survey," in *Proceedings of the IEEE*, vol. 84, 1996, pp. 1090–1126. [Online]. Available: citeseer.ist.psu.edu/lee96principles.html

[19] K. S. Sabnani and A. Dahbura, "A protocol test generation procedure," *Comput. Netw. ISDN Syst.*, vol. 15, no. 4, pp. 285–297, 1988.

[20] M. N. Sokolovskii, "Diagnostic experiments with automata," *Kibernetika*, no. 6, pp. 44–49, 1971.

[21] K. T. Tekle, H. Ural, M. C. Yalcin, and H. Yenigün, "Generalizing redundancy elimination in checking sequences," in *ISCIS*, ser. Lecture Notes in Computer Science, P. Yolum, T. Güngör, F. Gürgen, and C. Özturan, Eds., vol. 3733. Springer, 2005, pp. 915–926.

[22] H. Ural, X. Wu, and F. Zhang, "On minimizing the lengths of checking sequences," *IEEE Trans. Comput.*, vol. 46, no. 1, pp. 93–99, 1997.