Optimizing the Length of Checking Sequences

R. M. Hierons *Senior Member, IEEE* and H. Ural

Abstract

A checking sequence, generated from a finite state machine, is a test sequence that is guaranteed to lead to a failure if the system under test is faulty and has no more states than the specification. The problem of generating a checking sequence for a finite state machine M is simplified if M has a distinguishing sequence: an input sequence $\bar{D}$ with the property that the output sequence produced by M in response to $\bar{D}$ is different for the different states of M. Previous work has shown that, where a distinguishing sequence is known, an efficient checking sequence can be produced from the elements of a set A of sequences that verify the distinguishing sequence used and the elements of a set Υ of subsequences that test the individual transitions by following each transition t by the distinguishing sequence that verifies the final state of t. In this previous work A is a predefined set and Υ is defined in terms of A. The checking sequence is produced by connecting the elements of Υ and A, to form a single sequence, using a predefined acyclic set E_c of transitions. An optimization algorithm is used in order to produce the shortest such checking sequence that can be generated on the basis of the given A and E_c. However, this previous work did not state how the sets A and E_c should be chosen. This paper investigates the problem of finding appropriate A and E_c to be used in checking sequence generation. We show how a set A may be chosen so that it minimizes the sum of the lengths of the sequences to be combined. Further, we show that the optimization step, in the checking sequence generation algorithm, may be adapted so that it generates the optimal E_c. Experiments are used to evaluate the proposed method.

Index Terms

Finite State Machine, Checking Sequence, Test Minimization, Distinguishing Sequence.

I. Introduction

Finite state machines (FSMs) can be used to model many types of systems including communication protocols [24] and control circuits [22]. A number of specification languages such as SDL, Estelle, X-machines and Statecharts are based on extensions of FSMs. FSM based test techniques can often be applied to systems specified using such languages [13], [17], [21], [23], [25], [27].

Given a formal model or specification of the required behaviour of the *system under test (SUT)* I it is normal to assume that I behaves like an unknown model that can be described using a particular formalism [14]. Given an FSM M, that models the required behaviour of SUT I, it is normal to assume that I behaves like an (unknown) FSM M_I with the same input and output alphabets as M. A common further assumption is that M_I has no more states than M.

Suppose M has n states. Let the set of deterministic FSMs with the same input and output alphabets as M and no more than n states be denoted $\Phi(M)$. A finite set of input sequences is a *checking experiment* for M if, between them, they distinguish M from every element of $\Phi(M)$ which is not equivalent to M. Given FSM M, there is some checking experiment [20]. A *checking sequence* is an input sequence that forms a checking experiment.

The problem of generating a checking sequence for an FSM M is simplified if M has a distinguishing sequence: an input sequence $\bar{D}$ with the property that the output sequence produced by M in response to $\bar{D}$ is different for the different states of M. There are two main alternative approaches for verifying a state: using a unique input/output sequence (UIO) or a characterization set. An input/output sequence $\bar{x}/\bar{y}$ is a unique input/output sequence for state s if M produces $\bar{y}$ in response to $\bar{x}$ when in state s and does not produce $\bar{y}$ in response to $\bar{x}$ from any other state of M. A set W of input sequences is a characterization

R.M. Hierons is with the School of Information Systems, Computing and Mathematics, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK

H. Ural is with the School of Information Technology and Engineering, University of Ottawa, Ottawa, Ontario, K1N 6N5, Canada

set if each pair of distinct states of M is distinguished by a sequence from W. Every minimal FSM has a characterization set but need not have a UIO for every state or a distinguishing sequence. While checking sequences can be produced on the basis of UIOs or a characterization set, restrictive assumptions are made in the literature. One of these assumptions is that there is a reliable reset operation, i.e. a reset operation that is known to have been correctly implemented. It is then possible to produce a polynomial size checking experiment [3], [28]. However not all SUTs have such a reset and in some cases the use of a reset can make testing more expensive and reduce the expected effectiveness of a test sequence or checking sequence (see, for example, [2], [10], [29]).

There has been much interest in the generation of short checking sequences from an FSM M when a distinguishing sequence is known [6], [7], [12], [26]. Naturally, the use of a short checking sequence makes testing more efficient and this has is particularly beneficial if a checking sequence is to be reused, possibly in regression testing or for different implementations of a standard. Recently Hierons and Ural [12] showed that an efficient checking sequence may be produced by combining the elements in a predefined set A of sequences called α'-sequences[1] with the transition tests in a set Υ (defined on the basis of A and M) using a predefined acyclic set E_c of transitions from M. An optimization algorithm is used to generate the checking sequence from A, Υ, and E_c. However, they did not indicate how A and E_c should be chosen and these choices can have a significant impact on the overall checking sequence length.

This paper considers the problem of generating the sets A and E_c with the aim of producing a minimum length checking sequence amongst those that can result from the application of the algorithm from [12]. Such a checking sequence is said to be *optimal*. We give an algorithm that produces a set A that minimizes the sum of the lengths of the subsequences to be combined in generating the checking sequence. We also show that the optimization phase of the checking sequence generation algorithm can be adapted so that it also generates the set E_c: it produces the *optimal* E_c for the given A. Thus, the overall checking sequence generation approach can be seen as having two stages:

1) minimizing the sum of the sizes of the subsequence to be combined; then
2) combining these subsequences optimally.

This paper is structured as follows. Section II introduces the basic concepts and notation used in this paper. Section III states results due to Ural et al. [26] and Hierons and Ural [12] that will be used in generating a checking sequence. It then gives a new checking sequence generation algorithm that takes as input the FSM M and the set A of α'–sequences. This is followed, in Section IV, by an algorithm for generating a set of α'-sequences that minimizes the sum of the lengths of the subsequences to be combined. In Section V a number of general results are proved while Section VI contains an experimental evaluation which demonstrates that the choice of A and E_c can have a significant impact on the length of the resultant checking sequence. Finally, in Section VII conclusions are drawn.

II. Preliminaries

A. Finite State Machines

A (deterministic and completely specified) FSM M is defined by a tuple $(S, s_1, X, Y, \delta, \lambda)$ in which S is a finite set of *states*, $s_1 \in S$ is the *initial state*, X is the finite *input alphabet*, Y is the finite *output alphabet*, δ is the *next state function* and λ is the *output function*. The functions δ and λ can be extended to take input sequences. See, for example, [16] for general information on FSMs.

Throughout this paper $M = (S, s_1, X, Y, \delta, \lambda)$ denotes a deterministic completely specified FSM that describes the required behaviour of the SUT I. The number of states of M is denoted n and the states of M are enumerated, giving $S = \{s_1, \ldots, s_n\}$. Only deterministic completely specified FSMs are considered in this paper. For information on testing from non–deterministic finite state machines see, for example, [8], [9], [11], [18], [19], [30]. For information on testing from incompletely specified FSMs see, for example, [18].

[1]These are defined in Section III.

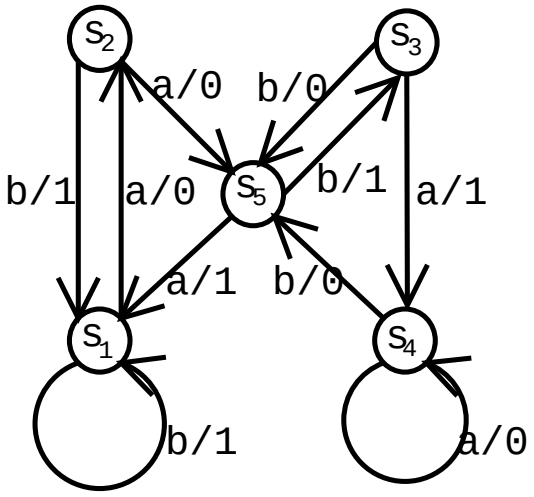

Fig. 1. The FSM M_0

An FSM, that is denoted M_0 throughout this paper, is described in Figure 1. Here, $S = \{s_1, \ldots, s_5\}$, $X = \{a, b\}$ and $Y = \{0, 1\}$. From the arc $s_1 \rightarrow s_2$ with label $a/0$ it is possible to deduce that if M_0 receives input a when in state s_1 it produces output 0 and moves to state s_2. Thus, in M_0, $\delta(s_1, a) = s_2$ and $\lambda(s_1, a) = 0$.

A transition τ is defined by a tuple $(s_i, s_j, x/y)$ in which s_i is the *starting state*, x is the input, $s_j = \delta(s_i, x)$ is the *ending state*, and $y = \lambda(s_i, x)$ is the output. Thus, for example, M_0 contains the transition $(s_1, s_2, a/0)$. Input r is a *reset operation* of M if, irrespective of the current state of M, it always takes M to its initial state. If M has a reset operation then it has *reset capacity*.

Two states s_i and s_j of M are *equivalent* if, for every input sequence $\bar{x}$, $\lambda(s_i, \bar{x}) = \lambda(s_j, \bar{x})$. If $\lambda(s_i, \bar{x}) \neq \lambda(s_j, \bar{x})$ then $\bar{x}$ *distinguishes* between s_i and s_j. Thus, for example, the input sequence a distinguishes states s_1 and s_3 of M_0. Two FSMs M_1 and M_2 are *equivalent* if and only if for every state of M_1 there is an equivalent state of M_2 and vice versa. An input sequence distinguishes between two FSMs if its application leads to different output sequences for these FSMs. An input sequence $\bar{x}$ is a *checking sequence* for M if and only if $\bar{x}$ distinguishes between M and all elements of $\Phi(M)$ that are not equivalent to M.

FSM M is *minimal* if no FSM with fewer states than M is equivalent to M. A sufficient condition for M to be minimal is that every state can be reached from the initial state of M and no two states of M are equivalent. There are algorithms that take an FSM and return an equivalent minimal FSM [20]. Thus only minimal FSMs are considered in this paper.

Given FSM M, a *distinguishing sequence* is an input sequence $\bar{D}$ whose output distinguishes all the states of M. More formally, for all $s, s' \in S$ if $s \neq s'$ then $\lambda(s, \bar{D}) \neq \lambda(s', \bar{D})$. Thus, for example, M_0 has distinguishing sequence aba. To see that aba is a distinguishing sequence for M_0 observe that the response to aba from the different states of M_0 are all different: from s_1 we get 010, from s_2 we get 011, from s_3 we get 101, from s_4 we get 001, and from s_5 we get 110. While not every FSM has a distinguishing sequence, there has been interest in the problem of generating a checking sequence in the presence of a distinguishing sequence [7], [12], [15], [26]. This paper considers the problem of generating an efficient checking sequence from a deterministic, minimal, and completely specified FSM M with a known distinguishing sequence $\bar{D}$.

B. Directed Graphs and Networks

A *directed graph (digraph)* G is defined by a tuple (V, E) in which V is a set of vertices and E is a set of directed edges between the vertices. Each edge may have a label. An edge e from vertex v_i to

vertex v_j with label l will be represented by (v_i, v_j, l). Edge e *leaves* v_i and *enters* v_j. For a vertex $v \in V$, $indegree_E(v)$ denotes the number of edges from E that enter v and $outdegree_E(v)$ denotes the number of edges from E that leave v.

Given an FSM, it is possible to produce a corresponding digraph in which each state is represented by a vertex and each transition is represented by an edge. Throughout this paper $G = (V, E)$ ($V = \{v_1, \ldots, v_n\}$) is a digraph, that represents M, in which state s_i is represented by vertex v_i. A transition from state s_i to state s_j with input x and output y is represented by edge $e = (v_i, v_j, x/y)$ from E. For example, $(v_2, v_5, a/0)$ is an edge of the digraph for M_0 that represents the transition $(s_2, s_5, a/0)$.

A sequence $\bar{P} = (n_1, n_2, x_1/y_1), \ldots, (n_{r-1}, n_r, x_{r-1}/y_{r-1})$ of pairwise adjacent edges from G forms a *walk* in which each *node* n_i represents a vertex from V and thus, ultimately, a state from S. Here $initial(\bar{P})$ denotes n_1, which is the *initial node* of $\bar{P}$, and $final(\bar{P})$ denotes n_r, which is the *final node* of $\bar{P}$. The sequence $\bar{T} = (x_1/y_1), \ldots, (x_{r-1}/y_{r-1})$ is the *label* of $\bar{P}$ and is denoted $label(\bar{P})$. $\bar{T}$ is said to be a *transfer sequence* from n_1 to n_r. The walk $\bar{P}$ can be represented by the tuple $(n_1, n_r, \bar{T})$ or by the tuple $(n_1, n_r, \bar{I}/\bar{O})$ in which $\bar{I} = x_1, \ldots, x_r$ is the *input portion* of $\bar{T}$ and $\bar{O} = y_1, \ldots, y_r$ is the *output portion* of $\bar{T}$. The *cost* of a sequence $\bar{\rho}$ is the number of elements in the sequence and is denoted $|\bar{\rho}|$.

A *tour* is a walk whose initial and final nodes are the same. Given a tour $\Gamma = e_1, \ldots, e_k$, $e_i = (n_i, n_{i+1}, l_i)$, $(1 \leq i < k)$ then $e_j, \ldots, e_k, e_1, \ldots, e_{j-1}$ is a walk formed by *starting* Γ with edge e_j. An *Euler Tour* is a tour that contains each edge exactly once. If the vertices represented by the nodes of walk $\bar{P}$ are distinct, $\bar{P}$ is said to be a *path*. A sequence of edges $e_1, \ldots, e_k$, $e_i = (n_i, n_{i+1}, l_i)$, $(1 \leq i < k)$ forms a *cycle* if $e_1, \ldots, e_{k-1}$ is a path and n_1 and n_{k+1} represent the same vertex. A set E' of edges from G is *acyclic* if no subset of E' forms a cycle.

A digraph is *strongly connected* if for any ordered pair of vertices (v_i, v_j) there is a walk from v_i to v_j. A digraph G is *weakly connected* if the underlying undirected graph is connected: for each ordered pair (v_i, v_j) of vertices there is a sequence $(n_1, n_2, l_1), \ldots, (n_k, n_{k+1}, l_k)$ in which each node n_r represents a vertex from V, n_1 represents v_i, n_{k+1} represents v_j, and for each (n_r, n_{r+1}, l_r) $(1 \leq r \leq k)$ at least one of (n_r, n_{r+1}, l_r) and (n_{r+1}, n_r, l_r) is in E. Naturally, every strongly connected digraph is weakly connected but the converse is not the case. An FSM is *strongly connected* if the digraph that represents it is strongly connected. Only strongly connected FSMs are considered in this paper.

A *network* is a digraph in which there are two special vertices, the *source* s and *sink* t, and each edge is given a *capacity* and a *cost*. A *flow* F for a network N is an assignment of non–negative integer values to each edge such that the flow through an edge (the value assigned to this edge) does not exceed the capacity of the edge and the flow is conserved: for each vertex, except s and t, the total flow entering the vertex is equal to the total flow leaving it. Given a flow F of a network N, the size of the flow, $|F|$, is the net flow leaving the source s of N. The *cost* of F is the sum, over the edges, of the flow through the edge multiplied by the cost of the edge. For more on digraphs and networks see, for example, [5].

C. Recognizing states and verifying edges

The algorithms of Ural et al. [26] and Hierons and Ural [12] use the notion of recognizing a node, corresponding to the state reached by a given input/output sequence, and verifying an edge of E. These notions, which are defined in terms of a given distinguishing sequence $\bar{D}$, are defined below. The key point is that, since the SUT I has no more states than M, if we observe the n possible responses of M to $\bar{D}$ when applied to I, then $\bar{D}$ must also be a distinguishing sequence for I. Once this has been demonstrated, we can use $\bar{D}$ to investigate the structure of I and thus to determine whether it is equivalent to M.

Consider a walk $\bar{P}$ and the nodes within it. Let $\bar{Q} = label(\bar{P})$.

Definition 1 1) *A node n_i of $\bar{P}$ is* d–recognized *in $\bar{Q}$ as state s of M if n_i is the initial node of a subpath of $\bar{P}$ whose label is input/output sequence $\bar{D}/\lambda(s, \bar{D})$.*

2) *Suppose that* $(n_q, n_i, \bar{T})$ *and* $(n_j, n_k, \bar{T})$ *are subpaths of* $\bar{P}$ *and* $\bar{D}/\lambda(s, \bar{D})$ *is a prefix to* $\bar{T}$ *(and thus* n_q *and* n_j *are d–recognized in* $\bar{Q}$ *as state* s *of* M*). Suppose also that node* n_k *is d–recognized in* $\bar{Q}$ *as state* s' *of* M*. Then* n_i *is* t–recognized *in* $\bar{Q}$ *as* s'*.*
3) *Suppose that* $(n_q, n_i, \bar{T})$ *and* $(n_j, n_k, \bar{T})$ *are subpaths of* $\bar{P}$ *such that* n_q *and* n_j *are either d-recognized or t–recognized in* $\bar{Q}$ *as state* s *of* M *and* n_k *is either d–recognized or t–recognized in* $\bar{Q}$ *as state* s' *of* M*. Then* n_i *is* t–recognized *in* $\bar{Q}$ *as* s'*.*
4) *If node* n_i *of* $\bar{P}$ *is either d–recognized or t–recognized in* $\bar{Q}$ *as state* s *then* n_i *is* recognized *in* $\bar{Q}$ *as state* s*.*
5) *Edge* $e = (v_a, v_b, x/y)$ *is* verified *in* $\bar{Q}$ *if there is a subpath* $(n_i, n_{i+1}, x_i/y_i)$ *of* $\bar{P}$ *such that* n_i *is recognized as* s_a *in* $\bar{Q}$*,* n_{i+1} *is recognized as* s_b *in* $\bar{Q}$*,* $x_i = x$ *and* $y_i = y$*.*

The first rule says that a node is d–recognized as a state s if it is followed by the input/output sequence $\bar{D}/\lambda(s, \bar{D})$. This is essentially saying that $\bar{D}$ defines a one–to–one correspondence between the states of the SUT and the states of M: this must be the case if the n different responses to $\bar{D}$ are observed in the SUT. The second and third rules say that if an input/output sequence is observed from two different nodes n and n' that are both recognized (d–recognized or t–recognized) as the same state then their final nodes should correspond to the same state of M.

The fifth rule is related to a transition test that is defined as follows: The *transition test* for a transition $\tau = (s_i, s_j, x/y)$ is $label(\tau)\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$ for some transfer sequence $\bar{T}_j$. The following result, that provides a sufficient condition for an input/output sequence to be a checking sequence, may now be stated.

Theorem 1 *(Theorem 1, [26]) Let* $\bar{P}$ *be a walk from* G *that starts at* v_1 *and* $\bar{Q} = label(\bar{P})$*. If every edge* $(v_i, v_j, x/y)$ *of* G *is verified in* $\bar{Q}$*, then* $\bar{Q}$ *is a checking sequence of* M*.*

In this paper checking sequence generation is based on Theorem 1.

III. Generating checking sequences

This section gives an algorithm for generating a checking sequence from M on the basis of a distinguishing sequence $\bar{D}$ for M. It starts by defining α'–sequences [12]. We then adapt the algorithm of Hierons and Ural [12]. The change introduced in this paper allows the set E_c of transitions used, to connect the required subsequences, to be chosen during optimization. The problem of choosing α'–sequences is considered in Section IV.

A. Defining α'–sequences

In previous work [12] α'–sequences were used as the basis for generating a checking sequence. First we define α'–sequences and we then explain their role in the construction of a checking sequence.

The α'–sequences are defined in the following way [12]. The first step is to choose $V_k \subseteq V$ ($1 \leq k \leq q$) whose union is V and to order the elements within each V_k, giving $V_k = \{v_1^k, \ldots, v_{m_k}^k\}$. Let s_i^k denote the state represented by v_i^k. For each v_i^k, produce a sequence $\bar{D}/\lambda(s_i^k, \bar{D})\bar{T}_i^k$; the result of applying $\bar{D}$ in state s_i^k followed by a transfer sequence $\bar{T}_i^k$ whose final state corresponds to v_{i+1}^k ($v_{m_k+1}^k$ can be any v_w^j, $1 \leq j \leq q, 1 \leq w \leq m_j$). For each V_k, form a walk $\bar{P}_k$ from s_1^k with label $\bar{\alpha}_k = \bar{D}/\lambda(s_1^k, \bar{D})\bar{T}_1^k\bar{D}/\lambda(s_2^k, \bar{D})\bar{T}_2^k \ldots \bar{D}/\lambda(s_{m_k}^k, \bar{D})\bar{T}_{m_k}^k\bar{D}/\lambda(s_w^j, \bar{D})\bar{T}_w^j$ ($1 \leq j \leq q$,$1 \leq w \leq m_j$). The set $\{\bar{\alpha}_1, \ldots, \bar{\alpha}_q\}$ is called an α'–set. Given an α'–set A, each sequence $\bar{\alpha}_i \in A$ is called an α'*–sequence from* A. Where the α'–set A is clear, its members are simply called α'*–sequences*.

The transfer sequence, that follows the execution of $\bar{D}$ from state s_i, is denoted $\bar{T}_i$.

The α'–sequences play the following roles in checking sequence generation.

1) They verify that the distinguishing sequence $\bar{D}$ used is also a distinguishing sequence for the SUT. This is achieved by applying $\bar{D}$ in every state of M: if the n different responses are observed then, since the SUT has at most n states, $\bar{D}$ must distinguish the states of the SUT.

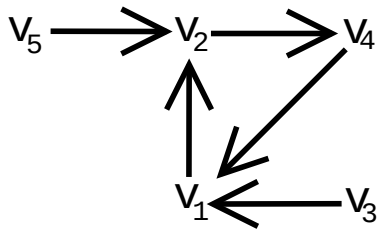

Fig. 2. The digraph $G_{\bar{D}}$

2) For each state s_i they d–recognize the final state (say s_j) reached by the walk from s_i with label $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$. This is achieved by the subsequence $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ followed by the input of $\bar{D}$. Note that if the subsequence $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ is seen elsewhere in the label of a walk, then the final node of this is t–recognized as the state s_j reached from s_i by a walk with label $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ since the initial node of $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ is d–recognized as s_i and the node reached by $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ has been d–recognized as s_j in an α'–sequence.
3) An α'–sequence $\bar{\alpha}_k$ from A starts with input sequence $\bar{D}$ and thus its initial node is recognized. Thus, an α'–sequence can be used to check the ending state of a transition [12].

The execution of $\bar{D}$, followed by a given transfer sequence, from each state, may be represented by a digraph $G_{\bar{D}}$ induced by the set of edges of the form (v_i, v_j) such that there is a walk from s_i to s_j with label $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$. The digraph $G_{\bar{D}}$ generated from M_0 with empty transfer sequences and distinguishing sequence aba is given in Figure 2. Recall that an α'–sequence must end in some $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ that is contained in the body of possibly another α'–sequence. Thus, an α'–set is represented by a set $\{\bar{p}_1, \ldots, \bar{p}_q\}$ of walks in $G_{\bar{D}}$ such that each $\bar{p}_i$ ends with an edge e with the property that there exists a walk $\bar{p}_j$ that contains e before its final edge.

From this it is possible to see that the following provide an α'–set for M_0:

- The sequence $\bar{\alpha}_1$ corresponding to the execution of $\bar{D}\bar{D}\bar{D}\bar{D}\bar{D}$ from s_5: this contains the edges of $G_{\bar{D}}$ that leave vertices v_5, v_2, v_4, v_1, and v_2. Note that here the walk ends with an edge (from v_2 to v_4) that was included earlier in the walk.
- The sequence $\bar{\alpha}_2$ corresponding to the execution of $\bar{D}\bar{D}$ from s_3 : this contains the edges of $G_{\bar{D}}$ that leave vertices v_3 and v_1. Here the walk ends with an edge (from v_1 to v_2) that was included in the walk in $G_{\bar{D}}$ representing $\bar{\alpha}_1$ and before the final edge of this walk.

We use these α'–sequences in checking sequence generation.

If a walk $\bar{P}$ contains every $\bar{P}_k$, ($1 \leq k \leq q$), and thus its label contains every α'–sequence from α'–set A, the final node of some $\bar{P}_k$ with label $\bar{\alpha}_k = \bar{D}/\lambda(s_1^k, \bar{D})\bar{T}_1^k\bar{D}/\lambda(s_2^k, \bar{D})\bar{T}_2^k \ldots \bar{D}/\lambda(s_{m_k}^k, \bar{D})\bar{T}_{m_k}^k\bar{D}/\lambda(s_w^j, \bar{D})\bar{T}_w^j$ is preceded by a subsequence, $\bar{D}/\lambda(s_w^j, \bar{D})\bar{T}_w^j$, contained within some $\bar{\alpha}_j \in A$ and thus followed by $\bar{D}$ in $\bar{\alpha}_j$. Thus, by the definition of recognition, if $\bar{P}$ contains every $\bar{P}_k$ ($1 \leq k \leq q$), then the final node of each $\bar{P}_k$ is recognized.

We use $E_{\alpha'}$ to denote the set of edges of the form $\bar{P}_k = (v_i, v_j, \bar{\alpha}_k)$, ($1 \leq k \leq q$).

B. Checking sequences: a sufficient condition

This section gives a sufficient condition, from [12], for a sequence to be a checking sequence. This result is a consequence of Theorem 1.

Theorem 2 *Let 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 ending state s_j, E_Υ contains one edge representing τ followed by either $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$ or some α'–sequence from A.*

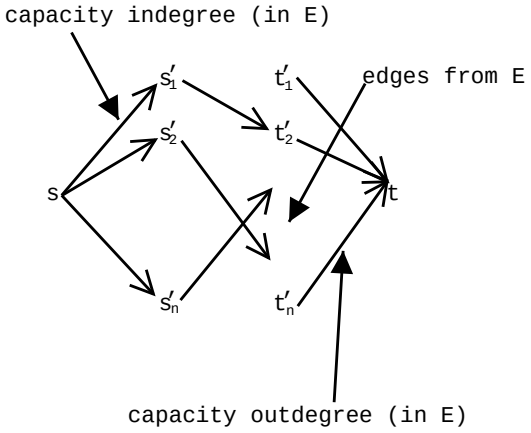

Fig. 3. The network N

2) *For every α'–sequence $\bar{\alpha}_k$ from A, E_Υ contains one edge that represents $\bar{\alpha}_k$ or a transition τ followed by $\bar{\alpha}_k$.*
3) *Every edge from E_Υ represents an α'–sequence or a transition τ, with ending state s_j, followed by either a sequence from A or $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$.*

Suppose Γ is a tour of G_Υ that contains every edge from E_Υ. Let e be an edge from E_Υ that represents the test for a transition τ whose ending state is s_1. Let Γ' denote Γ with e replaced by the corresponding sequence $e_1, \ldots, e_k$ of edges from G (and so e_1 represents τ) and let $\bar{P}$ denote the walk formed by starting Γ' with the edge e_2. Also let $G[E_C]$ denote the digraph induced by the set of edges in $\bar{P}$ that are not in E_Υ and suppose that $G[E_C]$ is acyclic. Then $\bar{Q} = label(\bar{P})\bar{D}/\lambda(s_1, \bar{D})$ is a checking sequence for M.

C. Producing the checking sequence

This subsection explains how, given an α'–set A, we can produce a checking sequence. The algorithm developed in this section utilizes the optimization algorithm, for the RCPP, used in [1]. By Theorem 2, it is sufficient to generate a checking sequence on the basis of a tour produced from the following:

1) For each transition τ, with ending state s_j, one instance of τ following by $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$ or an α'–sequence.
2) For every α'–sequence $\bar{\alpha}_i$, either $\bar{\alpha}_i$ or some transition τ followed by $\bar{\alpha}_i$.
3) Some acyclic set of connecting transitions.

If an α'–sequence $\bar{\alpha}_i$ is used to check the ending state of some transition τ we get overlap between a transition test and an α'–sequence. Thus, since we aim to produce an optimal checking sequence, each α'–sequence is used to check the ending state of some transition, except possibly one if the checking sequence starts with an α'–sequence.

The problem of producing a minimal length tour that satisfies these conditions can now be considered. The first step is to produce a network N from $G = (V, E)$, described below and outlined in Figure 3, and derive the minimum cost/maximum flow (min cost/max flow) F of N.

The network N has vertex set $\{s, t\} \cup \{s'_1, \ldots, s'_n\} \cup \{t'_1, \ldots, t'_n\}$, in which s is the source and t is the sink. The s'_i represent nodes after the execution of a transition being tested and before the execution of an α'–sequence or $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ and the t'_i represent nodes before the start of a transition test.

The edges are defined by the following rules:

1) For each i, there is an edge from s to s'_i with capacity $indegree_E(v_i)$ and cost 0. This is because there are $indegree_E(v_i)$ edges of G that end at v_i, each representing a transition that needs to be followed by an α'–sequence or $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$.

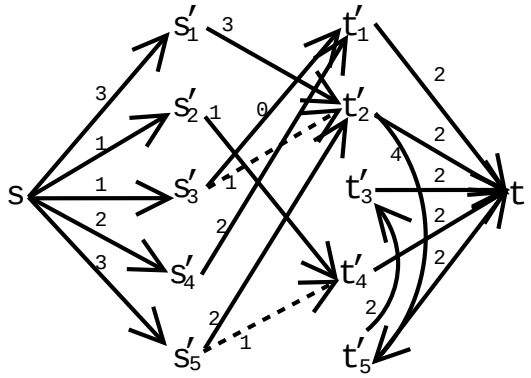

Fig. 4. The network and flow F_0 for M_0

2) For each i, there is an edge from t'_i to t with capacity $outdegree_E(v_i)$ and cost 0. This is because there are $outdegree_E(v_i)$ edges of G that leave v_i, each representing a transition that needs to be tested.
3) For each α'–sequence $\bar{\alpha}_k$ from v_i to v_j there is an edge from s'_i to t'_j with capacity 1 and cost $|\bar{\alpha}_k|$. This represents the execution of $\bar{\alpha}_k$ as part of a transition test.
4) For each state s_i, with s_j reached by the walk with label $\bar{D}/\lambda(s_i,\bar{D})\bar{T}_i$ from s_i, there is an edge from s'_i to t'_j with capacity $indegree_E(v_i) - outdegree_{E_{\alpha'}}(v_i)$ and cost $|\bar{D}/\lambda(s_i,\bar{D})\bar{T}_i|$. This represents the use of $\bar{D}/\lambda(s_i,\bar{D})\bar{T}_i$ as part of a transition test. The capacity is the number of transitions that will be followed by $\bar{D}/\lambda(s_i,\bar{D})\bar{T}_i$ but not an α'–sequence in the tour: each transition with ending state s_i must be followed by $\bar{D}/\lambda(s_i,\bar{D})\bar{T}_i$ but $outdegree_{E_{\alpha'}}(v_i)$ of these will be followed by an α'–sequence. The capacity of an edge leaving some s'_i and representing the execution of $\bar{D}/\lambda(s_i,\bar{D})\bar{T}_i$ is thus reduced by 1 if there is some α'–sequence leaving s_i, as this α'–sequence will be used to recognize the final state of one transition entering s_i. Each α'–sequence can always be executed in this manner as for every i, $1 \leq i \leq n$, $indegree_E(v_i) > 0$ (as M is strongly connected) and $outdegree_{E_{\alpha'}}(v_i) \leq 1$.
5) For each transition from s_i to s_j there is a corresponding edge from t'_i to t'_j with infinite capacity and cost 1. This represents an edge used to connect transition tests.

Consider transition $\tau = (s_i, s_j, x/y)$ and transition test $label(\tau)\bar{D}/\lambda(s_j,\bar{D})\bar{T}_j$ in which $\bar{D}/\lambda(s_j,\bar{D})\bar{T}_j$ labels a walk from s_j to s_k. The execution of τ as part of this transition test is represented by flow from t'_i to t and flow from s to s'_j. The execution of $\bar{D}/\lambda(s_j,\bar{D})\bar{T}_j$ as part of this transition test is represented by flow from s'_j to t'_k.

The min cost/max flow F is then found. This flow can be derived in low order polynomial time (see, for example, [1]). The network, and corresponding min cost/max flow, produced for M_0 is shown in Figure 4. Here, the only edges between the t'_i that are shown are those used in the flow. The actual flow through an edge is represented by an integer label and a dotted line represents an α'–sequence.

From F the digraph $G' = (V', E')$, in which $V' = \{a_1, \ldots, a_n\} \cup \{b_1, \ldots b_n\}$, is produced. The edge set E' is defined by the following:

1) For each transition τ from s_i to s_j in M there is a corresponding edge from b_i to a_j. This represents the execution of τ as part of a transition test.
2) Given an edge from s'_i to t'_j in N with flow f in F there are f corresponding edges from a_i to b_j. These represents the use of some $\bar{\alpha}_k$ or $\bar{D}/\lambda(s_i,\bar{D})\bar{T}_i$ as part of a transition test.
3) Given an edge from t'_i to t'_j in N with flow f in F, there are f corresponding edges from b_i to b_j. These represent the execution of transitions used to connect transition tests.

As flow is conserved at vertices, the digraph G' is *symmetric* (every vertex has an equal number of edges entering and leaving it). Thus, if G' is connected, it has an Euler Tour Γ (see, for example, [5])

and the corresponding checking sequence contains $cost(F) + |S||X| + |\bar{D}|$ transitions, where $cost(F)$ denotes the cost of the flow F. Conditions under which G' is guaranteed to be connected are considered in Section V. If G' is not connected then a set of tours can be produced. These tours can be connected by adding further transitions [12], [26].

We choose some edge e in Γ that represents a transition test for a transition τ that ends at s_1 and replace e by the corresponding sequence $e_1, \ldots, e_k$ of edges from G to form tour Γ'. We then start Γ' with e_2 to form a walk $\bar{P}$ with label $\bar{Q}$ and $\bar{Q}\bar{D}/\lambda(s_1, \bar{D})$ then forms a checking sequence. The (polynomial time) checking sequence generation algorithm can be summarised in the following way.

Algorithm 1

1) *Input M, distinguishing sequence $\bar{D}$ and α'–set A (and thus the transfer sequences $\bar{T}_1, \ldots, \bar{T}_n$).*
2) *Produce network N and min cost/max flow F for N.*
3) *Generate G' from F.*
4) *If G' is strongly connected, produce an Euler Tour Γ of G'; else produce a set of tours and connect these [12], [26] to form a tour Γ.*
5) *Choose some edge e in Γ that represents a transition test for a transition τ that ends at s_1 and replace e by the corresponding sequence $e_1, \ldots, e_k$ of edges from G to form tour Γ'.*
6) *Let $\bar{P}$ denote a walk produced by starting Γ' with e_2 and let $\bar{Q} = label(\bar{P})$.*
7) *Return the input/output sequence $\bar{Q}\bar{D}/\lambda(s_1, \bar{D})$.*

We now prove that the algorithm produces a checking sequence.

Lemma 3 *The set of edges between the t'_i, with non–zero flow in F, defines an acyclic subgraph of G.*

Proof: Proof by contradiction: suppose there is some set E^C of edges between the t'_i in N such that these edges define a cycle and they have non–zero flow in F. Produce an assignment F' of integers to edges of N by taking F and reducing the flow through each edge in E^C by 1. Since each edge in E^C has positive (integer) flow in F, no edge is given negative flow in F'. Further, since E^C defines a cycle, given a vertex t'_i, in forming F' we remove the same number of units of flow entering t'_i as we remove units of flow leaving t'_i. Thus, flow is conserved in F' and so F' is a flow. Finally, we have the same net flow leaving s in F and F' and the same net flow entering t in F and F'. Thus, F' is also a max flow but it is a max flow with lower cost than F'. This contradicts F being a min cost/max flow, as required. ■

Theorem 4 *The sequence produced by Algorithm 1 is a checking sequence.*

Proof: First observe that by Lemma 3 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 2. ■

The digraph G'_0 produced from flow F_0, for M_0, is shown in Figure 5. Here, $m > 1$ occurrences of an edge are represented by label m. Solid lines are used for edges that represent α'–sequences or instances of $\bar{D}$; individual transition (as part of transition tests or used to connect transition tests) are represented using dotted lines. An Euler tour of this leads to the following checking sequence in which the label of a transition from s_i to s_j is denoted by τ_{ij}.

$$\bar{D}/\lambda(s_1, \bar{D})\tau_{21}\bar{D}/\lambda(s_1, \bar{D})a/0b/1\tau_{34}\bar{D}/\lambda(s_4, \bar{D})\tau_{12}\bar{D}/\lambda(s_2, \bar{D})\tau_{45}\bar{D}/\lambda(s_5, \bar{D})\tau_{25}\bar{D}/\lambda(s_5, \bar{D})$$
$$a/0t_{51}\bar{D}/\lambda(s_1, \bar{D})a/0\tau_{53}\bar{\alpha}_2 a/0b/1\tau_{35}\bar{\alpha}_1\tau_{44}\bar{D}/\lambda(s_4, \bar{D})\tau_{11}\bar{D}/\lambda(s_1, \bar{D})$$

It is possible to check that all of the nodes are recognized and thus that all of the edges of G_0 are verified. This sequence thus defines a checking sequence.

Note that the set of connecting transitions is generated during optimization. In [12], [26] a set of connecting transitions is found prior to the optimization: this prior choice may be suboptimal.

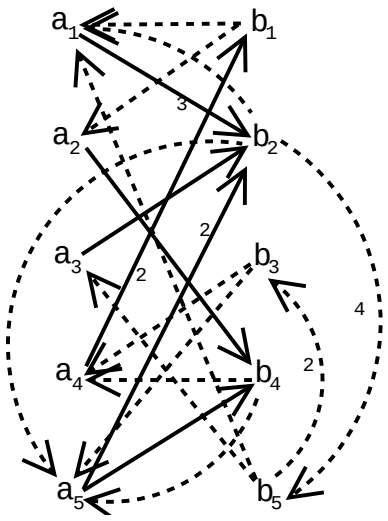

Fig. 5. The digraph G'_0 produced from F_0

IV. Finding an α'–set

The process of generating a checking sequence, in the presence of an α'–set, was described in Section III. This section discusses the problem of generating an α'–set A that minimizes the total length of the sequences in E_Υ, $length(E_\Upsilon) = \sum_{x \in E_\Upsilon} |x|$. For each state s_i, some α'–sequence will contain a corresponding subsequence $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ for some transfer sequence $\bar{T}_i$. In Section IV-A, an algorithm for generating an α'–set, once the $\bar{T}_i$ have been chosen, is described. Section IV-B contains a proof that if empty transfer sequences are used (i.e. $\bar{T}_i$ is the empty sequence for all $1 \leq i \leq n$) then any α'–set produced in this way minimizes $length(E_\Upsilon)$ and thus that empty transfer sequences should be used.

As noted earlier, the application of the $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ can be represented by a digraph $G_{\bar{D}} = (V, E_{\bar{D}})$ in which an edge from v_i represents a walk with label $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ from s_i. In $G_{\bar{D}}$, each vertex has one edge leaving it and $G_{\bar{D}}$ is composed of components in the form of circuits, possibly with trees attached. The digraph produced for M_0, using empty transfer sequences, is given in Figure 2.

A. Finding α'–sequences given the $\bar{T}_i$

Each α'–set $A = \{\bar{\alpha}_1, \ldots, \bar{\alpha}_q\}$ is defined by a set $\pi = \{\bar{P}_1, \ldots, \bar{P}_q\}$ of walks such that $label(\bar{P}_k) = \bar{\alpha}_k$, $(1 \leq k \leq q)$. To construct each $\bar{P}_k \in \pi$, first construct a set $P = \{\bar{\rho}_1, \ldots, \bar{\rho}_q\}$ of paths such that every edge of $G_{\bar{D}}$ is covered exactly once. For each $\bar{\rho}_k \in P$, we produce the sequence $label(\bar{\rho}_k)\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$, where s_i is the ending state of $\bar{\rho}_k$. This gives α'–set $A = \{label(\bar{\rho}_k)\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i | \bar{\rho}_k \in P, s_i$ is the ending state of $\bar{\rho}_k\}$. The problem of generating an α'–set may thus be reduced to that of producing such a set of paths given $G_{\bar{D}}$ (and thus from the transfer sequences $\bar{T}_1, \ldots, \bar{T}_n$).

The digraph $G_{\bar{D}}$ is composed of a number of (weakly connected) components $C_1, \ldots, C_r$, $1 \leq r \leq n$. The following algorithm produces paths that cover each component that is not in the form of a cycle. Cyclic components are then considered.

Algorithm 2

1) *Initially all edges of $G_{\bar{D}}$ are unmarked and $\pi = \emptyset$.*
2) *While there exists some v_i with an unmarked edge leaving it and no unmarked edge entering it, do*
   a) *Choose some v_i with an unmarked edge leaving it and no unmarked edge entering it.*
   b) *Find the longest path $\bar{\rho}$ in $G_{\bar{D}}$ that starts at v_i and does not use any marked edge. As $\bar{\rho}$ is a path it has no repeated edges.*
   c) *Follow $\bar{\rho}$ by the edge leaving its ending vertex in $G_{\bar{D}}$ to get the walk $\bar{P}$.*

d) *Add $\bar{P}$ to π and mark the edges of $\bar{\rho}$.*

endwhile

3) *Output π.*

The general problem of finding the longest path in a digraph is NP–complete (see, for example, [4]). However, since in $G_{\bar{D}}$ each vertex has only one edge leaving it, here the longest path problem can be solved in linear time.

In the example, there are two possible starting points: v_3 and v_5. If vertex v_5 is chosen initially the longest path is $v_5 \rightarrow v_2 \rightarrow v_4 \rightarrow v_1 \rightarrow v_2$ and thus the α'–sequence $\bar{\alpha}_1$, corresponding to $v_5 \rightarrow v_2 \rightarrow v_4 \rightarrow v_1 \rightarrow v_2 \rightarrow v_4$, is produced. The only remaining unmarked edge is $v_3 \rightarrow v_1$ and thus the α'–sequence $\bar{\alpha}_2$, corresponding to $v_3 \rightarrow v_1 \rightarrow v_2$, is then chosen.

At the end of Algorithm 2 there may still be unmarked edges in which case the set π output does not define an α'–set. However, we know that any vertex that has an unmarked edge leaving it also has an unmarked edge entering it. We thus get the following result.

Proposition 5 *When Algorithm 2 terminates the remaining unmarked edges of $G_{\bar{D}}$ form a set of cycles.*

Proof: Let $G_R = (V, E_R)$ denote the digraph defined by the vertex set of $G_{\bar{D}}$ and the set of edges of $G_{\bar{D}}$ that are unmarked at the end of Algorithm 2. By the termination criterion of Algorithm 2 we know that every vertex of G_R that has an edge that leaves it also has an edge that enters it.

First we prove that no vertex of G_R has an edge entering it but no edge leaving it. Proof by contradiction: suppose there is such a vertex v. Let $\bar{p}$ denote a maximal path from G_R that ends at v and let v' denote the starting vertex of $\bar{p}$. By the maximality of $\bar{p}$ and the fact that every vertex of G_R that has an edge that leaves it also has an edge that enters it, we know that v' has an edge from $\bar{p}$ entering it. Thus, $\bar{p}$ defines a subdigraph of G_R that is of the form of a cycle with a path leaving it. This contradicts each vertex having at most one edge leaving it as required.

Since no vertex of G_R has more than one edge leaving it, it is now sufficient to prove that no vertex of G_R has more than one edge entering it. Observe that the total number of edges entering vertices is equal to the total number of edges leaving vertices. The result thus follows from the facts that: no vertex has an edge entering it and no edge leaving it; no vertex has an edge leaving it and no edge entering it; and no vertex has more than one edge leaving it. ■

If the edges of a component C_i form a cycle then it is possible to start a walk whose label is an α'–sequence at any point within this. The walk produced has initial and final vertices corresponding to those of some edge in C_i. Suppose an edge from v_a to v_b is chosen and the corresponding α'–sequence is $\bar{\alpha}_k$. Then $\bar{\alpha}_k$ contains every $\bar{D}/\lambda(s_z, \bar{D})\bar{T}_z$ that corresponds to an edge from C_i. While $\bar{D}/\lambda(s_a, \bar{D})\bar{T}_a$ is included twice (once at the beginning, once at the end) the sequence $\bar{\alpha}_k$ is used to recognize s_a once in testing and thus, in E_Υ, replaces one execution of $\bar{D}/\lambda(s_a, \bar{D})\bar{T}_a$ from s_a. Thus the choice of edge from C_i does not affect $length(E_\Upsilon)$.

The final algorithm can now be given.

Algorithm 3

1) *Generate a set of walks π using Algorithm 2.*
2) *In $G_{\bar{D}}$ mark the edges contained in walks from π.*
3) *While there are unmarked edges in $G_{\bar{D}}$ do*
   a) *Choose a vertex v_i that has an unmarked edge leaving it.*
   b) *Find the longest walk $\bar{\rho}$ in $G_{\bar{D}}$ that starts at v_i and does not use any marked edge. This walk returns to v_i since only edges forming cyclic components remain unmarked after Algorithm 2.*
   c) *Follow $\bar{\rho}$ by the edge leaving its ending vertex to get $\bar{P}$.*
   d) *Add $\bar{P}$ to π and mark the edges of $\bar{\rho}$.*

   endwhile
4) *Output π.*

Theorem 6 *Algorithm 3 returns a set of walks that define an α'–set.*

Proof: By Proposition 5 we know that the set of unmarked edges after Algorithm 2 is of the form of a set of cyclic components. The result now follows from observing that each iteration of the loop creates a walk $\bar{P}$ that defines an α'–sequence and Algorithm 3 terminates when no edges are unmarked. ■

B. Finding the optimal $\bar{T}_i$

The previous section gave an algorithm that generates an α'–set given the set $\{\bar{T}_1, \ldots, \bar{T}_n\}$ of transfer sequences. This section contains results that prove that empty $\bar{T}_i$ lead to the minimal value of $length(E_\Upsilon)$ and that, given empty $\bar{T}_i$, any two α'–sets produce the same value of $length(E_\Upsilon)$. The first step is to place a lower bound on $length(E_\Upsilon)$.

Lemma 7 *Suppose M has distinguishing sequence $\bar{D}$, n states and input alphabet X. Then $length(E_\Upsilon) \geq n|X| + n|\bar{D}|(|X|+1)$.*

Proof: Suppose also that E_Υ has been formed using α'–set $A = \{\bar{\alpha}_1, \ldots, \bar{\alpha}_q\}$, where $\bar{\alpha}_i$ is $\bar{D}/\lambda(s^i_1, \bar{D})\bar{T}^i_1 \bar{D}/\lambda(s^i_2, \bar{D})\bar{T}^i_2 \ldots \bar{D}/\lambda(s^i_{m_i}, \bar{D})\bar{T}^i_{m_i}\bar{D}/\lambda(s^j_w, \bar{D})\bar{T}^j_w$ ($1 \leq j \leq q$,$1 \leq w \leq m_j$). Each $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ appears at least once within the body of some $\bar{\alpha}_j$. Repetition occurs through the final section of each $\bar{\alpha}_i$ appearing within the body of some $\bar{\alpha}_i$. Thus

$$\sum_{z=1}^{q} |\bar{\alpha}_z| \geq n|\bar{D}| + q|\bar{D}|.$$

The transitions may be enumerated to give $\{\tau_1, \ldots \tau_{n|X|}\}$ such that, in E_Υ, $\tau_1, \ldots, \tau_q$ are followed by $\bar{\alpha}_1, \ldots, \bar{\alpha}_q$ respectively. Given transition τ_z let $\sigma(z)$ satisfy the property that the ending state of τ_z is $s_{\sigma(z)}$. Therefore $E_\Upsilon = \{\tau_1\bar{\alpha}_1, \ldots, \tau_q\bar{\alpha}_q\} \cup \bigcup_{z=q+1}^{n|X|}\{\tau_z\bar{D}/\lambda(s_{\sigma(z)}, \bar{D})\bar{T}_{\sigma(z)}\}$. Thus

$$\sum_{\bar{x}\in E_\Upsilon} |\bar{x}| = \sum_{z=1}^{q} |\tau_z\bar{\alpha}_z| + \sum_{z=q+1}^{n|X|} |\tau_z\bar{D}/\lambda(s_{\sigma(z)}, \bar{D})\bar{T}_{\sigma(z)}|$$

$$= q + \sum_{z=1}^{q} |\bar{\alpha}_z| + (n|X| - q)(|\bar{D}| + 1) + \sum_{z=q+1}^{n|X|} |\bar{T}_{\sigma(z)}|$$

$$\geq q + n|\bar{D}| + q|\bar{D}| + (n|X| - q)(|\bar{D}| + 1)$$

$$= n|X| + n|\bar{D}| + n|X||\bar{D}| = n|X| + n|\bar{D}|(1 + |X|).$$

The result thus follows. ■

It is now sufficient to prove that any α'–set, produced by Algorithm 3, with empty $\bar{T}_i$ achieves this lower bound and thus is optimal.

Lemma 8 *Suppose M has distinguishing sequence $\bar{D}$, n states and input alphabet X. Suppose also that E_Υ contains the sequences produced using an α'–set A generated by Algorithm 3 in which, for all $1 \leq i \leq n$, $|\bar{T}_i| = 0$. Then $length(E_\Upsilon) = n|X| + n|\bar{D}|(|X|+1)$.*

Proof: Suppose $A = \{\bar{\alpha}_1, \ldots, \bar{\alpha}_q\}$. As, for all $1 \leq j \leq n$, $\bar{T}_j = \epsilon$, $\bar{\alpha}_i$ has input portion $\bar{D}^{k_i}\bar{D}$ for some k_i, $\sum_{i=1}^{r} k_i = n$. Thus

$$\sum_{z=1}^{q} |\bar{\alpha}_z| = (n+q)|\bar{D}|.$$

The transitions may be enumerated so that $E_\Upsilon = \{\tau_1\bar{\alpha}_1, \ldots, \tau_q\bar{\alpha}_q\} \cup \bigcup_{z=q+1}^{n|X|}\{\tau_z\bar{D}/\lambda(s_{\sigma(z)}, \bar{D})\}$. Thus

$$\sum_{\bar{x}\in E_\Upsilon} |\bar{x}| = \sum_{z=1}^{q} |\tau_z\bar{\alpha}_z| + \sum_{z=q+1}^{n|X|} |\tau_z\bar{D}/\lambda(s_{\sigma(z)}, \bar{D})|$$

$$= q + \sum_{z=1}^{q} |\bar{\alpha}_z| + (n|X| - q) + \sum_{z=q+1}^{n|X|} |\bar{D}|$$

$$= n|X| + \sum_{z=1}^{q} |\bar{\alpha}_z| + \sum_{z=q+1}^{n|X|} |\bar{D}|$$

$$= n|X| + (n+q)|\bar{D}| + (n|X| - q)|\bar{D}|$$

$$= n|X| + |\bar{D}|(n + q + n|X| - q)$$

$$= n|X| + n|\bar{D}|(1 + |X|)$$

The result thus follows. ■

Theorem 9 *Suppose that E_Υ contains the subsequences generated using α'–set A produced by Algorithm 3 in which, for all $1 \leq i \leq n$, $|\bar{T}_i| = 0$. Then this α'–set minimizes the value of $length(E_\Upsilon)$.*

Proof: This follows directly from Lemmas 7 and 8. ■

V. General properties of the algorithms

The proposed algorithm produces a symmetric digraph G' and if G' is strongly connected, an Euler Tour of G' is used to define a minimum length checking sequence, for the given A. This section gives two sufficient conditions for G' to be strongly connected. These conditions are equivalent to those given in [1] for an algorithm that connects a set of subsequences but need not generate a checking sequence.

Lemma 10 *If M has reset capacity then G' is strongly connected.*

Proof: As M has reset capacity, every b_i is connected to a_1. Thus the set of b_i is weakly connected. As M is strongly connected, every a_i is reached by some edge from some b_j. Thus, as the set of b_i is weakly connected, G' is weakly connected. It is known, however, that a weakly connected symmetric digraph is strongly connected (see, for example, [5]). Thus G' is strongly connected, as required. ■

Lemma 11 *If M has a loop (a transition whose initial and final states are the same) for every state then G' is strongly connected.*

Proof: As M has a loop for every state, each b_i is connected to the corresponding a_i. As it is sufficient to prove that G' is weakly connected, and each b_i is connected to some a_j, it is sufficient to prove that for any a_i there an undirected walk from a_1 to a_i. A walk $\bar{p}$ from G can be simulated by, for each edge e from v_i to v_j in $\bar{p}$, replacing e by a pair of edges (b_i, a_i) (b_i, a_j) in G'. Thus, as G is strongly connected, there is an undirected walk from a_1 to a_i for all $1 \leq i \leq n$. Thus G' is weakly connected and, as G' is symmetric, G' is strongly connected. ■

The proposed checking sequence generation algorithm has the same time complexity as those given in [26] and [12] and we now explore this complexity. For an FSM with n states Algorithms 2 and 3 both take time of $O(n)$. Thus the complexity of the algorithm is dominated by the time taken to find the min cost/max flow which is of $O(ev \log v)$ for a digraph with v vertices and e edges [1]. Thus, since the digraph representing M has n vertices and $n|X|$ edges, the worst case time complexity is $O(n^2|X| \log n)$.

VI. Experimental evaluation

This section describes an experimental evaluation that investigated the effect of using non–empty transfer sequences ($\bar{T}_i$) in the construction of the α'–sequences. There were two motivations for this study. First, while the proposed use of empty transfer sequences guarantees that the sum of the lengths of the subsequences to be combined is minimized, there is no guarantee that this leads to the shortest checking sequence. Second, while we might expect the use of empty transfer sequences to normally be desirable, experimental evaluation can provide some indication as to how significant an impact this has on the length of the resultant checking sequence.

We used a set of randomly generated FSMs with distinguishing sequences. We produced these FSMs in the following way. For a given integer n, for each state s_i ($1 \leq i \leq n$) and input x we randomly chose the next state s_j and output y. This led to an FSM with n states but this FSM might not have the desired properties. The FSM was rejected if it was not minimal, was not strongly connected, or we failed to find a distinguishing sequence.

For each FSM M we applied the following experiments:

1) We used Algorithm 3 to produce an α'–set with empty transfer sequences as proposed in Section IV. We then generated a checking sequence using Algorithm 1.
2) We applied the following procedure 1000 times: For each state s_i of M randomly choose some state s^i from M to be reached by the transfer sequence from $\delta(s_i, \bar{D})$. For each s_i, we generated a transfer sequence $\bar{T}_i$ that labelled a shortest walk from $\delta(s_i, \bar{D})$ to s^i and used Algorithm 3 to produce the corresponding α'–set A. We then applied Algorithm 1, with A and the transfer sequences, to produce a checking sequence. This was done for a randomly generated selection since for an FSM with n states there are n^n ways of choosing the transfer sequences.

For each FSM M we recorded the checking sequence length produced using the proposed algorithm and thus empty transfer sequences. The checking sequence algorithm is deterministic once the transfer sequences have been chosen and thus we produced only one such checking sequence for each FSM.

For the 1000 other experiments with a given FSM M we recorded the mean checking sequence length, the maximum checking sequence length, and the minimum checking sequence length. We used five FSMs with 5 states, five FSMs with 10 states, five FSMs with 15 states, and five FSMs with 20 states. The FSMs with 5 states had input and output alphabets of size 3, the FSMs with 10 and 15 states had input and output alphabets of size 4, and the FSMs with 20 states had input and output alphabets of size 5. The results are given in Table I.

In all cases the checking sequence with empty transfer sequences was the smallest found. It is interesting to look at how much of a saving is provided by using empty transfer sequences and to consider both the saving relative to the mean checking sequence length found and the maximum checking sequence length found: the former gives an indication of the *expected* saving while the latter gives an indication of the *maximum* saving that can be expected. Table II summarizes this information. For each FSM size it gives the following information:

1) The first column contains the number of states of the FSMs.
2) The second column contains the mean checking sequence length when we have empty transfer sequences. This is averaged across the five FSMs with the given number of states.
3) The third column contains the mean, over the five FSMs, of the mean checking sequence length when we do not use empty transfer sequences. In the fourth column we give the percentage saving: the difference between the values in the second and third columns divided by the value in the third column (the larger of the two values). This estimates the expected saving from using empty transfer sequences.
4) The fifth column gives the mean, over the five FSMs, of the length of the longest checking sequence found. The sixth column contains the percentage saving: the difference between the values in the fifth and second columns divided by the value in the fifth column (again, the larger of the two values). This estimates the maximum saving from using empty transfer sequences.

TABLE I

EXPERIMENTAL RESULTS

FSM	Number of states	Empty transfer	Maximum	Minimum	Mean
5_0	5	68	134	68	97
5_1	5	94	134	94	118
5_2	5	63	107	63	88
5_3	5	60	111	60	90
5_4	5	71	112	71	91
10_0	10	209	347	251	299
10_1	10	229	383	241	324
10_2	10	259	473	282	340
10_3	10	171	301	196	248
10_4	10	226	375	254	313
15_0	15	327	593	400	494
15_1	15	352	603	394	504
15_2	15	337	563	394	479
15_3	15	351	583	400	499
15_4	15	352	601	404	496
20_0	20	625	990	639	854
20_1	20	530	859	695	769
20_2	20	561	935	670	789
20_3	20	560	923	669	817
20_4	20	568	940	668	813

TABLE II

SUMMARY: MEAN SAVINGS

Number of states	mean empty transfer	mean	saving	mean maximum	saving
5	71.2	96.8	26.45%	119.6	40.47%
10	218.8	304.8	28.22%	375.8	41.78%
15	343.8	494.4	30.46%	588.6	41.59%
20	568.8	808.4	29.64%	929.4	38.80%

In the experiments, for each FSM size, the use of empty transfer sequences gave a saving of over 25% when compared to the mean checking sequence length and a maximum saving of in the order of 40%.

VII. CONCLUSIONS

When testing from a finite state machine (FSM) M it is often desirable to use a checking sequence: a test sequence that is guaranteed to lead to failures if the system under test (SUT) is faulty and has no more states than M. There has thus been much interest in the automated generation of efficient checking sequences [6], [7], [12], [26].

The method recently given in [12], to generate a checking sequence, produces a checking sequence by connecting a set of subsequences. However, it relies on two elements, the α'–set A and a set E_c of connecting transitions, to have already been defined. The choice of A and E_c can have a significant impact on the length of the resultant checking sequence. This paper has focussed on the problem of choosing A and E_c. The overall checking sequence generation approach, used in this paper, can be seen as having two stages:

1) minimize the sum of the lengths of the subsequences to be combined; then
2) combine these sequences optimally.

This paper has given an algorithm that finds an α'–set A that minimizes the sum of the lengths of the subsequences to be combined in checking sequence generation. The checking sequence generation algorithm given in this paper produces the set E_c of connecting transitions *during* the optimization phase of test generation. The algorithm thus produces the optimal E_c for the given A.

The choice of E_c is guaranteed to be optimal. Thus, experimental evaluation was used to investigate the other variable: the choice of transfer sequences (which define the set A). The experiments were over

twenty randomly generated FSMs with between 5 and 20 states. In all experiments, the checking sequence generated using the proposed approach was the shortest found. In the experiments, for each FSM size, the proposed approach gave a mean saving of over 25% and a maximum saving of in the order 40%.

For ease of presentation, we formulated the problem as that of forming a tour from which a checking sequence is extracted as given in Theorem 2. A succinct formulation of the minimum length checking sequence construction follows directly from our work: after forming G', find a rural Chinese postman path over the subset of edges E_γ starting with the application of $\bar{D}$ (or some α'–sequence) at s_1.

ACKNOWLEDGEMENTS

This work was supported in part by Leverhulme Trust grant number F/00275/D, Testing State Based Systems, Natural Sciences and Engineering Research Council (NSERC) of Canada grant number RGPIN 976, and Engineering and Physical Sciences Research Council grant number GR/R43150, Formal Methods and Testing (FORTEST). We would like to thank Karnig Derderian and Tuong Nguyen for their assistance in the experiments.

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*, pages 75–86, Atlantic City, 1988. Elsevier (North–Holland).

[2] B. Broekman and E. Notenboom. *Testing Embedded Software*. Addison–Wesley, London, 2003.

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

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Company, New York, 1979.

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

[6] G. Gonenc. A method for the design of fault detection experiments. *IEEE Transactions on Computers*, 19:551–558, 1970.

[7] F. C. Hennie. Fault–detecting experiments for sequential circuits. In *Proceedings of Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 95–110, Princeton, New Jersey, November 1964.

[8] R. M. Hierons. Adaptive testing of a deterministic implementation against a nondetermistic finite state machine. *The Computer Journal*, 41(5):349–355, 1998.

[9] R. M. Hierons. Generating candidates when testing a deterministic implementation against a non–deterministic finite state machine. *The Computer Journal*, 46(3):307–318, 2003.

[10] R. M. Hierons. Minimizing the number of resets when testing from a finite state machine. *Information Processing Letters*, 90(6):287–292, 2004.

[11] R. M. Hierons. Testing from a non–deterministic finite state machine using adaptive state counting. *IEEE Transactions on Computers*, 53(10):1330–1342, 2004.

[12] R. M. Hierons and H. Ural. Reduced length checking sequences. *IEEE Transactions on Computers*, 51(9):1111–1117, 2002.

[13] M. Holcombe and F. Ipate. *Correct Systems: Building a Business Process Solution*. Springer–Verlag, 1998.

[14] ITU-T. *Recommendation Z.500 Framework on formal methods in conformance testing*. International Telecommunications Union, Geneva, Switzerland, 1997.

[15] I. Kohavi and Z. Kohavi. Variable-length distinguishing sequences and their application to the design of fault–detection experiments. *IEEE Transactions on Computers*, pages 792–795, August 1968.

[16] Z. Kohavi. *Switching and Finite State Automata Theory*. McGraw–Hill, New York, 1978.

[17] G. Luo, A. Das, and G. v. Bochmann. Generating tests for control portion of SDL specifications. In *Protocol Test Systems VI*, pages 51–66. Elsevier (North-Holland), 1994.

[18] G. Luo, A. Petrenko, and G. v. Bochmann. Selecting test sequences for partially–specified nondeterministic finite state machines. In *The 7th IFIP Workshop on Protocol Test Systems*, pages 95–110, Tokyo, Japan, November 8–10 1994. Chapman and Hall.

[19] G. L. Luo, G. v. Bochmann, and A. Petrenko. Test selection based on communicating nondeterministic finite–state machines using a generalized Wp–method. *IEEE Transactions on Software Engineering*, 20(2):149–161, 1994.

[20] E. P. Moore. Gedanken-experiments. In C. Shannon and J. McCarthy, editors, *Automata Studies*. Princeton University Press, 1956.

[21] A. Petrenko, S. Boroday, and R. Groz. Confirming configurations in EFSM testing. *IEEE Transactions on Software Engineering*, 30(1):29–42, 2004.

[22] I. Pomeranz and S. M. Reddy. Test generation for multiple state–table faults in finite–state machines. *IEEE Transactions on Computers*, 46(7):783–794, 1997.

[23] Q. M. Tan, A. Petrenko, and G. v. Bochmann. Modeling basic LOTOS by FSMs for conformance testing. In *IFIP Protocol Specification, Testing, and Verification XV*, pages 137–152, 1995.

[24] A. S. Tanenbaum. *Computer Networks*. Prentice Hall International Editions, Prentice Hall, 3rd edition, 1996.

[25] H. Ural, K. Saleh, and A. Williams. Test generation based on control and data dependencies within system specifications in SDL. *Computer Communications*, 23:609–627, 2000.

[26] H. Ural, X. Wu, and F. Zhang. On minimizing the lengths of checking sequences. *IEEE Transactions on Computers*, 46(1):93–99, 1997.

[27] G. v. Bochmann, A. Petrenko, O. Bellal, and S. Maguiraga. Automating the process of test derivation from SDL specifications. In *SDL Forum'97*, Paris, France, 1997.

[28] S. T. Vuong, W. W. L. Chan, and M. R. Ito. The UIOv–method for protocol test sequence generation. In *The 2nd International Workshop on Protocol Test Systems*, Berlin, 1989.

[29] M. Yao, A. Petrenko, and G. v. Bochmann. Conformance testing of protocol machines without reset. In *Protocol Specification, Testing and Verification, XIII (C–16)*, pages 241–256. Elsevier (North–Holland), 1993.

[30] N. V. Yevtushenko, A. V. Lebedev, and A. F. Petrenko. On checking experiments with nondeterministic automata. *Automatic Control and Computer Sciences*, 6:81–85, 1991.