Distinguishing Sequence Based Checking Sequence Generation Implementation and Improvements

by Mehmet Cihan Yalçın

Submitted to the Graduate School of Sabancı University
in partial fulfillment of the requirements for the degree of
Master of Science

Sabanci University

August, 2006

Distinguishing Sequence Based
Checking Sequence Generation
Implementation and Improvements

Mehmet Cihan Yalçın
EECS, Master's Thesis, 2006
Thesis Supervisor: Hüsnü Yenigün

Keywords: Formal Testing Methods, Checking Sequences, Distinguishing Sequences

Abstract

With advances in computer technology and software engineering, systems are constantly becoming larger and more complex. Straightforward testing methods are insufficient to cope with the complexity and maintaining quality of service demands the use of more structured testing methods.

Checking sequences are testing mechanisms based on finite state behavior models that can offer guarantees about a system under test, under certain assumptions. However, their complexities are high, and to make their implementation feasible methods of their construction need to be refined.

We have studied several methods of checking sequence construction in the presence of distinguishing sequences, developed fully formed algorithms from loose specifications, then implemented and compared their performances. We have also proposed several improvements that will allow generation of shorter checking sequences. We are confident that these developments will be instrumental in making the use of checking sequences feasible in a larger scope.

Ayırıcı Dizilere Dayalı Kontrol Dizisi Üretimi Uygulama ve Geliştirmeleri

Mehmet Cihan Yalçın
EECS, Master Tezi, 2006
Thesis Supervisor: Hüsnü Yenigün

Keywords: Formal Test Metodları, Kontrol Dizileri, Ayırıcı Diziler

Özet

Bilgisayar teknolojisi ve yazılım mühendisliğindeki ilerlemelerle, sistemler gitgide daha büyüyor ve karmaşıklaşıyor. Sıradan test metodları bu karmaşıklıkla başetmekte yetersiz kalıyor ve hizmet kalitesini korumak için daha düzenli test metodları gerekiyor.

Kontrol dizileri, sonlu durumlu davranış modellerine dayanan ve belli koşullar altında test edilen sistem hakkında garantiler verebilen yapılardır. Ancak, karmaşıklıkları yüksektir ve kullanılmalarını uygulanabilir kılmak için üretim metodları geliştirilmelidir.

Biz, ayırıcı serilerin varlığında kontrol serisi üretiminde kullanılabilecek kimi metodları inceledik, esnek spesifikasyonlardan net algoritmalar üreterek bunları uyguladık ve metodların performanslarını karşılaştırdık. Ek olarak, daha kısa kontrol serilerinin üretimine olanak sağlayacak, çeşitli gelişmeler öneriyoruz. Bu gelişmelerin, kontrol serilerinin kullanılabileceği çerveçenin gelişmesini sağlamada yararlı olacağına inanıyoruz.

Acknowledgements

I wish to express my gratitude to,

Hüsnü Yenigün, for his infinite patience and support,

Hasan Ural, for his confidence and patronage,

TÜBİTAK, for providing the necessary motivation,

last, but not the least, to my family, for being there when I needed them to be.

Contents

1 Introduction **1**

2 Preliminaries **4**

2.1 Finite State Machines 4

2.1.1 Mealy Machine Definition 4

2.1.2 Determinism and Complete Specification 4

2.1.3 Transition Definition 5

2.1.4 Extension of Function Definitions to Sequences 5

2.1.5 Reachability, State Equivalence and Minimality 5

2.2 Checking Sequences 6

2.2.1 Unique Input-Output Sequences 6

2.2.2 Distinguishing Sequences 7

2.2.3 Distinguishing Machines and Checking Sequences . . . 7

2.3 Directed Graphs 8

2.3.1 Edges, Indegree and Outdegree 8

2.3.2 Graph Representation of FSMs 8

2.3.3 Walks and Tours on Graphs 9

2.3.4 Reachability and Strong Connectivity 9

2.3.5 Euler Tours 10

2.3.6 Rural Chinese Postman Tours 10

2.4 Recognizing States and Verifying Transitions 14

2.4.1 D-recognition 15

2.4.2 T-recognition 15

2.4.3 Edge Verification 16

2.4.4 Checking Sequences Implementing Edge Verification . . 18
2.4.5 α-Sequences 18
2.4.6 α'-Sequences 19
2.4.7 Special Sequences on M_0 22

3 Existing Approaches **25**
3.1 Using RCPP for Integrating Test Segments 26
3.1.1 Test Segments 26
3.1.2 Construction of the Augmented Graph 26
3.2 Optimizations on Test Segments 29
3.2.1 Test Segments 29
3.2.2 Construction of the Augmented Graph 29
3.3 Overlap Elimination 32
3.3.1 D-overlapping 32
3.3.2 Test Segments 32
3.3.3 Construction of the Augmented Graph 33
3.4 Initial Idea for Redundant Transition Test Elimination 36
3.4.1 Identifying Redundant Transition Tests 37
3.4.2 Checking Sequence Construction 39

4 Proposed Improvements **43**
4.1 Expanding the Dependency Graph 43
4.2 R-recognition 47
4.3 Generalized Redundant Transition Test Elimination 48
4.3.1 Identifying Redundant Transition Tests 50
4.3.2 Checking Sequence Construction 53

4.4 Introducing UIOs 56
4.4.1 Test Segments 57
4.4.2 Choosing UIOs to Use 57
4.4.3 Checking Sequence Construction 58
4.4.4 Further Improvements 61

5 Experiments **63**
5.1 Experiment Setup 63
5.2 Experiment Results 66

6 Conclusion **68**

List of Figures

1 The FSM M_0 9
2 Conformance Testing Problem 14
3 D-recognition, T-recognition and Edge Verification on M_0 . . . 17
4 Graph of T-sequences that are to be used in construction of α-sequences 20
5 Construction of α-sequences for FSM M_0 21
6 Construction of α'-sequences for FSM M_0 23
7 Augmented graph for M_0 constructed according to Ural, Wu and Zhang '97 28
8 $G' = (V', E')$ for M_0, constructed according to Ural and Hierons '06 31

9 Augmented graph for M_0 constructed according to Ural and Zhang 35
10 Full Dependency Graph of M_0 for Basic Redundant Transition Test Elimination 38
11 $G' = (V', E')$ for M_0, constructed according to Chen, Hierons, Ural and Yenigun '05 41
12 The FSM M_1 44
13 Dependency graph for M_1 44
14 Expanded dependency graph for M_1 45
15 Average Number of Exempted Transition Tests 46
16 Transition verification over D-sequences through use of R-recognition 49
17 Full Dependency Graph of M_0 for Generalized Redundant Transition Test Elimination 52
18 Dependency Graph for Generalized Redundant Transition Test Elimination 54
19 $G' = (V', E')$ for M_0, constructed according to Tekle, Ural, Yalcin, Yenigun '05 55
20 $G' = (V', E')$ for M_0 implementing UIO sequences. 61
21 $G' = (V', E')$ for M_0. Implementing both redundant transition test elimination and UIO sequences. 62
22 Average Checking Sequence Lengths 65

List of Tables

1 Checking Sequence Length for Each FSM Used in the Experiment 64

1 Introduction

With advances in computer technology and software engineering, systems are constantly becoming larger and more complex. Same protocols are being implemented in many different applications, by different vendors. As tasks and interactions become more and more intensive, however, they are also getting more unreliable. Therefore testing of systems and verification of protocols continuously gain importance.

The systems that are being dealt with are enormous, therefore exhaustive testing is trivially out of question. Mainstream testing procedures can cover only a small fraction of cases, and even with expert guidance toward boundary conditions and known possible errors, are therefore inadequate, and let many bugs slip into releases. Software is costly to maintain in any case, and in it is used to control some sort of critical application, the risks involved increase the operation costs even more. Demand for system reliability necessitates reliable testing methods that can, at least to some extent, offer guarantees.

Finite state machines are widely used to model system behavior in many areas, including circuits, software, and communication protocols. The demand for formally structured test generation motivates their study and devising of methods of testing that rely on their properties. Examining aspects of model behavior can ensure reliability of protocols, and the models can be used in order to devise methods to check the reliability of various implementations.

There is a wide literature on testing based on finite state machines dating back to 50s. Moore's study [7] of machine identification problem, which is

determining the state diagram of a given unknown machine according to its I/O behavior, introduced the framework for testing problems. As an additional problem Moore outlined conformance testing,

Conformance testing problem introduced by him was answered by Hennie[3], who showed that if a distinguishing sequence[1] exists for the machine, a checking sequence[2], that is a sequence that answers the conformance testing problem, polynomial in its length and size of the machine is possible, and a checking sequence of exponential length can be constructed even if no distinguishing sequence can be found. Studies in 60s and 70s were motivated mainly by automata theory, theoretical bounds were shown and machine identification problem was still a point of interest. Afterwards the field was not very active until 90s, when it was resurrected due to its applications in testing communication protocols[6]. At this point, refinements on the methods gained importance, for formal testing methods are invariably of high complexity, and efficient implementation is key to their feasability.

Testing of hardware and software is an extensive field, and we are focusing on a very specific portion of it. The problem we are interested in is conformance testing, which can be stated as, given a specification finite state machine and a "black box" implementation for which we can only observe the input/output behavior, determining if it conforms to the specification. We are interested in testing the control portion of systems which we can model as ordinary finite state machines. Moreover, we are only interested in a specific

[1] A sequence that allows the initial state of the FSM to be identified according to the output whenever it is applied.

[2] A sequence that answers the conformance testing problem for the automaton when it is applied to FSM under certain constraints.

way of conformance testing which is based on state identification through distinguishing sequences. Accurately, we are interested in improvements on D-method that had first been proposed by Hennie[3].

We will outline several concepts and define the structures we will be using in the next section. In section three, we will deal with several checking sequence generation methods we will be studying. We will outline the improvements and new methods we have proposed in section four. In section five, we will examine our experiment results, comparing the methods in practice.

2 Preliminaries

2.1 Finite State Machines

As we have discussed above, the models we are working on are to be modeled as finite state machines. Specifically, we are interested in deterministic, completely specified, minimal Mealy Machines, which we will define in the following.

2.1.1 Mealy Machine Definition

A Mealy Machine M is defined by a 6-tuple $(S, s_1, \Sigma, \Lambda, \delta, \lambda)$ where S is a finite set of states, $s_1 \in S$ is a specific start state, Σ is a finite set that is the input alphabet, Λ is a finite set that is the output alphabet, $\delta : S \times \Sigma \rightarrow S$ is the next state function and $\lambda : S \times \Sigma \rightarrow \Lambda$ is the output function.

2.1.2 Determinism and Complete Specification

Machine M is deterministic if and only if δ and λ are functions rather than relations. In other words, there exists no more than one value for any given input. Machine M is completely specified if and only if δ and λ are total, that is there exists exactly one value for any given input.

Behavior models, by their nature, are deterministic, however, they may not be completely specified, and in practice they almost always are not. However, we can convert them to complete specifications by imposing some sort of behavior model, such as digesting unspecified input silently or failing in a predetermined manner and including this behavior in our specification models. The former corresponds to adding $\delta(s_i, x) = s_i$ and $\lambda(s_i, x) = \epsilon$

(where ϵ stands for null element, that is no output being produced) to each function whereever they are not specified for given s_i, x pair. The latter consists of addition of an error state s_e to S and adding transitions to the machine leading to the error state for every unspecified input.

2.1.3 Transition Definition

A transition τ is defined by a 3-tuple $(s_i, s_j, x/y)$ where $s_i \in S$ is a start state, $s_j \in S$ is a end state, $x \in \Sigma$ is an input and $y \in \Lambda$ is an output. Given a machine M, a transition $\tau = (s_i, s_j, x/y)$ belongs to M if and only if $\delta(s_i, x) = s_j$ and $\lambda(s_i, x) = y$.

2.1.4 Extension of Function Definitions to Sequences

Let us extend the definitions of δ and λ to handle input sequences such that where $\bar{x} = x_1 x_2 \ldots x_{n-1} x_n$

- $\delta(s_i, \bar{x}) = \delta(\delta(\ldots \delta(\delta(s_i, x_1), x_2) \ldots, x_{n-1}), x_n)$
- $\lambda(s_i, \bar{x}) = \lambda(s_i, x_1)\lambda(\delta(s_i, x_1), x_2) \ldots \lambda(\delta(s_i, x_1 x_2 \ldots x_{n-1}), x_n)$

We will use this barred notation ($\bar{x}$, $\bar{D}$ etc.) to denote sequences of symbols throughout the thesis.

2.1.5 Reachability, State Equivalence and Minimality

A Mealy Machine M is minimal if there exists no Mealy Machine M' such that M' produces the same output to every input as M and M' has fewer states than M.

A state s_j is reachable from a state s_i if and only if there exists an input sequence $\bar{x} \in \Sigma^*$ such that $\delta(s_i, \bar{x}) = s_j$.

An input sequence $\bar{x} \in \Sigma^*$ distinguishes two states s_i and s_j of M if $\lambda(s_i, \bar{x}) \neq \lambda(s_j, \bar{x})$. Two states s_i and s_j are equivalent if and only if for all input sequences $\bar{x} \in \Sigma^*$ $\lambda(s_i, \bar{x}) = \lambda(s_j, \bar{x})$, that is to say there are no input sequences that distinguish them.

Two machines M_1 and M_2 are equivalent if their start states are equivalent. A machine is minimal if and only if there exists no equivalent machine with a fewer number of states. Consequently, machine M is minimal if and only if all of its states are reachable from the initial state and no two states are equivalent.

2.2 Checking Sequences

2.2.1 Unique Input-Output Sequences

Given FSM M and a state s_i, a *unique input-output sequence* is an input sequence $\bar{U}_i$ that distinguishes s_i from all other states in S. Formally:

For $M = (S, s_1, \Sigma, \Lambda, \delta, \lambda), s_i \in S, \forall s_j \in S, \lambda(s_i, \bar{U}_i) = \lambda(s_j, \bar{U}_i) \Leftrightarrow s_i = s_j$

Consider FSM M_0 given in figure 1.

- $\lambda(s_1, ab) = 01$
- $\lambda(s_2, ab) = 10$
- $\lambda(s_3, ab) = 10$
- $\lambda(s_4, ab) = 00$

- $\lambda(s_5, ab) = 00$

Therefore $\bar{U_1} = ab$ is a UIO sequence for state s_1 in M_0.

2.2.2 Distinguishing Sequences

Given an FSM M, a distinguishing sequence is an input sequence $\bar{D}$ that distinguishes all states in the machine. In other words, a distinguishing sequence is an input sequence that produces a unique output when applied to each state, differentiating them. Formally:

For $M = (S, s_1, \Sigma, \Lambda, \delta, \lambda), \forall s_i, s_j \in S, \lambda(s_i, \bar{D}) = \lambda(s_j, \bar{D}) \Leftrightarrow s_i = s_j$

Consider FSM M_0 given in figure 1.

- $\lambda(s_1, abb) = 010$
- $\lambda(s_2, abb) = 100$
- $\lambda(s_3, abb) = 101$
- $\lambda(s_4, abb) = 000$
- $\lambda(s_5, abb) = 001$

Therefore $\bar{D} = abb$ is a distinguishing sequence for FSM M_0.

2.2.3 Distinguishing Machines and Checking Sequences

An input sequence $\bar{x} \in \Sigma^*$ is said to distinguish two machines M_1 and M_2 if and only if $\lambda(s_1^{M_1}, \bar{x}) \neq \lambda(s_1^{M_2}, \bar{x})$.

An input sequence $\bar{C} \in \Sigma^*$ is a checking sequence for M if and only if $\bar{x}$ distinguishes between M and all elements of a class of machines $\Phi(M)$ that are not equivalent to M, that is to say,

For $M = (S^M, s_1^M, \Sigma^M, \Lambda^M, \delta^M, \lambda^M), \forall M' \in \Phi(M), \lambda^M(s_1^M, \bar{C}) = \lambda^{M'}(s_1^{M'}, \bar{C}) \Leftrightarrow M \equiv M'$

2.3 Directed Graphs

A directed graph (digraph) G is define by a tuple (V, E) where V is a set of vertices and E is a set of directed edges such that $E \subset V \times V \times L$, where L is a set of labels.

2.3.1 Edges, Indegree and Outdegree

Like transitions from the previous chapter, edges can be represented by 3-tuples (v_i, v_j, l) where the edge leaves v_i, enters v_j (i.e. v_i and v_j are the tail and head of the edge respectively) and has label l. For a vertex $v \in V$, $indegree_E(v)$ denotes the number of edges entering v and $outdegree_E(v)$ denotes the number of edges leaving v.

2.3.2 Graph Representation of FSMs

Given an FSM M, it can be represented by a digraph in which each state $s_i \in S$ is represented by a vertex $v_i \in V$ and each transition $\tau = (s_i, s_j, x/y)$ by an edge $e = (v_i, v_j, x/y) \in E$. Moreover, we can represent transition sequences rather than single transitions using edges, such that $e = (v_i, v_j, \bar{x}/\bar{y})$ where $s_j = \delta(s_i, \bar{x})$ and $\bar{y} = \lambda(s_i, \bar{x})$.

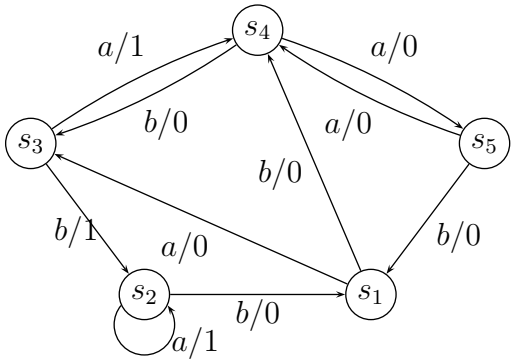


Figure 1: The FSM M_0

2.3.3 Walks and Tours on Graphs

A walk on a directed graph is a sequence of edges $\bar{W} = e_1, e_2, \ldots, e_r$ where the head of each edge is the same as the tail of the next. Where $\bar{W} = (n_1, n_2, x_1/y_1), (n_2, n_3, x_2/y_2), \ldots, (n_{r-1}, n_r, x_{r-1}/y_{r-1})$, n_1 is called the initial node, n_r is called the final node, and the sequence $\bar{T} = (x_1/y_1), (x_2/y_2), \ldots, (x_{r-1}/y_{r-1}) = (\bar{x}/\bar{y})$ is called the label of the walk $\bar{W}$. Given these, the walk $\bar{W}$ can be represented by the tuple $(n_1, n_r, \bar{T})$, and $\bar{T}$ is said to be a transfer sequence from n_1 to n_r.

A tour is a walk whose initial and final nodes are the same.

2.3.4 Reachability and Strong Connectivity

A node v_j is reachable from a node n_i if and only if there exists a walk from v_i to v_j.

A directed graph is strongly connected if and only if for any ordered pair (v_i, v_j) of vertices, there exists a walk from v_i to v_j, that is to say every node is reachable from every node.

2.3.5 Euler Tours

An Euler Path is a path that contains each edge on the graph exactly once. Similarly, an Euler Tour is a tour that contains each edge on the graph exactly once. If such a cycle exists on a graph, that graph is called Eulerian. A directed graph is Eulerian if and only if it is connected and symmetric, that is to say its every vertex has equal indegree and outdegree.

Construction of an Euler Tour on a connected, symmetric graph is simple:

- Find any cycle on the graph, removing every edge that is used
- If there are any remaining edges,
  - Pick a node on the cycle that has been found with nonzero indegree, find a cycle starting on it, removing the used edges
  - Insert the new cycle within the old cycle, between an edge entering and exiting the picked node
  - Repeat, until there are no unused edges

We are interested in Euler Tours because they are easy to construct, and we can use them to solve another problem we want to deal with, Rural Chinese Postman Path problem that we will explain below.

2.3.6 Rural Chinese Postman Tours

A Postman Path is a path that contains all edges of a graph. A Rural Postman Path is a path that contains a specific subset of the edges of the graph. A Rural Chinese Postman Path is a Rural Postman Path with minimum cost. A Rural Chinese Postman Tour is a RCPP that is also a tour.

The methods we are going to study use RCPT construction as the final step of their operation. Aho[1] proposes a method using flow networks to solve the problem, and the same approach is adopted in several later papers. The solution consists of building a symmetric, connected augmentation of the graph induced by the edges that have to be included in the rural tour using any edge in the graph, then finding an Euler Tour on the augmented graph.

Instead of using flow networks, we have formulated the construction of the symmetric augmentation as an integer programming problem defined as:

- Minimize $\sum_{\forall(s_i,s_j,l)\in E} c_{ijl}x_{ijl}$
- Subject to $\sum_{\forall(s_i,s_j,l)\in E,\forall s_k\in V} x_{ikl} - x_{kjl} = \sum_{\forall(s_i,s_j,l)\in E',\forall s_k\in V} n_{kjl} - n_{kil}$

Where $E' \subset E$ is the set of edges that have to be included in the rural path, c_{ijl} are costs of including the edge (s_i, s_j, l) once in the solution, n_{ijl} stands for the number of times the edge (s_i, s_j, l) has to be included, where x_{ijl} are the variables to be optimized, which correspond to addition of edges.

The formulation idea starts with a graph that contains the required edges and computes the degrees of each node. Then it aims to set the indegree-outdegree sums of all nodes to zero by adding edges from the overall graph, while aiming to minimize the cost. The x_{ijl} values stand for the number of times the edge (s_i, s_j, l) will be added to the graph to make it symmetric.

The augmented graph is symmetric, however it may be disconnected. If this is the case, we reformulate the problem, adding a constraint:

- Subject to $\sum_{\forall(s_i,s_j,l)\in V, s_i\in V_1, s_j\notin V_1} x_{ijl} + x_{jil} \geq 1$

Where V_1 is a component of the augmented graph that is disconnected from the rest of it. This constraint essentially mean that there will be at least one edge traversing the cut between V_1 and $V - V_1$, ensuring that the two components will be connected[3]. Then the optimization problem needs to be solved from scratch, and again checked for connectivity.

We do not add the connection conditions all at once, because there are exponentially many cuts on a graph, and we can in practice come up with a connected augmentation without resorting to listing all of them in our problem formulation. However, this formulation still has an exponential worst case complexity, as there is no formal guarantee that we will not have to add constraints for exponentially many cuts.

We could, as a compromise, resort to keeping the current augmentation as it is at any stage and formulate the optimization problem on it, resulting in a polynomial worst case complexity. However, this scheme would not yield an optimal solution to the RCPT problem.

Once we have defined a solution to the RCPT problem as such, we can generalize it to RCPP, which we are going to use. The exact specification of our formulation is as follows, where E_{imp} is the set of edges that are to be included in the RP, $G[E_{imp}]$ is the graph induced by it, $G' = (V', E')$ is the overall graph whose edges will be used in the augmentation, $s_1 \in V'$ is the start state of G' and $U' \subset V'$ is the set of possible final nodes for the RCPP we are looking for.

- Minimal symmetricity augmentation of $G[E_{imp}]$ in G':

[3]Note that, given that the graph is symmetric, connectivity trivially implies strong connectivity.

- Add vertex σ to V' and edges $e = (v, \sigma, \epsilon, \epsilon)$, $\forall v \in U'$, all with cost zero.
- Add edge $(\sigma, s_1, \epsilon, \epsilon)$ to E_{imp}
- $\forall v \in V'$, create constraints $\Sigma_{e \text{ entering } v, \forall e \in E'} X[e] - \Sigma_{e \text{ leaving } v, \forall e \in E'} X[e] = \Sigma_{e \text{ leaving } v, \forall e \in E_{imp}} 1 - \Sigma_{e \text{ entering } v, \forall e \in E_{imp}} 1$
- Solve the system so as to minimize the sum $\Sigma_{\forall e \in E'} X[e] \times cost[e]$
- Let $E_{new} = E_{imp}$, $\forall e \in E'$ add edge e to E_{new} $X[e]$ times. With this set let $G_\sigma = G[E_{new}]$
- Remove the edge entering and the edge leaving σ from E_{new}.
- Return $\bar{G} = G[E_{new}]$.

- If the result of the minimal symmetric augmentation part is connected, the RCPP is an Euler Path on $\bar{G}$. Otherwise G_σ as defined above is disconnected, and we add connectivity constraints.

- Adding connectivity constraints:
  - Maintain the symmetricity constraints from before
  - Find the connected component of G_σ that contains s_1. Let this subgraph be $G_1 = (V_1, E_1)$
  - Create equation $\Sigma_{e \text{ from } v \text{ to } u, v \in V_1, u \notin V_1, \forall e \in E'} X[e] > 0$
  - Solve the system so as to minimize the sum $\Sigma_{\forall e \in E'} X[e] \times cost[e]$
  - Construct E_{new} with the new $X[e]$ values
  - If G_σ is still not connected repeat the procedure

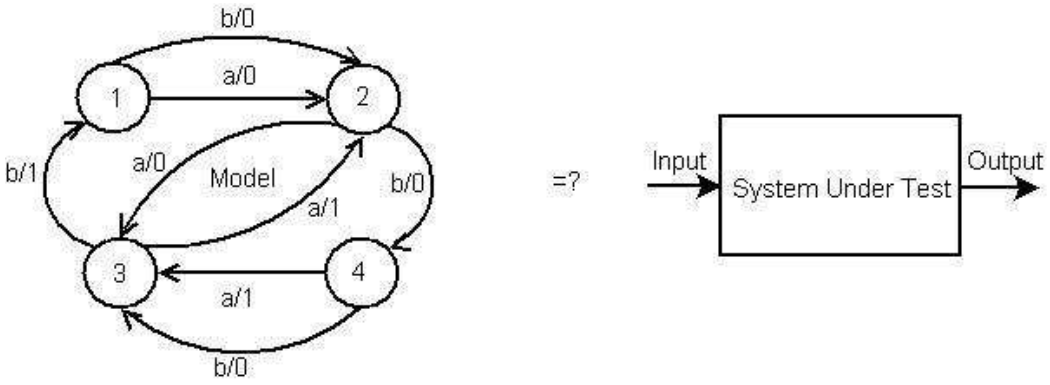


Figure 2: Conformance Testing Problem

  - Otherwise remove the edge entering and the edge leaving σ from E_{imp} and return $\bar{G} = G[E_{imp}]$

- Once $\bar{G}$ is symmetric and connected, the RCPP is an Euler Path on $\bar{G}$.

2.4 Recognizing States and Verifying Transitions

Our goal in conformance testing, as outlined in figure 2, is ensuring that a given black box system, of which we can only observe the input/output behavior, is equivalent to a given model specification. Both the implementation and specification are represented as Finite State Machines and our aim is to recognize every state of the specification in the implementation, then verify that every transition is between the correct ordered pairs of transitions and has the correct output.

Recognition of a state in our models depend upon the existence of a distinguishing sequence $\bar{D}$ for a model specification M with n states, and the assumption that any system under test I is deterministic, has the same input alphabet as M, does not change during operation, and has at most

n states. Therefore, if we observe n different responses to $\bar{D}$ from I, $\bar{D}$ is a distinguishing sequence for I. Once we observe this, we can use $\bar{D}$ to determine the structure of I, and check if it is equivalent to the model specification M.

Below we use $\bar{P}$ to denote a walk and $\bar{Q}$ to denote the label of $\bar{P}$.

2.4.1 D-recognition

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 $\bar{D}/\lambda(s, \bar{D})$.

We can extend this definition in two ways. First, we can do so in order to handle prefix DSs, that is to say, a prefix of the DS that has a unique output for some state s is enough to d-recognize the state s:

Let $\bar{D}'$ be a prefix of $\bar{D}$ such that $s \in S, \forall s_j \in S, \lambda(s, \bar{D}') = \lambda(s_j, \bar{D}') \Leftrightarrow s = s_j$. 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 $\bar{D}'/\lambda(s, \bar{D}')$.

We can also extend the definition to handle any transfer sequences that are appended to any such prefix:

Let $\bar{D}'$ be a prefix of $\bar{D}$ such that $s \in S, \forall s_j \in S, \lambda(s, \bar{D}') = \lambda(s_j, \bar{D}') \Leftrightarrow s = s_j$. Let $\bar{B}$ be an arbitrary transfer sequence. 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 $\bar{D}'\bar{B}/\lambda(s, \bar{D}'\bar{B})$.

2.4.2 T-recognition

Another assumption that is made during black box testing is that the structure of the system under test remains the same. Therefore once we recognize

to which state a given sequence takes the system when applied at a recognized state, we can conclude that it will take the system to that state on every occasion.

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 of $\bar{T}$ (where $\bar{D}'$ is defined as above, hence n_q and n_j are d-recognized in $\bar{Q}$ as state s of M). Suppose also that n_i is d-recognized in $\bar{Q}$ as state s' of M. Then state n_k is t-recognized in $\bar{Q}$ as s'.

Once we have t-recognized nodes in this manner, we can extend the definition to use these nodes to recognize other nodes as well.

Suppose that $(n_q, n_i, \bar{A})$ and $(n_j, n_k, \bar{A})$ are subpaths of $\bar{P}$ where $\bar{A}$ is an arbitrary sequence, and n_q and n_j are recognized in $\bar{Q}$ as state s of M. Suppose also that n_i is recognized in $\bar{Q}$ as state s' of M. Then state n_k is t-recognized in $\bar{Q}$ as s'.

Then we can establish the relation between the graph and the corresponding FSM using the recognition information.

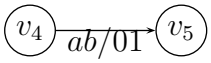
If node n_i of $\bar{P}$ is either d-recognized or t-recognized in $\bar{Q}$ as state s then n_i is recognized in $\bar{Q}$ as state s.

2.4.3 Edge Verification

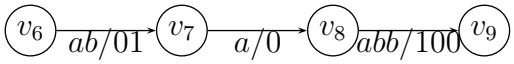
Finally, we can use the recognized states to verify transitions.

A transition $t = (s_i, s_j, x/y)$ is said to be verified in $\bar{P}$ 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_i, n_{i+1} is recognized as s_j, $x_i = x$ and $y_i = y$. This verification can be accomplished by a subpath $\bar{P}' = x\bar{D}_j/y\lambda(s, \bar{D}_j)$ of $\bar{P}$ where initial node of $\bar{P}'$ is recognized as s_i. The subpath $\bar{P}'$ is called the transition test for transition t.

(a) v_1 is d-recognized as s_1 and v_2 is d-recognized as s_2 by the corresponding distinguishing sequence prefixes



(b) v_4 is d-recognized as s_1 and v_5 is t-recognized as s_2 using the information from the fragment above



(c) v_6 is d-recognized as s_1, v_7 is t-recognized as s_2, v_8 is d-recognized as s_2, and hence the endpoints of the transition labeled $a/0$ is recognized as s_2, which means transition $(s_2, s_2, a/0)$ of M is verified

Figure 3: D-recognition, T-recognition and Edge Verification on M_0

2.4.4 Checking Sequences Implementing Edge Verification

Let $\bar{P}$ be a walk from G representing M 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 the input portion of $\bar{Q}$ is a checking sequence of M.

In every method we are dealing with, checking sequence generation is based on this result.

2.4.5 α-Sequences

α-sequences are constructs that serve the dual purpose of verifying the states of a system under test and recognizing the ending point of distinguishing sequences so that at any point in the generated sequence the point a DS takes the system to can be t-recognized.

To serve their purpose they should, considered together, contain distinguishing sequences initiating from each state, and recognize the endpoint for all of these distinguishing sequences.

- Select subsets $V_k \subset V$, a set which corresponds to the set of states of machine M, where $\bigcup_{k=1}^{q} V_k = V$
- Where $V_k = \{v_1^k, \ldots, v_{m_k}^k\}$ where $s_{f(i,k)}$ is the state of M represented by v_i^k
- Let $\delta(s_{f(i,k),\bar{T}_f(i,k)} = s_{f(i+1,k)}, \forall k, \forall i < m_k$
- Define $\alpha_k = \bar{T}_f(1,k) \ldots \bar{T}_f(m_k,k)\bar{T}_f(w,k)$ where $1 \leq w \leq m_k$
- Define α-set $A = \{\alpha_k, \forall k\}$

A construction scheme for an α-set with minimal total length is as follows:

- Let $V_D = V$
- Build a set of edges $E_D = \{(s_i, \delta(s_i, \bar{T}_i), \bar{T}_i/\lambda(s_i, \bar{T}_i)), \forall s_i \in V_D\}$
- Let $G_D = (V_D, E_D)$
- For each node of G_D with indegree zero
  - Do a DFS starting at that state until the traversed path reaches a node that has been visited on current search
  - Add the next edge to the path and let the label of this walk be an α-sequence
  - Mark all visited nodes to show that no new DFS need to start on any of them
- If there are any unmarked nodes remaining, pick an arbitrary node and repeat the above steps. Note that for all the paths that remain are cycles and the final edge added to the path will be the edge with which the path started

2.4.6 α'-Sequences

α'-sequences are a refinement to α-sequences, which serve the same purpose. The difference is, the sequences within the set utilize t-recognition using information from each other, hence they need not end with a T-sequence from wihtin themselves.

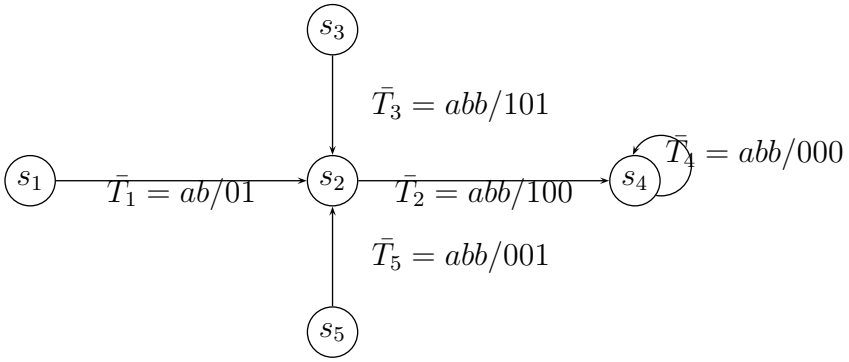
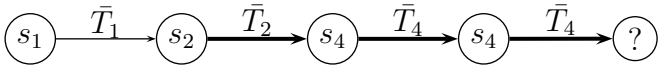


Figure 4: Graph of T-sequences that are to be used in construction of α-sequences

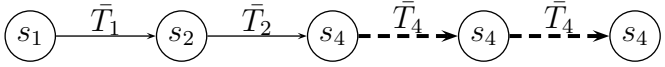
- Select subsets $V_k \subset V$, a set which corresponds to the set of states of machine M, where $\bigcup_{k=1}^{q} V_k = V$
- Where $V_k = \{v_1^k, \ldots, v_{m_k}^k\}$ where $s_{f(i,k)}$ is the state of M represented by v_i^k
- Let $\delta(s_{f(i,k),\bar{T}_f(i,k)} = s_{f(i+1,k)}, \forall k, \forall i < m_k$
- Define $\alpha'_k = \bar{T}_f(1,k) \ldots \bar{T}_f(m_k,k)\bar{T}_f(w,j)$ where $1 \leq w \leq m_k$ and $1 \leq j \leq q$
- Define α'-set $A = \{\alpha'_k, \forall k\}$

A construction scheme for an α'-set with minimal total length is as follows:

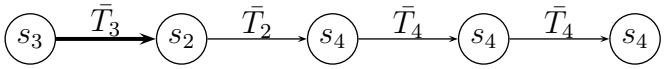
- Let $V_D = V$
- Build a set of edges $E_D = \{(s_i, \delta(s_i, \bar{T}_i), \bar{T}_i/\lambda(s_i, \bar{T}_i)), \forall s_i \in V_D\}$



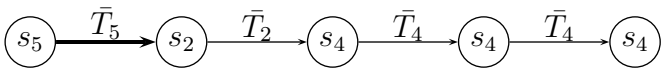
(a) Distinguishing sequences can be added for D-recognition until a node within the sequence is D-recognized again



(b) The endpoint of the overall sequence is then T-recognized for the same sequence has been verified before



(c) Unused T-sequences should be included in additional α-sequences



(d) Each T-sequence should be included in at least one α-sequence for α set to be complete

Figure 5: Construction of α-sequences for FSM M_0

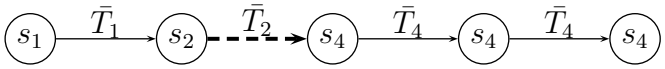
- Let $G_D = (V_D, E_D)$
- For each node of G_D with indegree zero
  - Do a DFS starting at that state until the traversed path reaches a marked node
  - Add the next edge to the path and let the label of this walk be an α'-sequence
  - Mark all visited nodes to show that they have been visited
- If there are any unmarked nodes remaining, pick an arbitrary node and repeat the above steps. Note that for all the paths that remain are cycles and the final edge added to the path will be the edge with which the path started

2.4.7 Special Sequences on M_0

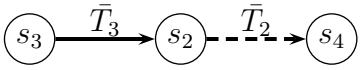
The input sequence abb is a distinguishing sequence for machine M_0. Moreover, its prefix ab is sufficient to distinguish state s_1. Also,

We choose to use empty transfer sequences, therefore:

- T-sequences:
  - $\bar{T}_1 = ab/01$, endpoint s_2
  - $\bar{T}_2 = abb/100$, endpoint s_4
  - $\bar{T}_3 = abb/101$, endpoint s_2
  - $\bar{T}_4 = abb/000$, endpoint s_4



(a) α'-sequences utilize D-recognition and T-recognition in the same manner as α-sequences



(b) Additionally, T-recognition can be used between α' elements to recognize endpoints, eliminating the need to always repeat a transition within the sequence

(c) As before, each T-sequence should be included in at least one α-sequence for α set to be complete

Figure 6: Construction of α'-sequences for FSM M_0

- $\bar{T}_5 = abb/001$, endpoint s_2

- α-sequences:
  - $\alpha'_1 = \bar{T}_1\bar{T}_2\bar{T}_4\bar{T}_4$, endpoint s_4
  - $\alpha'_2 = \bar{T}_3\bar{T}_2\bar{T}_4\bar{T}_4$, endpoint s_4
  - $\alpha'_3 = \bar{T}_5\bar{T}_2\bar{T}_4\bar{T}_4$, endpoint s_4

- α'-sequences:
  - $\alpha'_1 = \bar{T}_1\bar{T}_2\bar{T}_4\bar{T}_4$, endpoint s_4
  - $\alpha'_2 = \bar{T}_3\bar{T}_2$, endpoint s_4
  - $\alpha'_3 = \bar{T}_5\bar{T}_2$, endpoint s_4

- Some UIO sequences:
  - $\bar{U}_2 = aa/11$, endpoint s_2
  - $\bar{U}_3 = aa/10$, endpoint s_5

3 Existing Approaches

The approaches to checking sequence generation we will deal with have the same general structure. They all construct a specialized graph, and solve a RCPP problem on it to construct a checking sequence. The point they differ in is the construction of the graph and selection of the edges that will be mandatory in the rural path.

In all the methods, let:

- $M = (S, s_1, \Sigma, \Lambda, \delta, \lambda)$ be the specification FSM
- $\bar{D}$ be a distinguishing sequence for M
- $\bar{D}'_i$ be the shortest prefix of the distinguishing sequence $\bar{D}$ of the specification FSM M for the state $s_i \in S$ such that, $\forall s_j \in S, \lambda(s_i, \bar{D}') = \lambda(s_j, \bar{D}') \Leftrightarrow s_i = s_j$
- $t_i = \delta(s_i, \bar{D}'_i)$
- $\bar{B}_i$ an arbitrary (possibly empty) transfer sequence
- $\bar{T}_i = \bar{D}'_i \bar{B}_i$ be the sequence to be used to identify state s_i during the tests
- $f_i = \delta(s_i, \bar{T}_i)$
- $l_i = \lambda(s_i, \bar{T}_i)$

3.1 Using RCPP for Integrating Test Segments

The series of improvements we are dealing with begin with a method that constructs necessary test segments, build a graph using these and some connecting components as edges, and solving a RCPT problem on the graph [10].

3.1.1 Test Segments

As explained in part 2.5.4, to generate a checking sequence, we need to verify every edge of the given specification in the system under test. To serve this purpose,

- Transition tests $\bar{P}' = x\bar{D}_j/y\lambda(s, \bar{D}_j)$ are constructed for every transition $t = (s_i, s_j, x/y)$.
- α sequences are constructed for state recognition and to facilitate the use of $\bar{T}$-sequences for recognition of their endpoints.

And these test segments are used in construction of a graph that will be traversed in order to extract a combination of all these segments, to eventually generate a checking sequence. Exact construction of the graph is as follows.

3.1.2 Construction of the Augmented Graph

The checking sequence construction method first constructs a digraph G' by augmenting $G = (V, E)$ representing a model FSM M where:

- $U' = \{s'_i | \forall s_i \in V\}$ representing the set of recognized versions of the states in V
- $V' = V \cup U'$ is the set of vertices for the augmented graph G'
- $E_C = \{(s'_i, f'_j, x/yl_i) | \forall (s_i, s_j, x/y) \in E\}$ transition test segments
- $E_\alpha = \{(s_i, \delta(s_i, input(\alpha_k))', \alpha_k)\}$ where $initial(\alpha_k) = s_i$ representing the α edges, which are used to enable the T-recognition of final nodes of T-sequences
- $E_T = \{(s_i, f'_i; l_i)) | \forall s_i \in V\}$ representing $\bar{T}_i$s, which are used to D-recognize state s_i and move the system to the recognized state f'_i
- $E_\epsilon = \{(s'_i, s_i, \epsilon/\epsilon)\}$ representing moving from the recognized version of a state to its unrecognized version. These might be used to transfer the machine from one state to another
- $E'' \subset \{(s'_i, s'_j, x/y) | \forall (s_i, s_j, x/y) \in E\}$ a minimum spanning tree in U', to be used as transfer sequences, taking the machine to proper state for transition tests and alpha sequences
- Create $E' = E_C \cup E_T \cup E_\epsilon \cup E_{\alpha'} \cup E''$
- $G' = (V', E')$ is the augmented graph to be used in construction of the checking sequence
- $E_{imp} = E_C \cup E_{\alpha'}$ are the edges that have to be included in the rural path on G'

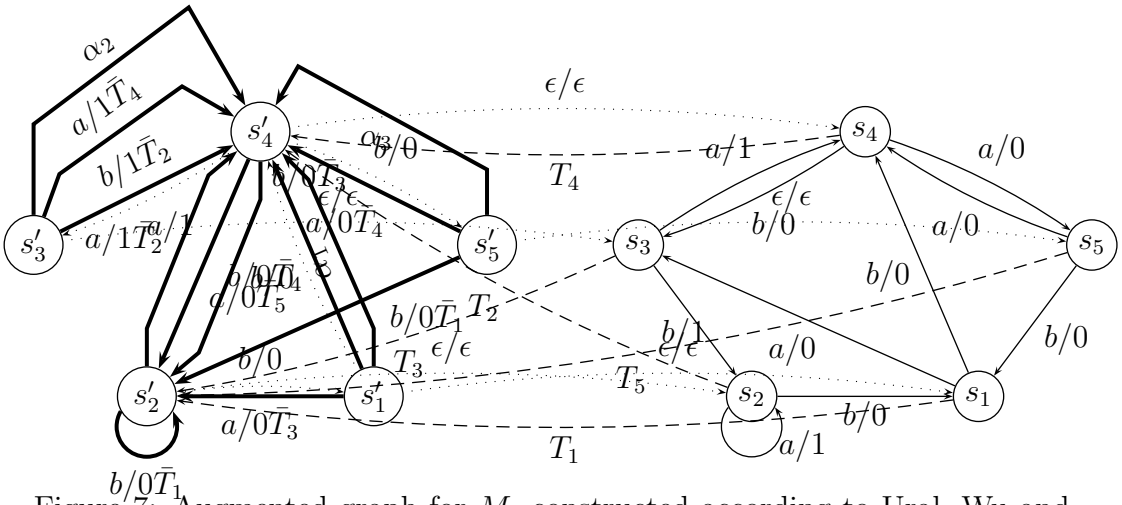


Figure 7: Augmented graph for M_0 constructed according to Ural, Wu and Zhang '97

A graph constructed according to this formulation for FSM M_0 is given in figure 7.

Once G' is thus constructed, an RCPP over the noted edges that starts at the vertex corresponding to the initial state of the specification automaton and ends in a recognized state can be found. The label of such a path is a checking sequence for the automaton.

Such a path is:

$$(v_1, v'_2, \bar{T}_1), (v'_2, v'_4, a/0\bar{T}_2), (v'_4, v'_2, a/1\bar{T}_5), (v'_2, v'_2, b/0\bar{T}_1), (v'_2, v'_1, b/0)$$

$$(v'_1, v'_4, \alpha_1), (v'_4, v'_2, b/0\bar{T}_3), (v'_2, v'_1, b/0), (v'_1, v'_2, a/0\bar{T}_3), (v'_2, v'_1, b/0),$$

$$(v'_1, v'_4, b/0), (v'_4, v'_3, b/0), (v'_3, v'_4, \alpha_2), (v'_4, v'_3, b/0), (v'_3, v'_4, a/1\bar{T}_4),$$

$$(v'_4, v'_3, b/0), (v'_3, v'_4, b/1\bar{T}_2), (v'_4, v'_5, a/0), (v'_5, v'_4, \alpha_3), (v'_4, v'_5, a/0),$$

$$(v'_5, v'_4, a/0\bar{T}_4), (v'_4, v'_5, a/0), (v'_5, v'_2, b/0\bar{T}_1), (v'_2, v'_1, b/0), (v'_1, v'_4, b/0\bar{T}_4)$$

whose label is a checking sequence of length 86.

3.2 Optimizations on Test Segments

Some later revisions[5][4] to the scheme presented above introduced several refinements. These are essentially modifications on construction of test segments, however, the structure of the augmented graph is significantly altered.

3.2.1 Test Segments

The focus of the refinement is the α sequences. The scheme proposed not only shortens their length, but also utilized them in transition verification.

- α-sequences are replaced by α' sequences as described in section 2.5.6. Moreover, α' sequences are used for transition verification as well, replacing distinguishing sequences where viable.
- Transition tests are not represented by single edges, but are divided into transition and identification parts instead. A transition edge takes the machine to an unidentified state, which in turn is identified by either a distinguishing sequencs or an α' sequence.

These test segments are again used in construction of a graph that will be traversed in order to extract a combination of all these segments, to eventually generate a checking sequence. Exact construction of the graph is as follows.

3.2.2 Construction of the Augmented Graph

The checking sequence construction method first constructs a digraph G' by augmenting $G = (V, E)$ representing a model FSM M where:

- $U' = \{s'_i | \forall s_i \in V\}$ representing the set of recognized versions of the states in V
- $V' = V \cup U'$ is the set of vertices for the augmented graph G'
- $E_C = \{(s'_i, s_j, x/y) | \forall (s_i, s_j, x/y) \in E\}$ representing edges in V, so that the transitions to be verified start at recognized versions of the states they belong
- $E_{\alpha'}$ representing the α' edges, which are used to D-recognize state s_i and move the system to the recognized state f'_j as well as enable the T-recognition of final nodes of T-sequences
- $E_T = \{(s_i, f'_i; l_i)) | \forall s_i \in V\}$ representing $\bar{T}_i$s, which are used to D-recognize state s_i and move the system to the recognized state f'_i
- $E'' \subset \{(s'_i, s'_j, x/y) | \forall (s_i, s_j, x/y) \in E\}$ such that $G'' = (V', E'')$ does not have a tour, and $G' = (V', E')$ is strongly connected
- Create $E' = E_C \cup E_T \cup E_{\alpha'} \cup E''$
- $G' = (V', E')$ is the augmented graph to be used in construction of the checking sequence
- $E_{imp} = E_C \cup E_{\alpha'}$ are the edges that have to be included in the rural path on G'

A graph constructed according to this formulation for FSM M_0 is given in figure 8. The nodes in V and U' are at the bottom, and at the top respectively. The dashed lines are the edges in E_T, and the dotted lines are the edges in E''. The edges in $E_{\alpha'} \cup E_C$ are given in bold solid lines.

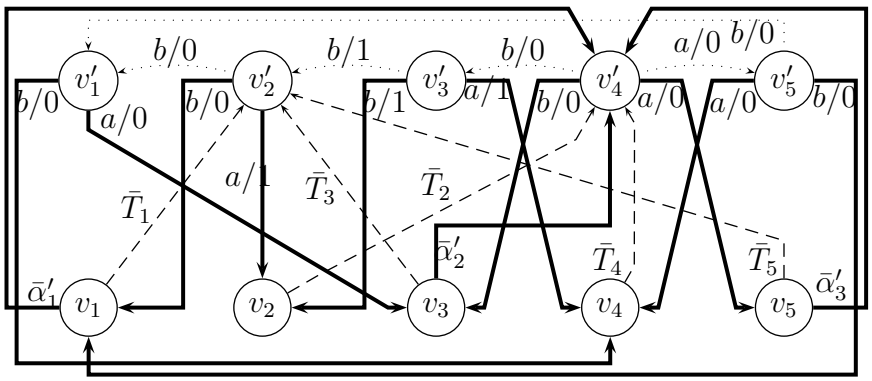


Figure 8: $G' = (V', E')$ for M_0, constructed according to Ural and Hierons '06

Once G' is thus constructed, a RCPP over the noted edges that starts at the vertex corresponding to the initial state of the specification automaton and ends in a recognized state can be found. The label of such a path is a checking sequence for the automaton.

It is proposed in the paper that a tour concatenated with a T-sequence should be found. This is a sufficient condition, but a tour is not necessary. A path that ends in a recognized state is sufficient.

A path over G' containing all edges in $E_C \cup E_{\alpha'}$ is:

$$(v_1, v'_4, \alpha'_1), (v'_4, v_5, a/0), (v_5, v'_4, \alpha'_3), (v'_4, v_3, b/0), (v_3, v'_4, \alpha'_2)$$

$$(v'_4, v'_5, a/0), (v'_5, v_4, a/1), (v_4, v'_4, \bar{T}_4), (v'_4, v'_5, a/0), (v'_5, v_1, b/0),$$

$$(v_1, v'_2, \bar{T}_1), (v'_2, v_2, a/1), (v_2, v'_4, \bar{T}_2), (v'_4, v'_3, b/0), (v'_3, v_4, a/1),$$

$$(v_4, v'_4, \bar{T}_4), (v'_4, v'_3, b/0), (v'_3, v_2, b/1), (v_2, v'_4, \bar{T}_2), (v'_4, v'_5, a/0),$$

$$(v'_5, v'_1, b/0), (v'_1, v_3, a/0), (v_3, v'_2, \bar{T}_2), (v'_2, v_1, b/0), (v_1, v'_2, \bar{T}'_2),$$

$$(v_2', v_1', b/0), (v_1', v_4, b/0), (v_4, v_4', \bar{T}_4)$$

whose label is a checking sequence of length 63.

3.3 Overlap Elimination

Ural and Zhang[9] proposes a method to take advantage of overlaps in test segments and generate α' sequences to be constructed dynamically.

3.3.1 D-overlapping

Let,

- $\bar{P}_1$ and $\bar{P}_2$ be two walks of G
- $\bar{P}_1 = \bar{R}_1\bar{R}$ and $\bar{P}_2 = \bar{R}\bar{R}_2$ for some walks $\bar{R}_1$, $\bar{R}$ and $\bar{R}_2$

$\bar{P}_1$ and $\bar{P}_2$ are said to overlap by $\bar{R}$.

If additionally $\bar{P}_2$ has a distinguishing sequence for its initial state as a prefix, this overlap is called a D-overlap. Then $\bar{P}_1$ and $\bar{P}_2$ can be combined into a new walk $\bar{P}_{12} = \bar{R}_1\bar{R}\bar{R}_2$. If $\bar{P}_1$ and $\bar{P}_2$ are test segments, the two segments can be replaced by $\bar{P}_{12}$, effectively shortening the overall length by length of $\bar{R}$.

3.3.2 Test Segments

The test segments are not very different from what was given in part 3.1.1. However, the way they are used to build the augmented graph and combined to facilitete overlapping cause the results be more like what we have seen in part 3.2.

- Transition tests in the form of $\bar{P}' = x\bar{D}_j/y\lambda(s, \bar{D}_j)$ are created for every transition in the specification.

- α' sequences are replaced by structures named α-elements, which are essentially tests that recognize the endpoints of T-sequences, for every state. Their combination in a manner resembling the formation of α' sequences is a part of overlap elimination process.

However, unlike other methods we have dealt with, these segments are not connected in generic versions of recognized nodes. Instead each test segment is between nodes that are particular to that test. These nodes are then connected by edges that denote D-overlappings between test segments.

Exact construction of the augmented graph is as follows:

3.3.3 Construction of the Augmented Graph

- $P_C = \{(s_i, f_j; x/yl_j) | \forall (s_i, \delta(s_i, x); x/y) \in E\}$, transition test segments

- $P_\alpha = \{(s_i, f_{f_i}; l_i l_{f_i}) | \forall s_i \in V\}$, α elements

- $V' = \{s'_{i\tau} | \forall \tau = (s_i, s_j; I_\tau/O_\tau) \in P_\alpha \cup P_C\}$, initial nodes of test segments

- $V'' = \{s''_{j\tau} | \forall \tau = (s_i, s_j; I_\tau/O_\tau) \in P_\alpha \cup P_C\}$, final nodes of test segments

- $V^* = V \cup V' \cup V'' \cup \{s_1^*\}$, the set of vertices for augmented graph G^*

- $E_0 = \{(s_1^*, s'_{1\tau}; \epsilon)\ with\ cost\ 0, \forall \tau = (s_1, s_j; I_\tau/O_\tau) \in P_\alpha \cup P_C\ s.t\ D\ is\ a\ prefix\ of\ I_\tau\}$, the set of starting edges

- $E_\alpha = \{(s'_{i\tau}, s''_{j\tau}; I_\tau/O_\tau)\ with\ cost\ |I_\tau|, \forall \tau = (s_i, s_j; I_\tau/O_\tau) \in P_\alpha\}$, the set of α-edges

- $E_C = \{(s'_{i\tau}, s''_{j\tau}; I_\tau/O_\tau)\ with\ cost\ |I_\tau|, \forall \tau = (s_i, s_j; I_\tau/O_\tau) \in P_C\}$, the set of transition test edges
- $E' = \{(s_i, s'_{i\tau}; \epsilon)\ with\ cost\ 0, \forall \tau = (s_i, s_j; I_\tau/O_\tau) \in P_\alpha \cup P_C\}$, the set of edges that move the system to initial states of test segments from the connecting part of the augmented graph
- $E'' = \{(s''_{j\tau}, s_j; \epsilon)\ with\ cost\ 0, \forall \tau = (s_i, s_j; I_\tau/O_\tau) \in P_\alpha \cup P_C\}$, the set of edges that move the system from final states of test segments to connecting part of the augmented graph
- $E_D = \{(s''_{j\tau}, s'_{k\mu}; \epsilon)\}\ with\ cost\ -|R|, \forall \tau = (s_i, s_j; I_\tau/O_\tau), \mu = (s_k, s_r; I_\mu/O_\mu) \in P_\alpha \cup P_C\ s.t.\ \tau\ D-overlaps\ \mu\ by\ some\ R\}$, the set of edges representing D-overlaps from final to initial states of test segments
- $E^* = E \cup E_0 \cup E_\alpha \cup E_C \cup E' \cup E'' \cup E_D$
- $G^* = (V^*, E^*)$ is the augmented graph that will be used to create a checking sequence
- $E_{imp} = E_C \cup E_\alpha$ is the set of edges that have to be included in the rural path on G^*

The scheme yields very good results when there is a large amount of D-overlaps, which typically happens when distinguishing sequence is composed of a single input character, but otherwise yields very similar results as the method using optimized test segments otherwise[4]. If distinguishing sequence is not composed of a single character, D-overlaps only happen when the second test segment in question is an α-element, therefore this is to be expected.

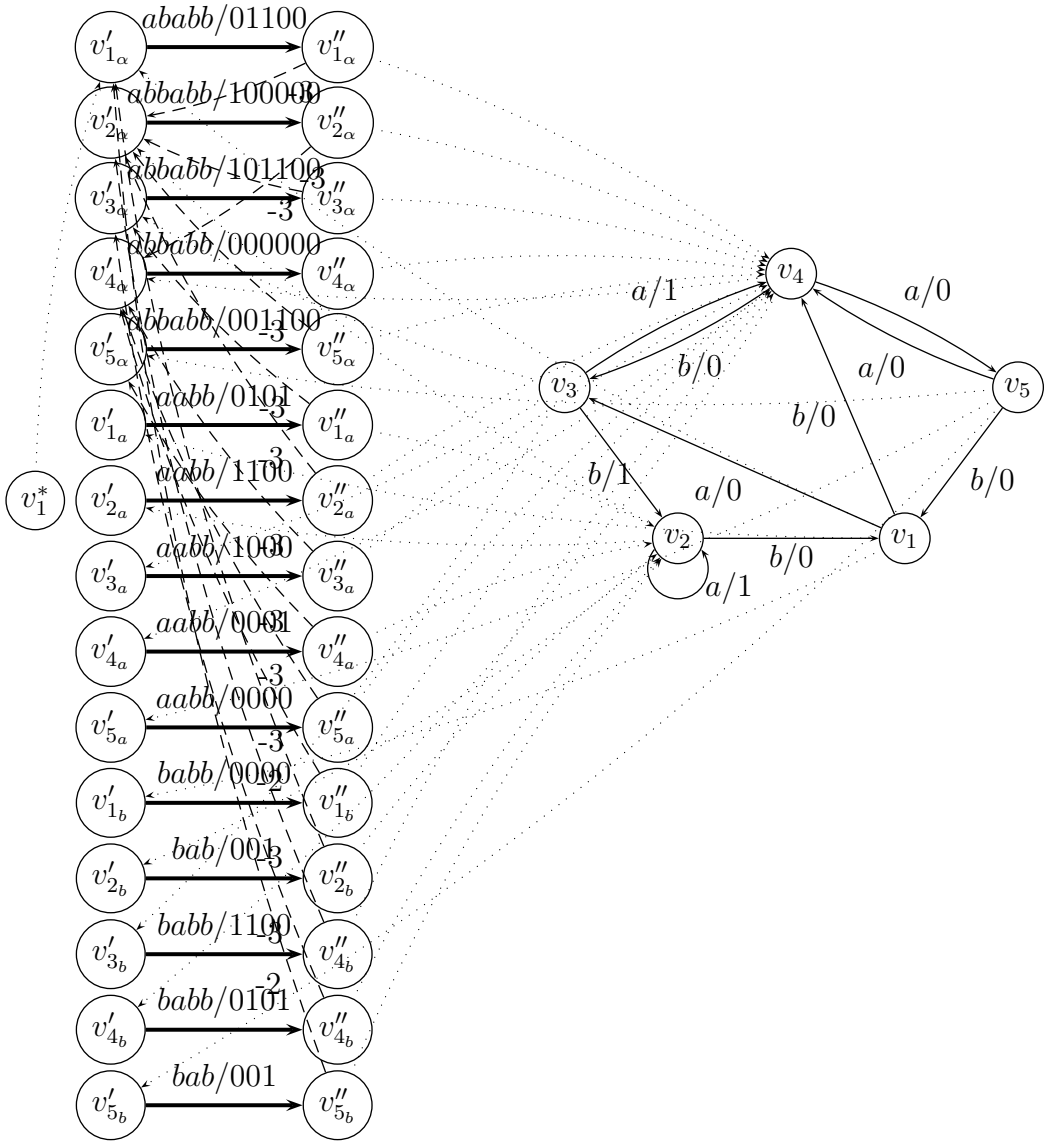


Figure 9: Augmented graph for M_0 constructed according to Ural and Zhang

A graph constructed using this scheme for M_0 is given in figure 9.

Once G^* is thus constructed, an RCPP over the noted edges that starts at the vertex v_1^* and ends within V'' can be found. The label of such a path then can be processed to find a checking sequence for the automaton. Essentially, each negative cost edge (D-overlap edge) with cost $-c$ implies that the next c characters should be deleted from the label to generate a checking sequence.

A checking sequence equivalent to that generated by part 3.2 can be found for FSM M_0, with length 63. However, the optimization problem is inflated due to the size of the augmented graph and the corresponding walk contains over three times as many edges. As noted above, this scheme does not offer any advantages when the distinguishing sequence does not allow many D-overlaps.

3.4 Initial Idea for Redundant Transition Test Elimination

The methods we have considered so far all construct transition tests for every transition. However, that may not be necessary in every case, as some transitions may be verified in some other part of the checking sequence implicitly. A paper by Chen, Hierons, Ural and Yenigun[2] aims to identify some such transitions and exempt from explicit testing.

The basic idea is that, within α' sequences, test segments in the form of $\bar{P}' = x\bar{D}_j/y\lambda(s, \bar{D}_j)$ exist, where x/y is the last transition of some T-sequence, and the distinguishing sequence is the following T-sequence used in the construction of α' sequence. If the initial node in the path can be recognized, the transition can be verified at that point. In this case, this is

accomplished by T-recognition, therefore, every transition on the T-sequence whose last transition is to be exempted from tests needs to have been verified.

In the following, we will define a set of edges $L \subset E$ that need not be subjected to transition tests to be verified given some α' set A.

3.4.1 Identifying Redundant Transition Tests

Let $R_i = e_{i_1} e_{i_2} \dots e_{i_h}$ be the sequence of edges corresponding to application of $\bar{T}_i$ at state s_i. If $\forall r, 1 \leq r < h$, e_{i_r} is verified in $\bar{Q}$, e_{i_l} is verified in $\bar{Q}$.

Therefore we can pick edges that can be verified as a result of this condition and exempt them from transition tests. However, if exemption of an edge e depends on verification of another edge e' when exemption of e' depends on e, we cannot exclude both edges from transition tests.

Also note that two paths R_i and R_j, corresponding to application of $\bar{T}_i$ at state s_i and $\bar{T}_j$ at state s_j respectively, may end with the same transition e. Transition e can be excluded from the transition tests if the conditions imposed by either one is satisfied. Therefore a straightforward dependency graph as proposed in the paper is not effective in accurately representing the relationships between transition tests.

The following construction gives an accurate representation of the dependencies:

Let:

- $V_E = \{s_{ix} | \forall (s_i, s_j, x/y) \in E\}$ be a set of states representing the transitions of FSM M
- $V_S = \{s^k | \forall R_k\}$ be a set of states representing last edges on all paths R_i

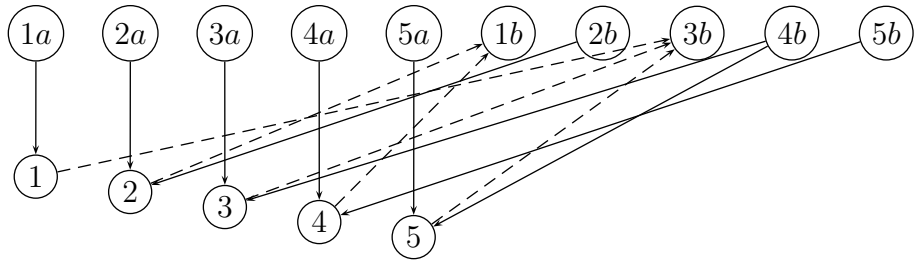


Figure 10: Full Dependency Graph of M_0 for Basic Redundant Transition Test Elimination

- $V_D = V_E \cup V_S$ be the set of states for the dependency graph
- $E_R = \{(s_{i_j^k x_j^k}, s^k, \epsilon/\epsilon) | \forall t_n^k, 1 \leq j < n\}$ be the set representing the dependencies of an edge to edges before it
- Let $E_A = \{(s^k, s_{i_n^k x_n^k}, \epsilon/\epsilon) | \forall t^k$ *such that* t^k *is the last edge on some* $R_i\}$ be the set representing exemption of a transition test given that dependencies of node s^k are satisfied
- $E_D = E_R \cup E_A$
- Remove nodes from V_S until G_D is acyclic, while aiming to maximize cardinality of L
- $L = \{(s_i', s_j, x/y) | \forall (s_k, s_i x, \epsilon/\epsilon) \in E_D\}$

Once G_D is constructed, all cycles on it have to be removed, through removal of nodes in V_S, which corresponds to refraining from using the cor-

responding $\bar{R}_i$ edge for exemption. Any acyclic subgraph of G_D thus constructed gives us a valid set L of transitions that can be excluded from tests. After cycle removal, all nodes in V_D with an incoming edge can be included in L. Therefore the cycle removal algorithm should aim to maximize the number of V_D elements with incoming transitions.

The maximal acyclic subgraph problem itself is NP-complete, so we do not aim for the best result in all cases. Also note that a maximal acyclic subgraph does not necessarily constitute as the best solution to this problem, as our main interest is preserving the maximum amount of V_E edges with incoming transitions. However, this construction yields a rather simple dependency graph, therefore we can make a good selection of transition tests to be eliminated with a simple cycle removal scheme.

The dependency graph constructed according to this formulation for FSM M_0 is given in figure 10. The graph is acyclic, therefore transitions $(s_1, s_4, b/0)$ and $(s_3, s_2, b/1)$ can be exempted from transition tests.

3.4.2 Checking Sequence Construction

The checking sequence construction method first constructs a digraph G' by augmenting $G = (V, E)$ representing a model FSM M where:

- $U' = \{s'_i | \forall s_i \in V\}$ representing the set of recognized versions of the states in V
- $V' = V \cup U'$ is the set of vertices for the augmented graph G'
- $E_C = \{(s'_i, s_j, x/y) | \forall (s_i, s_j, x/y) \in E\}$ representing edges in V, so that the transitions to be verified start at recognized versions of the states

they belong

- $E_{\alpha'}$ representing the α' edges, which are used to D-recognize state s_i and move the system to the recognized state f'_j as well as enable the T-recognition of final nodes of T-sequences
- $E_T = \{(s_i, f'_i; l_i))|\forall s_i \in V\}$ representing $\bar{T}_i$s, which are used to D-recognize state s_i and move the system to the recognized state f'_i
- $E'' \subset \{(s'_i, s'_j, x/y)|\forall (s_i, s_j, x/y) \in E\}$ such that $G'' = (V', E'')$ does not have a tour, and $G' = (V', E')$ is strongly connected
- Pick a subset $L' \subset L$ such that after setting $E' = E' - (s'_i, s'_j, x/y)|(s'_i, s_j, x/y) \in L$, a valid RCPP on G' can still be found[4]
- Set the set of connecting transition candidates $E'' = E'' - (s'_i, s'_j, x/y)|(s'_i, s_j, x/y) \in L'$
- Set the set of necessary transitions tests $E'_C = E_C - L'$
- Create $E' = E_C \cup E_T \cup E_{\alpha'} \cup E''$
- $G' = (V', E')$ is the augmented graph to be used in construction of the checking sequence
- $E_{imp} = E'_C \cup E_{\alpha'}$ are the edges that have to be included in the rural path on G'

A graph constructed according to this formulation for FSM M_0 is given in figure 11. The nodes in V and U' are at the bottom, and at the top respectively. The dashed lines are the edges in E_T, and the dotted lines are

[4] A sufficient condition is maintaining strong connectivity, but it is not always necessary.

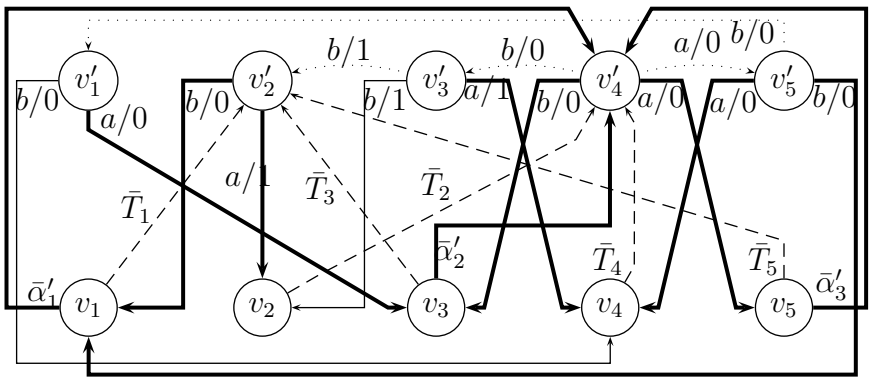


Figure 11: $G' = (V', E')$ for M_0, constructed according to Chen, Hierons, Ural and Yenigun '05

the edges in E''. The edges in $E_{\alpha'} \cup E'_C$ are given in bold solid lines. The rest of the solid lines are the edges in L.

Once G' is thus constructed, a RCPP over the noted edges that starts at the vertex corresponding to the initial state of the specification automaton and ends in a recognized state can be found. The label of such a path is a checking sequence for the automaton.

Again, it is proposed in the paper that a tour concatenated with a T-sequence should be found. This is a sufficient condition, but a tour is not necessary. As before, a path that ends in a recognized state is sufficient.

A path over G' containing all edges in $E'_C \cup E_{\alpha'}$ is:

$$(v_1, v'_4, \alpha'_1), (v'_4, v_5, a/0), (v_5, v'_4, \alpha'_3), (v'_4, v_3, b/0), (v_3, v'_4, \alpha'_2)$$

$$(v'_4, v'_5, a/0), (v'_5, v_4, a/1), (v_4, v'_4, \bar{T}_4), (v'_4, v'_5, a/0), (v'_5, v_1, b/0),$$

$$(v_1, v'_2, \bar{T}_1), (v'_2, v_1, b/0), (v_1, v'_2, \bar{T}_2), (v'_2, v_2, a/1), (v_2, v'_4, \bar{T}_2),$$

$$(v'_4, v'_3, b/0), (v'_3, v_4, a/1), (v_4, v'_4, \bar{T}_4), (v'_4, v'_5, a/0), (v'_5, v'_1, b/0),$$

$$(v'_1, v_3, a/0), (v_3, v'_2, \bar{T}_3)$$

whose label is a checking sequence of length 53.

4 Proposed Improvements

All methods propose sufficient conditions for checking sequence generation. However, their conditions can be relaxed even more. What we propose is a new way to recognize states through a sequence.

4.1 Expanding the Dependency Graph

The dependency graph scheme given by Chen, Hierons, Ural and Yenigun[2] does not precisely represent the actual dependencies between the transitions that can be saved. If a there is more than one way to save a given transition, the dependencies in the two cases are not related, and satisfaction of either set of dependencies would allow us to exempt the given transition from explicit testing. However, in a straightforward representation including only the transitions and dependencies between them, there is no way to distinguish these separate sets of dependencies. As a result, that construction gives a sufficient condition, however, the condition is not necessary and a more accurate representation results in a salection that can exempt more transitions from explicit testing.

Consider the FSM M_1 given in figure 12, which has a distinguishing sequence of aa.

- $\bar{D}_1 = (s_1, s_2, a/0)(s_2, s_3, a/0)$
- $\bar{D}_2 = (s_2, s_3, a/0)(s_3, s_2, a/1)$
- $\bar{D}_3 = (s_3, s_2, a/1)(s_2, s_3, a/0)$

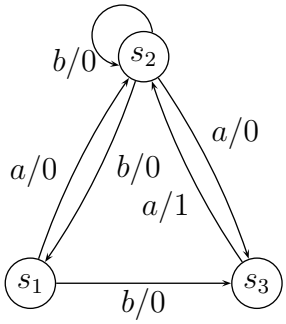
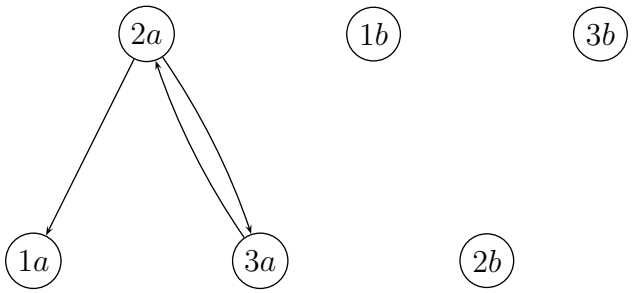
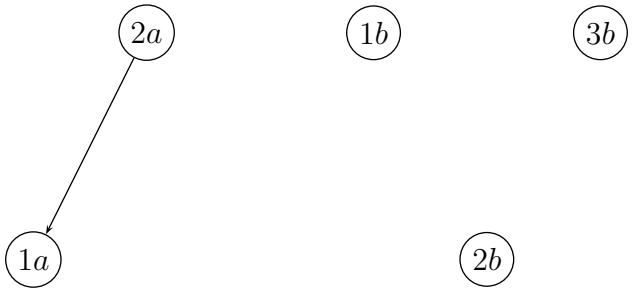


Figure 12: The FSM M_1

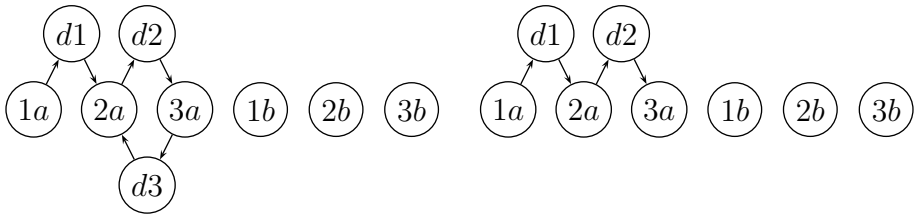


(a) Before Cycle Removal



(b) After Cycle Removal

Figure 13: Dependency graph for M_1



(a) Before Cycle Removal (b) After Cycle Removal

Figure 14: Expanded dependency graph for M_1

The resulting dependency graph according to the method proposed in the paper is shown in figure 13. Exemption of the transition $(s_2, s_3, a/0)$ using $\bar{D}_1$ depends on verification of the transition $(s_1, s_2, a/0)$. Likewise, $(s_3, s_2, a/1)$ using $\bar{D}_2$ depends on verification of the transition $(s_2, s_3, a/0)$. The problem part is that the transition $(s_2, s_3, a/0)$ can also be exampted using $\bar{D}_3$, if $(s_3, s_2, a/1)$ is verified. When there is no distinction between the two dependency conditions, cycle removal on the dependency graph allows exemption of either $(s_2, s_3, a/0)$ or $(s_3, s_2, a/1)$, not both.

The accurate representation[5] given in figure 14 on the other hand, allows both transitions to be saved. Transition $(s_2, s_3, a/0)$ is verified on $\bar{D}_1$ depending on transition $(s_1, s_2, a/0)$ as represented by node $d1$, and $(s_3, s_2, a/1)$ is verified on $\bar{D}_2$ as represented by node $d2$ depending on transition $(s_2, s_3, a/0)$.

Of course the problem in general is more complicated than shown in this example, where each way of saving a transition depended on one other transition only. Normally, each way of saving a transition depends on a set of transitions, all of which have to be satisfied for exempting the transition in

[5]Note that the directions for dependencies are reversed in our construction. This is merely a design choice and is equivalent to the inverse construction.

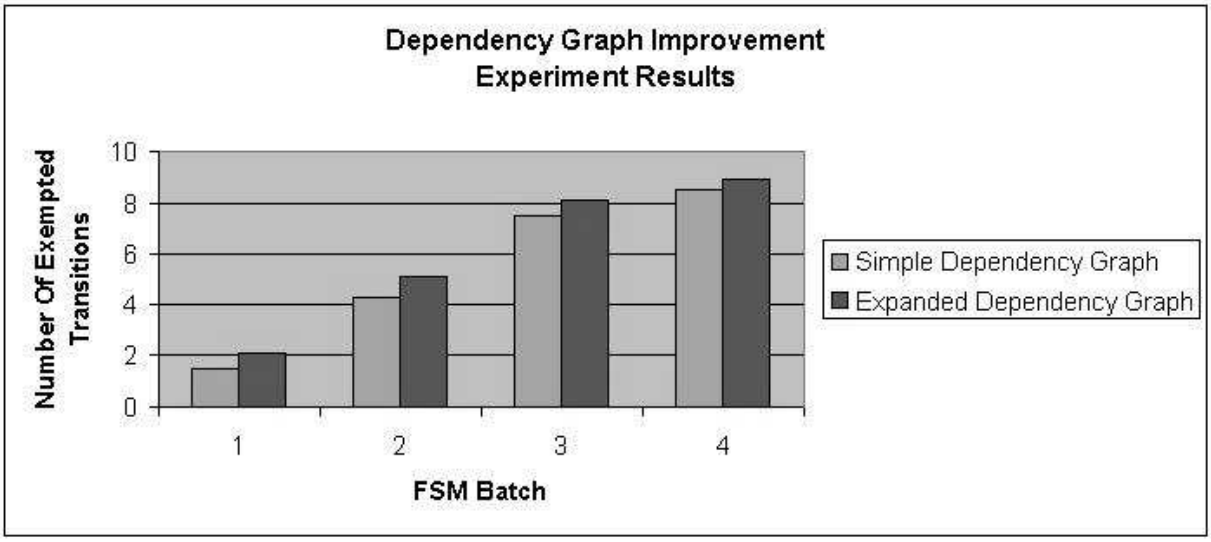


Figure 15: Average Number of Exempted Transition Tests

question from testing. However, satisfaction of all conditions in any single way of saving the transition is sufficient. Therefore there is a complicated and-or relationship between dependencies. Proper handling of this relationship merely improves the performance of the basic redundant transition test elimination scheme given in part 3.4. However, it is essential for the generalized transition test elimination scheme that we will discuss in part 4.3.

To show the practical gain of using an expanded dependency graph, we have included both ways of constructing dependency graphs in our experiments. The experiments were conducted on four seperate batches of randomly generated FSMs, batch 1 with 5 states, 2 input and output symbols each, batch 2 with 10 states, 3 input and output symbols each, batch 3 with 15 states, 4 input and output symbols each and batch 4 with 20 states, 5 input and output symbols each[6]. The average number of transitions that

[6]The experiment setup is given in more detail in part 5.1.

were exempted from explicit testing are given in figure 15.

4.2 R-recognition

The idea in this new way of state recognition is going backwards, where t-recognition consider going forward.

Suppose that:

- $(n_q, n_i, \bar{R})$ and $(n_j, n_k, \bar{R})$ are subpaths of $\bar{P}$
- n_i and n_k are d or t-recognized in $\bar{Q}$ as state s of M
- n_q is recognized in $\bar{Q}$ as state s' of M
- $\bar{R}$ is a nonconverging path, in other words is unique among sequences ending at state s. That is to say, for $\bar{R} = \bar{x}/\bar{y}$, $\forall s'' \in S, \lambda(s', \bar{x}) = \lambda(s'', \bar{x}) \wedge \delta(s', \bar{x}) = \delta(s'', \bar{x}) = s \Leftrightarrow s' = s''$.
- All transitions whose input labels are in the sequence $\bar{R}$ are verified

Then n_j is r-recognized in $\bar{Q}$ as s'.

R-recognition is equivalent to D and T-recognition for the purposes of transition verification. That is to say we can update our definition of state verification to use R-recognition as well as D and T-recognition.

The proof of this is fairly straightforward:

- Due to the assumption that the system under test has no more states than the specification FSM, and the existence of alpha sequences within the test sequence, we know that the states are verified in the system under test.

- Due to the assumption that the system under test is deterministic and it does not change, and the constraint that all transitions with their input labels in $\bar{R}$ are verified, all paths with the input portion of $\bar{R}$ are verified in the system under test.

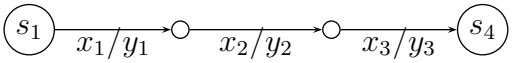
- For $\bar{R}$ is nonconverging in the specification FSM, and all paths with the input portion of $\bar{R}$ are verified in the system under test, $\bar{R}$ is also nonconverging in the system under test.

- For the endpoint of $\bar{R}$ is recognized as state s, $\bar{R}$ is unique among the paths with its input label in the system under test that end at state s, its starting state in the implementation is the state of the implementation that corresponds to state s'.

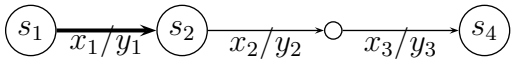
4.3 Generalized Redundant Transition Test Elimination

We can then identify edges that are verified through R-recognition within the α' sequences and exempt them from the transition tests. This idea is similar to the test exemption proposed in section 3.4 and the construction is very similar. The only difference being additional candidates for exemption and a complex dependency relationship between those.

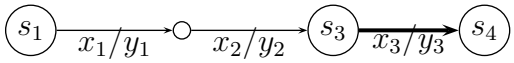
This generalization has been proposed first by Tekle, Ural, Yalcin and Yenigun[8]. There are three minor divergences here from the paper, one is relaxation of nonconverging edges condition to nonconverging paths that are enough to facilitate R-recognition, the expanded dependency graph that can



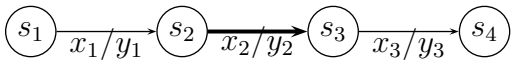
(a) A D-sequence has its endpoints d-recognized



(b) We can t-recognize nodes in the forward direction given that the involved transitions -$(s_1, s_2, x_1/y_1)$ in this example- are verified



(c) We can r-recognize nodes in the backward direction given that the related transitions (all transitions with input x_3 in this example) are verified and the label is unique among all labels of paths that end at the endpoint -that is to say $(s_i, s_4, x_3/y_3) \Rightarrow s_i = s_3$



(d) We can verify an edge whose endpoints we recognize in this manner -$(s_2, s_3, x_2/y_2)$ in this example-, hence avoid testing it elsewhere

Figure 16: Transition verification over D-sequences through use of R-recognition

be used to choose transitions to be exempted in practice, and as in other methods, replacement of a RCPT with an RCPP.

In the following, we will define a set of edges $L \subset E$ that need not be subjected to transition tests to be verified given some α' set A.

4.3.1 Identifying Redundant Transition Tests

Let $R_i = e_{i_1} e_{i_2} \dots e_{i_h}$ be the sequence of edges corresponding to application of $\bar{T}_i$ at state s_i. Let $e_{i_l} = (v_{i_l}, v_{j_l}, x_{i_l}/y_{i_l})$ be an edge in R_i. If

- $\forall r, 1 \leq r < l$, e_{i_r} is verified in $\bar{Q}$
- $e_{i_{l+1}} e_{i_{l+2}} \dots e_{i_h}$ is a nonconverging path
- $\forall r, l < r \leq h$, $\forall v \in V$ all edges $(v, v', x/y)$ such that $x = x_{i_r}$ are verified in $\bar{Q}$

e_{i_l} is verified in $\bar{Q}$.

Therefore we can pick edges that can be verified as a result of this condition and exempt them from transition tests. However, if exemption of an edge e depends on verification of another edge e' when exemption of e' depends on e, we cannot exclude both edges from transition tests. Note also that edges in a given R_i are dependent on each other, therefore at most one edge can be saved per state.

Also note that two paths R_i and R_j, corresponding to application of $\bar{T}_i$ at state s_i and $\bar{T}_j$ at state s_j respectively, may contain the same transition e. Transition e can be excluded from the transition tests if the conditions imposed by either one is satisfied. Therefore a straightforward dependency

graph as proposed in the paper is not effective in accurately representing the relationships between transition tests.

The following construction gives an accurate representation of the dependencies:

Let:

- $V_E = \{s_{ix} | \forall (s_i, s_j, x/y) \in E\}$ be a set of states representing the transitions of FSM M
- $V_S = \{s_n^k | \forall e_n^k\}$ be a set of states representing edges on all paths $R_i =$
- $V_D = V_E \cup V_S$ be the set of states for the dependency graph
- $E_R = \{(s_{i_j^k x_j^k}, s_k, \epsilon/\epsilon) | \forall t_n^k, 1 \leq j < n\}$ be the set representing the dependencies of an edge to edges before it
- Let $E'_R = \{(s_{i_j^k x}, s_k, \epsilon/\epsilon) | \forall t_n^k, \forall x, n < j \leq l\}$ where l is the length of DS_k be the set representing the dependencies of an edge to edge labels after it
- Let $E_A = \{(s_n^k, s_{i_n^k x_n^k}, \epsilon/\epsilon) | \forall t_n^k \textit{ such that } t_{n+1}^k \ldots t_l^k\} \textit{ is a nonconverging path}\}$ be the set representing exemption of a transition test given that dependencies of node s_n^k are satisfied
- $E_D = E_R \cup E'_R \cup E_A$
- Remove nodes from V_S until G_D is acyclic, while aiming to maximize cardinality of L
- $L = \{(s'_i, s_j, x/y) | \forall (s_k, s_i x, \epsilon/\epsilon) \in E_D\}$

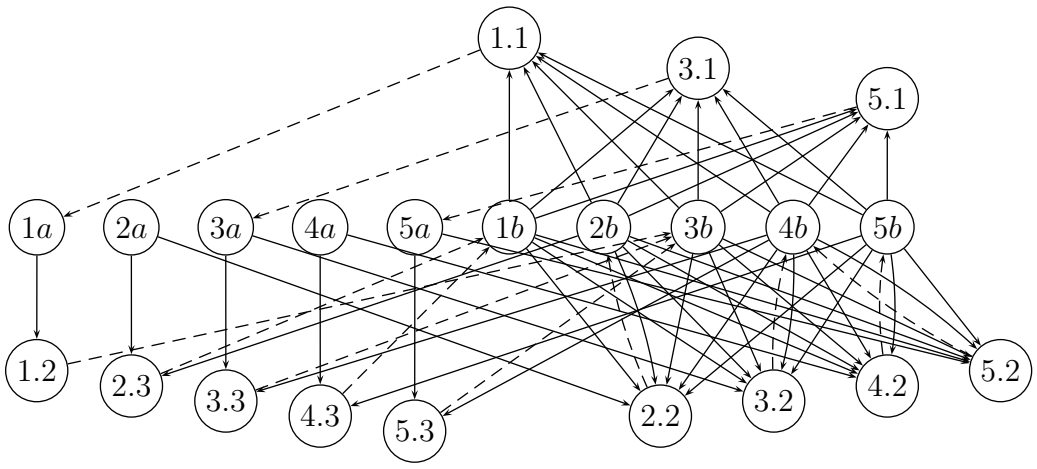


Figure 17: Full Dependency Graph of M_0 for Generalized Redundant Transition Test Elimination

Once G_D is constructed, all cycles on it have to be removed, through removal of nodes in V_S, which corresponds to refraining from using the corresponding $\bar{R}_i$ edge for exemption. Any acyclic subgraph of G_D thus constructed gives us a valid set L of transitions that can be excluded from tests. After cycle removal, all nodes in V_D with an incoming edge can be included in L. Therefore the cycle removal algorithm should aim to maximize the number of V_D elements with incoming transitions.

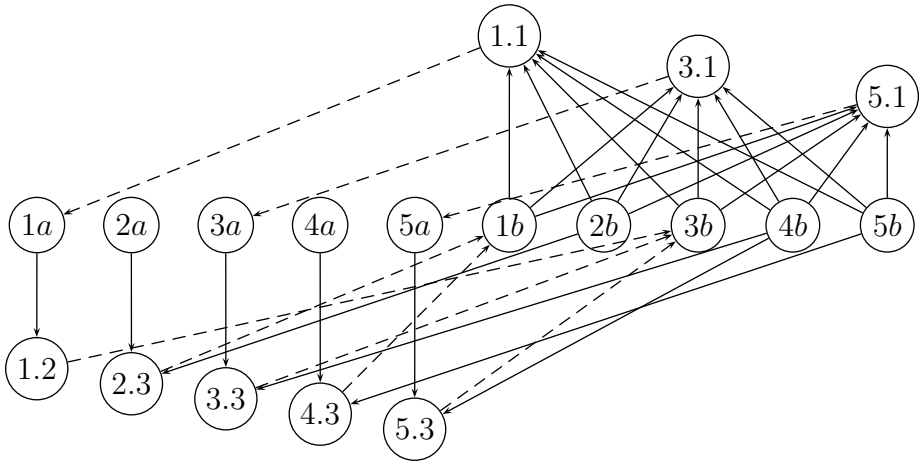
The maximal acyclic subgraph problem itself is NP-complete, so we do not aim for the best result in all cases. Also note that a maximal acyclic subgraph does not necessarily constitute as the best solution to this problem, as our main interest is preserving the maximum amount of V_E edges with incoming transitions. Moreover, this construction yields a very complex dependency graph, making a good selection of transition tests to be eliminated a difficult problem.

The dependency graph constructed according to this formulation for FSM M_0 is given in figure 17. An acyclic subgraph that can be constructed through removals from V_D is given in figure 4.3.1. The resulting graph is acyclic, therefore transitions $(s_1, s_3, a/0)$, $(s_1, s_4, b/0)$, $(s_3, s_4, a/1)$ and $(s_5, s_4, a/0)$ can be exempted from transition tests.

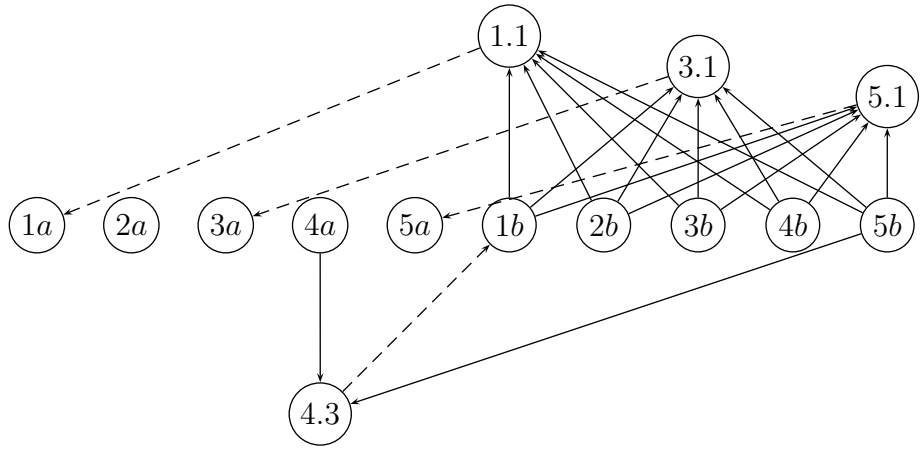
4.3.2 Checking Sequence Construction

The checking sequence construction method first constructs a digraph G' by augmenting $G = (V, E)$ representing a model FSM M where:

- $U' = \{s'_i | \forall s_i \in V\}$ representing the set of recognized versions of the states in V
- $V' = V \cup U'$ is the set of vertices for the augmented graph G'
- $E_C = \{(s'_i, s_j, x/y) | \forall (s_i, s_j, x/y) \in E\}$ representing edges in V, so that the transitions to be verified start at recognized versions of the states they belong
- $E_{\alpha'}$ representing the α' edges, which are used to D-recognize state s_i and move the system to the recognized state f'_j as well as enable the T-recognition of final nodes of T-sequences
- $E_T = \{(s_i, f'_i; l_i)) | \forall s_i \in V\}$ representing $\bar{T}_i$s, which are used to D-recognize state s_i and move the system to the recognized state f'_i
- $E'' \subset \{(s'_i, s'_j, x/y) | \forall (s_i, s_j, x/y) \in E\}$ such that $G'' = (V', E'')$ does not have a tour, and $G' = (V', E')$ is strongly connected



(a) Dependency Graph of M_0 After Removal of 2-node Cycles



(b) Dependency Graph of M_0 After Cycle Removal

Figure 18: Dependency Graph for Generalized Redundant Transition Test Elimination

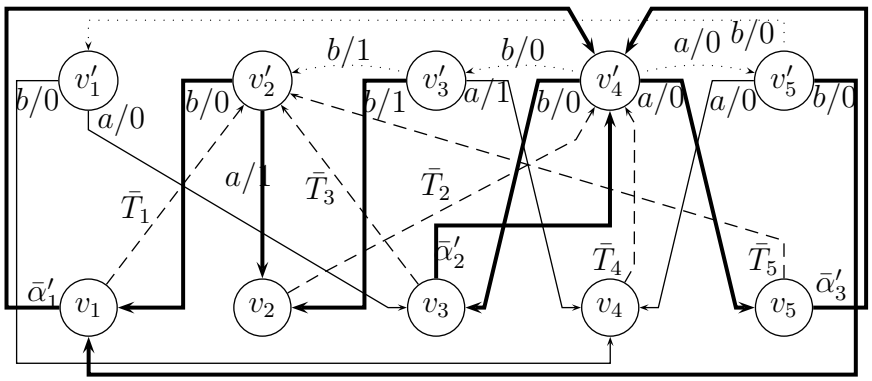


Figure 19: $G' = (V', E')$ for M_0, constructed according to Tekle, Ural, Yalcin, Yenigun '05

- Pick a subset $L' \subset L$ such that after setting $E' = E' - (s'_i, s'_j, x/y) | (s'_i, s_j, x/y) \in L$, a valid RCPP on G' can still be found[7]
- Set the set of connecting transition candidates $E'' = E'' - (s'_i, s'_j, x/y) | (s'_i, s_j, x/y) \in L'$
- Set the set of necessary transitions tests $E'_C = E_C - L'$
- Create $E' = E_C \cup E_T \cup E_{\alpha'} \cup E''$
- $G' = (V', E')$ is the augmented graph to be used in construction of the checking sequence
- $E_{imp} = E'_C \cup E_{\alpha'}$ are the edges that have to be included in the rural path on G'

A graph constructed accordingly for FSM M_0 is given in figure 19. The nodes in V and U' are at the bottom, and at the top respectively. The

[7] A sufficient condition is maintaining strong connectivity, but it is not always necessary.

dashed lines are the edges in E_T, and the dotted lines are the edges in E''. The edges in $E_{\alpha'} \cup E_C$ are given in solid lines. The bold solid lines are the edges in $E_{\alpha'} \cup E'_C$, and the remaining solid lines are the edges in L. Once G' is thus constructed, a RCPP over the noted edges that starts at the vertex corresponding to the initial state of the specification automaton and ends in a recognized state can be found. The label of such a path is a checking sequence for the automaton.

Again, it is proposed in the paper that a tour concatenated with a Tsequence should be found. This is a sufficient condition, but a tour is not necessary. As before, a path that ends in a recognized state is sufficient.

A path over G' containing all edges in $E'_C \cup E_{\alpha'}$ is:

$$(v_1, v'_4, \alpha'_1), (v'_4, v_5, a/0), (v_5, \alpha'_3), (v'_4, v_3, b/0), (v_3, v'_4, \alpha'_2)$$
$$(v'_4, v'_5, a/0), (v'_5, v_1, b/0), (v_1, v'_2, \bar{T}_1), (v'_2, v_2, a/1), (v_2, v'_4, \bar{T}_2),$$
$$(v'_4, v'_3, b/0), (v'_3, v_2, b/1), (v_2, v'_4, \bar{T}_2), (v'_4, v'_3, b/0), (v'_3, v'_2, b/1)$$
$$(v'_2, v_1, b/0), (v_1, v'_2, \bar{T}_1)$$

whose label is a checking sequence of length 43.

4.4 Introducing UIOs

UIO sequences, described in 2.3.1, can be used to generate checking sequences in the absence of distinguishing sequences, however if distinguishing sequences exist, there are methods that can yield much shorter checking sequences. But UIO sequences can be incorporated into these methods as well,

replacing distinguishing sequences where applicable to shorten the length of the overall sequence[11].

4.4.1 Test Segments

The test segments are the same as 3.2.1, save the case in which they can be tested using UIO sequences. For transitions that can be tested using UIO sequences, we represent the option to do so in the graph using additional nodes.

A UIO sequence can be used for transition tests if all input symbols in it are verified. Therefore edges whose input symbols occur in active UIO sequences cannot be tested using UIO sequences in order to avoid cyclic dependencies. Consequently, a set of symbols S has to be selected, which will denote the input symbols that can be verified using UIO sequences, but may not occur within them.

The optimal selection of the set S is important for the efficiency of the proposed method, however, it is not a simple problem. We are using a simulated annealing method that aims to maximize the number of transitions that can be verified for our implementations, but the algorithm is yet to be finalized.

4.4.2 Choosing UIOs to Use

There is no trivial way of choosing the set of symbols S. One way that we propose that this can be done is by some sort of simulated annealing. The procedure is as follows:

- Set $S = \Sigma$

- Begin loop: $\forall x \in S$ let $S_x = S - \{x\}$
  - Let U_i denote the set of UIO sequences of state s_i that contain input symbols from S only. Let l be the length of the shortest UIO in U_i.
  - Let U_i^x be the set of UIO sequences of state s_i that contain input symbols from S_x only. Let l_x be the length of the shortest UIO in U_i^x.
  - Let T_i denote the set of transitions whose tail is state s_i and input symbols not in S. Let n be the cardinality of T_i.
  - Let T_i^x denote the set of transitions whose tail is state s_i and input symbols not in S_x. Let n_x be the cardinality of T_i^x.
- If $n \times l < n_x \times l_x, \forall x \in S$, current S is a local minimum, and is our final selection
- Otherwise, set $S = S_x$ for the x value that yields the minimum $n_x \times l_x$ and reiterate the loop

Once the set S is chosen, construction of the augmented graph is a straightforward process, a lot like the methods we have considered so far. The only addition is the nodes added to handle transitions that can be verified using UIO sequences.

4.4.3 Checking Sequence Construction

The checking sequence construction method first constructs a digraph G' by augmenting $G = (V, E)$ representing a model FSM M where:

- S a set of input symbols, which stand for the input symbols of the transitions that may be tested using UIO sequences
- $U' = \{s'_i | \forall s_i \in V\}$ representing the set of recognized versions of the states in V
- $V^U = \{s_i^U | \forall s_i \in V\}$ is the set of vertices that can be recognized through use of UIO sequences
- $V' = V \cup U' \cup V^U$ is the set of vertices for the augmented graph G'
- $E_C = \{(s'_i, s_j, x/y) | \forall (s_i, s_j, x/y) \in E, x \notin S\}$ representing edges in V, edges that have to be tested using distinguishing sequences
- $E_C^U = \{(s'_i, s_j^U, x/y) | \forall (s_i, s_j, x/y) \in E, x \in S\}$ representing edges in V, edges that may be tested using UIO sequences
- $E_\epsilon^U = \{(s_i^U, s_i, x/y) | \forall s_i \in V\}$ enabling the transitions that can be verified using UIO sequences to be verified by distinguishing sequences, if doing so would yield a shorter checking sequence
- $E_{\alpha'}$ representing the α' edges, which are used to D-recognize state s_i and move the system to the recognized state f'_j as well as enable the T-recognition of final nodes of T-sequences
- $E_T = \{(s_i, f'_i, l_i)) | \forall s_i \in V\}$ representing $\bar{T}_i$s, which are used to D-recognize state s_i and move the system to the recognized state f'_i
- $E_T^U = \{(s_i, \delta(s_i, UIO_i^k), \lambda(s_i, UIO_i^k)) | \forall s_i \in V, UIO_i^k \; s.t. \; UIO_i^k \; is \; a \; UIO \; for \; state \; s_i \; and \; does$ set of UIO sequences that can be used for state recognition

- $E'' \subset \{(s'_i, s'_j, x/y) | \forall (s_i, s_j, x/y) \in E\}$ such that $G'' = (V', E'')$ does not have a tour, and $G' = (V', E')$ is strongly connected
- Create $E' = E_C \cup E_T \cup E_{\alpha'} \cup E''$
- $G' = (V', E')$ is the augmented graph to be used in construction of the checking sequence
- $E_{imp} = E_C \cup E_C^U \cup E_{\alpha'}$ are the edges that have to be included in the rural path on G'

A graph constructed accordingly for FSM M_0 is given in figure 20, nodes from V_U that are unused are removed. Once G' is thus constructed, a RCPP over the noted edges that starts at the vertex corresponding to the initial state of the specification automaton and ends in a recognized state can be found. The label of such a path is a checking sequence for the automaton.

A path over G' containing all edges in $E_C \cup E_C^U \cup E_{\alpha'}$ is:

$$(v_1, v'_4, \alpha'_1), (v'_4, v_5, a/0), (v_5, v'_4, \alpha'_3), (v'_4, v_3^U, b/0), (v_3^U, v'_5, \bar{U}_3)$$

$$(v'_5, v_4, a/1), (v_4, v'_4, \bar{T}_4), (v'_4, v'_5, a/0), (v'_5, v_1, b/0), (v_1, v'_2, \bar{T}_1),$$

$$(v'_2, v_1, b/0), (v_1, v'_2, \bar{T}_2), (v'_2, v_2, a/1), (v_2, v'_4, \bar{T}_2), (v'_4, v'_3, b/0),$$

$$(v'_3, v_4, a/1), (v_4, v'_4, \bar{T}_4), (v'_4, v'_3, b/0), (v'_3, v_2^U, b/1), (v_2^U, v'_2, \bar{U}_2)$$

$$(v'_2, v'_1, b/0), (v'_1, v_4, b/0), (v_4, v'_4, \bar{T}_4), (v'_4, v'_5, a/0), (v'_5, v'_1, b/0)$$

$$(v'_1, v_3, a/0), (v_3, v'_4, \alpha'_2)$$

whose label is a checking sequence of length 59, which is an improvement of 4 over what could be generated without using UIO sequences.

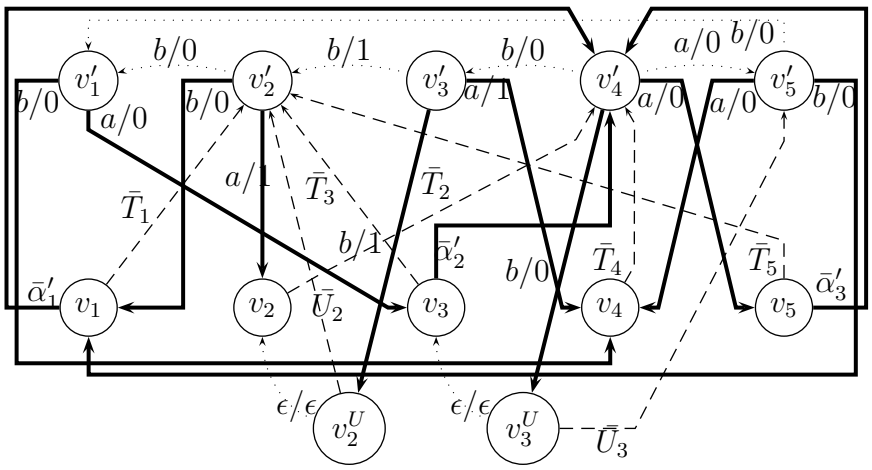


Figure 20: $G' = (V', E')$ for M_0 implementing UIO sequences.

4.4.4 Further Improvements

We can also incorporate UIO sequences in a redundant transition elimination scheme. If all transitions with a given input label are verified, then that input label is enabled to be used within UIO sequences. Again, cyclic dependencies need to be avoided, therefore the selection of input characters that can be verified using UIO sequences need to be integrated to the dependency graph of transition verifications somehow, and this is an open problem.

However, an example integration of UIO sequences and redundant transition elimination on M_0 is given in figure 21. This construction reduces the length of resulting checking sequence by a further 3. The path, which has a label of length 50, is as follows:

$$(v_1, v'_4, \alpha'_1), (v'_4, v_5, a/0), (v_5, v'_4, \alpha'_3), (v'_4, v_3^U, b/0), (v_3^U, v'_5, \bar{U}_3)$$

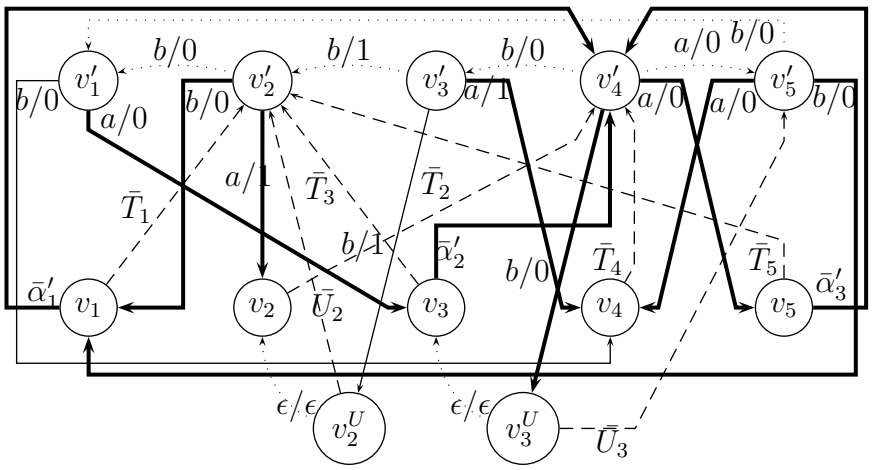


Figure 21: $G' = (V', E')$ for M_0. Implementing both redundant transition test elimination and UIO sequences.

$$(v_5', v_4, a/1), (v_4, v_4', \bar{T}_4), (v_4', v_5', a/0), (v_5', v_1, b/0), (v_1, v_2', \bar{T}_1),$$

$$(v_2', v_1, b/0), (v_1, v_2', \bar{T}_2), (v_2', v_2, a/1), (v_2, v_4', \bar{T}_2), (v_4', v_3', b/0),$$

$$(v_3', v_4, a/1), (v_4, v_4', \bar{T}_4), (v_4', v_5', a/0), (v_5', v_1', b/0), (v_1', v_3, a/0),$$

$$(v_3, v_2', \alpha_2')$$

It is possible to integrate use of UIOs with generalized redundant transition test elimination as well, however, M_0 is an inadequate example to demonstrate any improvement.

5 Experiments

5.1 Experiment Setup

Our implementation of the methods presented is done on Java, and uses the optimization tool AMPL to solve the integer programming problem for symmetricity augmentation.

We worked on randomly generated, strongly connected, completely specified, deterministic FSMs that have distinguishing sequences. We used full-length, preset DSs, that is to say we did not replace DS with a prefix when possible as was the case in the examples so far.

The data sets we worked on consisted of FSMs with 5 states, 2 input and 2 output symbols, 10 states, 3 input and 3 output symbols, 15 states, 4 input and 4 output symbols and 20 states, 5 input and 5 output symbols respectively, each set containing 15 FSMs. The sets are labeled batch 1, 2, 3 and 4 respectively in the graph given in figure 22. The tool can handle FSMs with greater complexity, however our FSM generator could not produce FSMs with greater number of states that satisfied the conditions.

In each case, we followed the steps outlined in the specifications, and extracted labels of the rural paths that were extracted after the augmentation. Our focus is the lengths of resulting checking sequences. A full layout of all results can be observed in table 1. A comparison of average lengths is given in figure 22.

(a) 5 States, 2 I/O

5_2_2	Basic	Optimized	Over. Elim	Red. Elim.	G. R. Elim.
FSM_1	90	74	42	67	63
FSM_2	74	61	35	56	61
FSM_3	83	66	66	62	52
FSM_4	97	74	74	61	67
FSM_5	69	62	62	57	53
FSM_6	90	81	81	73	73
FSM_7	85	68	68	62	62
FSM_8	83	76	40	71	71
FSM_9	109	66	38	62	58
FSM_10	89	68	68	56	58
FSM_11	97	80	80	74	74
FSM_12	104	63	36	63	55
FSM_13	67	60	35	60	56
FSM_14	114	91	91	85	90
FSM_15	93	74	74	68	64
	89,6	70,93333	59,33333	65,133333	63,8

(b) 10 States, 3 I/O

10_3_3	Basic	Optimized	Over. Elim	Red. Elim.	G. R. Elim.
FSM_1	310	213	213	178	180
FSM_2	338	232	232	212	205
FSM_3	234	191	136	185	182
FSM_4	202	173	116	167	163
FSM_5	320	228	228	194	203
FSM_6	290	232	232	195	226
FSM_7	269	215	215	211	205
FSM_8	260	214	214	199	197
FSM_9	230	176	176	157	152
FSM_10	317	271	271	253	247
FSM_11	448	246	246	204	226
FSM_12	227	183	183	170	167
FSM_13	310	248	167	227	219
FSM_14	260	189	189	178	166
FSM_15	301	185	185	169	174
	287,7333	213,0667	200,2	193,2667	194,13333

(c) 15 States, 4 I/O

15_4_4	Basic	Optimized	Over. Elim	Red. Elim.	G. R. Elim.
FSM_1	475	408	408	398	394
FSM_2	514	403	403	381	383
FSM_3	539	344	344	312	319
FSM_4	643	425	425	389	392
FSM_5	518	367	367	332	348
FSM_6	457	394	394	357	380
FSM_7	657	500	500	463	473
FSM_8	548	436	436	381	388
FSM_9	482	373	373	348	336
FSM_10	526	372	278	348	340
FSM_11	513	362	270	341	342
FSM_12	496	339	339	298	293
FSM_13	476	356	356	303	312
FSM_14	604	518	518	479	499
FSM_15	558	427	427	386	408
	533,7333	401,6	389,2	367,73333	373,8

(d) 20 States, 5 I/O

20_5_5	Basic	Optimized	Over. Elim	Red. Elim.	G. R. Elim.
FSM_1	940	666	528	651	632
FSM_2	974	578	578	534	514
FSM_3	905	581	581	531	532
FSM_4	1181	765	765	727	727
FSM_5	881	587	587	550	544
FSM_6	917	603	603	560	560
FSM_7	954	720	720	655	655
FSM_8	702	597	597	538	552
FSM_9	1227	881	881	838	835
FSM_10	902	695	695	666	663
FSM_11	796	611	611	582	591
FSM_12	1063	830	830	773	778
FSM_13	1104	798	798	754	754
FSM_14	823	595	595	543	543
FSM_15	1135	842	842	816	810
	966,9333	689,9333	680,7333	647,8667	646

Table 1: Checking Sequence Length for Each FSM Used in the Experiment

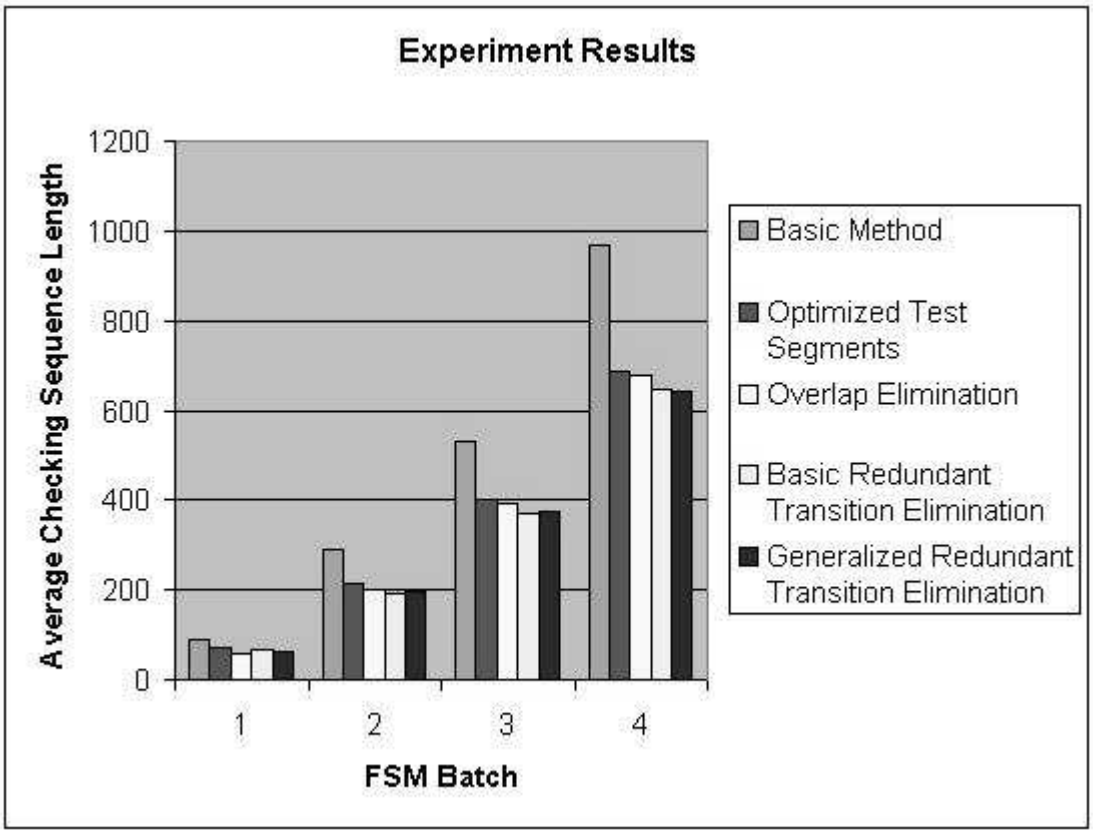


Figure 22: Average Checking Sequence Lengths

5.2 Experiment Results

The basic case[10] is included in the experiments for comparison purposes, and it was not expected for it to perform as well as the other methods. Optimization of the test segments[4] provides a noticable improvement over the basic case. The remaining methods improve upon this one, therefore they behave the same in the worst case.

Overlap elimination[9] invariably performs the best among all methods whenever distinguishing sequence is composed of a single input character, but otherwise performs exactly the same as the previous method. Considering the facts that the complexity in practical applications will be high, and the optimization problem that needs to be solved is far greater than that in the other methods due to the complexity of the augmented graph, this method does not seem to be promising.

Redundant transition test elimination[2], on the other hand performs well and scales complexity better than previous methods, reliably providing about 10% reduction in checking sequence length over the method using optimized test segments. Building upon this method seems reasonable, considering that it does not increase the complexity of the optimization problem.

Generalized redundant transition test elimination[8] performs well, yielding the shortest checking sequences in 5 and 20 state FSM batches, and would perform better in cases where distinguishing sequences are long in comparison to the number of states of the FSM. Theorethically it should never have performed worse than basic redundant transition test elimination, due to the fact that any choice the latter can make is viable for the former. Handling the complex dependency graph that is formed during the operation of gener-

alized case is far more difficult in comparison to the simple one generated in the basic case, and hence there is still room for improvement in the selection of transitions to be exempted. With corrections to that part of the algorithm we use, we expect to significantly surpass the efficiency of the basic case. Even in its current state, the method is performing adequately, and we expect it to allow more savings as DS length increases, therefore work better with on larger FSMs.

6 Conclusion

Finite state systems have, since early days of computer science, been a useful tool of representing systems in a variety of fields, and study of their properties have proved useful for testing and verification of real life systems.

When applicable, it is often desirable to use checking sequences to test systems. Although they do not pinpoint the error, which is the more complex problem of fault identification, which extends into the domain of reverse engineering, a checking sequences guarantee detection of errors on systems of up to the size of specification, and even when such a guarantee cannot be attained, provide a structured way of building meaningful test cases.

We have provided outlines of several methods used in construction of checking sequences that are detailed enough to convert into implementations. In addition, we compared their efficiency using our implementations according to the given specifications, and compared their performances and also outlined possible methods of improvement for the methods in hand.

Checking sequence construction methods we currently have deal with sufficient conditions and we do not know how close these are to necessary conditions. However, the amount of redundancy we observe in them show that there is yet room for improvement. More methods for state recognition, transition verification and better ways to combine them, as we have been working to build, can increase the feasability of checking sequences and allow them to be used in a broader scope.

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. *IEEE Transactions on Communications*, 39(11):1604–1615, 1991.

[2] J. Chen, R.M. Hierons, H. Ural, and H. Yenigun. Eliminating Redundant Tests in a Checking Sequence. *Proc. TESTCOM*, pages 146–158, 2005.

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

[4] RM Hierons and H. Ural. Optimizing the Length of Checking Sequences. *Computers, IEEE Transactions on Computers*, 55(5):618–629, 2006.

[5] Robert M. Hierons and Hasan Ural. Reduced length checking sequences. *IEEE Trans. Comput.*, 51(9):1111–1117, 2002.

[6] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[7] E.F. Moore. Gedanken-experiments on sequential machines. *Automata Studies*, 34:129–153, 1956.

[8] K.T. Tekle, H. Ural, M.C. Yalcin, and H. Yenigun. Generalizing redundancy elimination in checking sequences. *Lecture notes in computer science*, pages 915–926.

[9] H. Ural and F. Zhang. An Optimization Model for Generating Checking Sequences with Overlapping.

[10] Hasan Ural, Xiaolin Wu, and Fan Zhang. On minimizing the lengths of checking sequences. *IEEE Trans. Comput.*, 46(1):93–99, 1997.

[11] M. Cihan Yalçın and Hüsnü Yenigün. Using distinguishing and uio sequences together in a checking sequence. In M. Ümit Uyar, Ali Y. Duale, and Mariusz A. Fecko, editors, *TestCom*, volume 3964 of *Lecture Notes in Computer Science*, pages 259–273. Springer, 2006.