INCREMENTAL AND HEURISTIC ALGORITHMS FOR DERIVING ADAPTIVE

DISTINGUISHING TEST CASES FOR NONDETERMINISTIC FINITE STATE

MACHINES

by

Ayat Saleh

A Thesis presented to the Faculty of the
American University of Sharjah
College of Engineering
In Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in
Computer Engineering

Sharjah, United Arab Emirates

September 2017

Approval Signatures

We, the undersigned, approve the Master's Thesis of Ayat Saleh.

Thesis Title: Incremental and Heuristic Algorithms for Deriving Adaptive Distinguishing Test Cases for Nondeterministic Finite State Machines.

Signature   **Date of Signature**
                                                                                               (dd/mm/yyyy)

_____   _____
Dr. Khaled El-Fakih
Associate Professor, Department of Computer Science and Engineering
Thesis Advisor

_____   _____
Dr. Rana Ahmed
Associate Professor, Department of Computer Science and Engineering
Thesis Committee Member

_____   _____
Dr. Usman Tariq
Assistant Professor, Department of Electrical Engineering
Thesis Committee Member

_____   _____
Dr. Fadi Aloul
Head, Department of Computer Science and Engineering

_____   _____
Dr. Ghaleb Husseini
Associate Dean for Graduate Affairs and Research
College of Engineering

_____   _____
Dr. Richard Schoephoerster
Dean, College of Engineering

_____   _____
Dr. Mohamed El-Tarhuni
Vice Provost for Graduate Studies

Acknowledgements

Abstract

Many methods are proposed for the construction of distinguishing test cases (DTCs) based on a specification given in the form of a Finite State Machine (FSM). In FSM-based testing, we have a black-box FSM Implementation Under Test (IUT) about which we lack some information, and we want to conclude this information by using the applied input sequences of DTCs to the IUT, then by observing the output responses to the applied input sequences final conclusions about the IUT are drawn. A DTC is adaptive if the next input of a DTC is selected based on the previously observed outputs. In this thesis, we propose an incremental approach, called Inc, for the construction of an adaptive DTC for a given set of states of a nondeterministic FSM. In addition, two heuristics are proposed for the derivation of adaptive DTCs. The first heuristic, called H, uses depth first search for a given fixed height while appropriately utilizing hashing to speed up the search for a DTC. The second heuristic, called Hc, is similar to the first; however, it uses a cost function for ordering the inputs to be considered while conducting the search. Comprehensive experiments are conducted, using both real and randomly generated FSMs, to assess the existence of DTCs and compare the performance of the proposed approaches. According to these experiments, in terms of execution time, Inc usually outperforms an existing non-incremental algorithm, called A, when a DTC does not exist. However, in contrary to the H and Hc methods, both A and Inc do not scale well for large size FSMs. Both H and Hc have comparable execution time; however, for large size FSMs, in terms of quality of obtained solutions (length of obtained DTC), usually Hc outperforms H.

Search Terms: *Model Based Testing, Distinguishing Test case, Incremental Algorithms, Heuristics.*

Table of Contents

List of Figures

List of Tables

9

List of Abbreviations

A	The (Exact) Adaptive Algorithm
ADS	Adaptive Distinguishing Sequence
b	Subset of States of S that we want to distinguish
CPU	Central Processing Unit
DTC	Distinguishing Test Case
DFS	Depth First Search
FSM	Finite State Machine
H	Heuristic Algorithm
Hc	Heuristic Algorithm with Cost Function
I	Set of Inputs of an FSM
Inc	Incremental Algorithm
Inc(k)	Incremental Algorithm(k), $k = 2,3,4,8$
IUT	Implementation Under Testing
MBT	Model Based Testing
O	Set of Outputs of an FSM
QA	Quality Assurance
S	Set of States of an FSM
SDL	Standard Description Language
UML	Unified Modeling Language

Chapter 1 : Introduction and Related Work

1.1 Introduction

As a result of continuous advancements in computer technology, systems have become larger so they can fulfill more demanding and complicated tasks. Consequently, systems become vulnerable, and less reliable [1]. This makes software testing an important part of software development. By 1968, the first conference was held which had the goal of "the establishment and use of sound engineering principles in order to obtain reliable, efficient and economically viable software" [2]. This conference came up with the conclusion that applying quality assurance to products, process, components, and materials is so important. However, quality assurance and testing is a critical and an expensive process. In 1970, quality assurance of any typical programming project could cost more than 50 percent of the total cost; it also could take 50 percent of the whole time spent on that project. For the time being, the time and cost of software testing are the same [3, 4].

Model-Based Testing (MBT) is the most famous approach for reducing the cost and time of testing. Prominent state based models include (Mealy) Finite State Machines (FSMs), labeled transition systems, finite state automata. In this thesis, we consider systems modeled as FSMs. An FSM consists of states, inputs, outputs, and transitions between states each labeled by an input/output pair. FSMs are the underlying models for essential description techniques, such as the Standard Description Language (SDL), Statecharts, and the Unified Modeling Language (UML) [5].

Test derivation from FSMs provides a rigor approach for functional testing of interactive systems and communication protocols [6, 7, 8], web services [9-13], software design [14], sequential circuits [15], lexical analysis [16], graphical user interfaces [17], object oriented systems [18, 19], embedded systems [20, 21], and industrial projects [22].

A fundamental FSM-based testing problem deals with the derivation of *distinguishing sequences* that can identify the initial state of an FSM describing the behavior of a black-box Implementation Under Test (IUT) [7, 8, 14]. An input sequence is *adaptive* if the selection of the next input to be applied to an IUT is based

on the observed outputs of the IUT to the previous inputs, and an input sequence is *preset* if it is a single input sequence that is fixed before performing the experiment. An adaptive distinguishing sequence is usually realized from a special FSM, which can be thought as a rooted decision tree, called a *distinguishing test case* (DTC) or an *adaptive distinguishing sequence* (ADS).

In fact, since the seminal paper by Moore [23], the distinguishability problem is still under further investigation for various classes of FSMs. For more detailed information, the reader may refer to the related studies and textbooks reported by Alur et al. [24], Dorofeeva et al. [22], Gill [25], Güniçen et al. [26], Güniçen et al. [27], Hierons and Türker [28], Kohavi [29], Kushik et al. [30, 31], Lee and Yannakakis [32], Mathur [33], Spitsyna et al. [34], Türker and Yenigün [35], and Türker et al. [36].

We note that well-studied classes of FSMs include *complete* or *partial* FSMs depending upon whether there exists an outgoing transition under each input at each state or not; *deterministic* if at each state under each input there is at most a single outgoing transition under the input or *nondeterministic* if at some state(s) there exists many outgoing transitions under the input. A nondeterministic FSM is *observable* if at for every state found in the machine has maximum one transition for every input/output pair; else, the FSM is called *non-observable.* Almost all the studies focused on observable FSMs because any non-observable specification FSM can be transformed into an observable FSM with the same behavior. In this thesis, we consider complete observable nondeterministic FSMs. We note that non-determinism occurs due to various reasons such as performance, flexibility, limited controllability, and/or abstraction [24, 37, 38].

In general, as mentioned above, the distinguishability problem is studied assuming either a preset or adaptive testing mode. In the former, a distinguishing sequence is derived in advance and all the inputs of the distinguishing sequence are applied to the given IUT FSM while in the latter, the selection of the next input to apply is based on the observed outputs to the previous inputs. Accordingly, an adaptive experiment is usually represented by a special FSM with a tree structure that is usually called a DTC.

Contribution: In this thesis, we target the problem of deriving a distinguishing test case DTC for a set of states of a (complete and observable) nondeterministic FSM. In [39], a procedure, called A, is provided for checking the existence of a distinguishing test case for a set of states of an FSM. In particular, given an FSM S and a set b of several states, based on some established construction rules, a special distinguishing machine $S_{b\text{-}dist}$ that includes all distinguishing test cases (if any) for the set b is derived. Then, using the FSM $S_{b\text{-}dist}$, the procedure returns a message indicating that the set b has no DTC (i.e., the set is not adaptively distinguishing), or a message indicating that the set is adaptively distinguishing (i.e. there exists a DTC for the set). If a DTC exists for the set b, then a simple procedure for constructing a DTC is provided. As the complexity of deriving the distinguishing machine $S_{b\text{-}dist}$ significantly depends on the cardinality of the subset b, in this thesis, we propose an incremental approach (strategy), called Inc, for checking the existence of a DTC for a given set b of several states of an FSM. Instead of considering all the states of b at once, as done in [39], we start by partitioning the set b into disjoint subsets of states. Then, given a subset c of the partition, a distinguishing machine $S_{c\text{-}dist}$ is derived for c. If there is no DTC for subset c, then the algorithm directly terminates concluding that the given set b has no DTC. Otherwise, another subset c of the partition is incrementally added to the so-far selected subsets. That is, again a distinguishing machine for c is derived that is composed of the already obtained composition of the distinguishing machines for the previously considered subsets. The process is repeated till the procedure terminates at some iteration as there is no DTC for a current subset of the partition, or the procedure terminates declaring that the given set of states b has a DTC. It should be noted that basically, in order to be able to incrementally consider subsets (of the partition) of states for checking the existence of a DTC, a commutative and associative composition operator is proposed over already constructed machines $S_{c\text{-}dist}$ which represent corresponding sets of test cases. We experiment with the algorithm while initially considering two states of the subset of b, then incrementally considering the other states of b one after the other. However, we also experiment with different partitions of the set b, namely, we study the effect splitting b into two, three, four, and eight disjoint subsets of the same cardinality. A lot of experiments are conducted to assess the performance of A and the incremental implementations.

In addition, as the A [39] and Inc algorithms do not scale well in terms of execution time for large size machines. In the second part of this thesis, two heuristic approaches for the derivation of a DTC for a given set b are proposed. The first approach, called H, is based on traversing the truncated successor tree, used for constructing a DTC, using Depth First Search (DFS) for the given fixed height L while appropriately utilizing hashing to speed up the search process. The second approach, called Hc, is the same as the first; however, it uses a cost function for ordering the inputs to be considered in while conducting the search. A comprehensive assessment of H and Hc implementations in terms of execution time and quality of obtained solutions (height of obtained DTC) is provided. In theory, H (and Hc) may not provide a solution while a solution exists as its search is restricted by fixed height L; accordingly, L is set appropriately such that H (Hc) always find a solution for the considered experiments. A detailed summary of the obtained experiments is provided in the Conclusion section.

1.2 Related Work

"Gedanken experiments" paper by Moore was the first step in research on distinguishing experiments for deterministic FSMs [23]. More surveys about FSM experiments and related algorithms can be found in [30-32, 36].

Previous research on distinguishability focuses on two aspects, *(i)* establishing the theoretic upper bound on the height of a DTC (i.e. the complexity); *height* of a DTC means the maximum length that can input sequence reach during an experiment. Height of the experiment is usually used for representing how much the experiment is complex. Finding the tight upper bound is an important constituent of any intended experiment. *(ii)* Other research on distinguishability, as the one considered in this thesis, focuses on providing procedures for the derivation of DTCs.

Gill [25] and Lee and Yannakakis [32] presented methods for deriving adaptive distinguishing experiments with the evaluation of the height of these experiments. Kohavi [29] showed that the upper bound on the height of a preset experiment is exponential with respect to the number of states of a given machine, while Sokolovskii, as reported in [40], has shown that such an upper bound for adaptive distinguishing experiments is polynomial. For deterministic FSMs El-Fakih

et al. [41] presented heuristics, including a genetic algorithm, for deriving preset distinguishing sequences for observable nondeterministic FSMs.

For research related to adaptive experiments with complete observable nondeterministic specifications, Alur et al. [24] proved that for two states for a complete observable machine that has n states, the shortest length is maximum $n(n-1)/2$. When $n > 2$ states, Kushik et al. [31] showed that $2^n - n - 1$ is the maximum height or (upper bound of the height) of any a shortest DTC. El-Fakih et al. [31, 39] showed that this exponential bound is tight.

On the derivation of DTCs, a method is presented in [39] for checking the existence of DTC; and if it exists then a DTC is derived. The work presented in this thesis extends previous work in [39] by introducing and assessing incremental approach for deriving DTCs. In addition, in this thesis, a heuristic approach is proposed and assessed for efficient deriving DTCs considering very large size FSMs. It should be noted that recently, Türker et al. [36] presented and assessed effective algorithms for constructing minimum cost DTCs, but they were for deterministic FSMs

In addition, it is worth mentioning that there is lately a new research direction in solving distinguishability problems. Some work has been proposed for using state-of-the-art parallel technologies for efficiently solving some distinguishability problems. For instance, Hierons and Türker [42] presented parallel algorithms for deriving DTCs for observable nondeterministic FSMs using Graphical Processing Units (GPU) and El-Fakih et al. [43] proposed two GPU based implementations and a Network-of-Workstations based implementation for the derivation of all the successors of all pairs of states of an observable nondeterministic FSM. This is done in order to reduce the time and efforts of deriving sequences for distinguishing (states of) nondeterministic FSMs.

In summary, in this thesis, adaptive distinguishing experiments for nondeterministic FSMs are studied. The main objective is to reduce the execution time of deriving an experiment, as execution time increases drastically as the size of the FSM increases (i.e., the number of transitions of a machine). The work presented in this thesis extends previous work in [39] by introducing and assessing incremental

approach for deriving DTCs. In addition, in this thesis, two heuristic approach are proposed and assessed for efficient deriving DTCs considering very large size FSMs.

This thesis is organized as follows; Chapter 2 discusses the notions and definitions used in the thesis and it also includes the exact algorithm [39]. Chapter 3 explains the incremental approach for deriving a distinguishing test case. It also discusses the composition operator used by the incremental algorithm. Chapter 4 includes the proposed two heuristic algorithms H and Hc. Chapter 5 includes experimental evaluation of the proposed work and Chapter 6 concludes this thesis and includes a detailed summary of the obtained results.

Chapter 2 : Preliminaries

In this chapter, the notions and terminologies used throughout the thesis. In addition, the exact algorithm, called A, given in [39], for checking the existence and deriving a distinguishing test case for a subset of states of a nondeterministic FSM is introduced.

2.1 Preliminaries

A *finite state machine* (FSM), or simply a *machine*, is a 4-tuple S = (S, I, O, h_S), where S is a finite nonempty set of states; I and O are finite input and output alphabets; and $h_S \subseteq S \times I \times O \times S$ is a (*behavior*) *transition relation*. An FSM is *nondeterministic* if for some pair $(s, i) \in S \times I$ there exist several pairs $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_S$. If the FSM has the designated initial state then the FSM is an *initialized* FSM, written (S, I, O, h_S, s_0). An FSM S is *complete* if for each pair (s, i) $\in S \times I$ there exists $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_S$. FSM S is *observable* if for each two transitions $(s, i, o, s_1), (s, i, o, s_2) \in h_S$. It holds that $s_1 = s_2$. FSM S is *single-input* if at each state, there is at most one defined input at the state, and S is *output-complete* if for each pair $(s, i) \in S \times I$ such that the input i is defined at state s. For every output $o \in O$, there exists a transition from s with the input i and output o. An initialized FSM S is *acyclic* if the FSM transition diagram has no cycles. An initialized FSM S is (*initially*) *connected* if each state is reachable from the initial state. See Table 2-1 as an example of FSM S. The machine has four states named 1, 2, 3, and 4, three inputs a, b, and c, two outputs named 0 and 1. Each transition is labeled with an input/output pair. S is nondeterministic because at state 1 under the input a, the machine takes us either to state 2 while producing the output 1, or to state 3 with the output 0.

Table 2-1: Tabular representation of FSM S.

Input\State	1	2	3	4
a	**2** / 1, **3** / 0	**2** / 0	**2** / 0, **4** / 1	**3** / 1
b	**1** / 0	**2** / 1	**3** / 0	**2** / 1
c	**2**/0	**3** / 0	**4** / 1	**2** / 1

In this thesis, the use of adaptive distinguishing experiments is considered with complete nondeterministic observable FSMs which are *non-initialized*. A distinguishing experiment for a subset of states of a non-initialized FSM can be described using an initialized single-input output-complete FSM with an acyclic transition graph that is usually referred to as a *test case* [39, 44].

Test Case: Given an input alphabet *I* and an output alphabet *O*, a *test case* is an initialized initially connected single-input output-complete observable initialized FSM P with an acyclic transition graph.

A state of test case P with no outgoing transitions is a *deadlock state*. By definition, at each intermediate state of P, only a single input is defined with all outputs. A test case over alphabets *I* and *O* defines an adaptive experiment with any complete FSM S over the alphabets *I* and *O*.

In general, given a test case P, the *height* of the test case P is determined as the length of a longest trace from the initial state to a deadlock state of P and it specifies the length of a longest input sequence that can be applied to an FSM S during the experiment.

A *trace* of S at state *s* is a sequence of input/output pairs of sequential transitions starting from state *s* and the set of all traces of S at state *s* includes the empty trace. As an example, the machine S is considered. In Table 2-1, *a*/0 *b*/1 and *a*/0 *b*/0 are two traces at the initial state 1. For state *s* and a sequence $\gamma \in (IO)^*$ of input-output pairs, the γ-*successor* of state *s* is the set of all states that can be reached from *s* by γ. If γ is not a trace at state *s* then the γ-successor of state *s* is the empty set and in this case, we sometimes say that the γ-successor of *s* *does not exist*. For an observable FSM S, the γ-successor of any state *s* has at most one item. Given a subset

b of states of S and a sequence $\gamma \in (IO)^*$, the γ-*successor* of *b* is the union of γ-*successors* over all states of *b*. As examples, considering the FSM in Table 2-1, the *a*/0- *successor* of state 1 is state 2 and the *a*/1- *successor* of state 1 is state 3. Considering the set of states *b* = {1, 2, 3, 4}, it can be concluded that the *c*/0-*successor* of *b* is the set {2, 3}, see Table 2-2.

Table 2-2: Tabular representation of FSM S_{dist}.

input\states	(1,2,3,4)	(2,3)	(2,4)	F
a	*F* / 0 , 1	*F* / 0 , 1	-	*F* / *f*
b	*F* / 0, 1	-	*F* / 1	*F* / *f*
c	(2,3) / 0 , (2, 4) / 1	-	-	*F* / *f*

Let S = (*S*, *I*, *O*, h_S) be a complete observable possibly nondeterministic FSM and *b* be a subset of states of S.

Throughout the thesis, the notations $\bar{s}$ and $\overline{s_1,...,s_m}$ are used for representing a singleton {*s*} and a subset {s_1, …, s_m} of states of an FSM.

Distinguishing test case: A test case P over input and output alphabets *I* and *O* is a *distinguishing test case* (DTC) for the subset *b* if every trace from the initial state of P to a deadlock state of P is a trace at most at one state of the set *b*. If there exists a distinguishing test case for the subset *b* then the set *b* is a *distinguishable* set, or the set *b* is *distinguishable*. If there exists a distinguishing test case for the set *b* = S, then the FSM S is *distinguishable*. See Table 2-3 which depicts a tabular representation of a distinguishing test case for FSM S in Table 2-1.

Table 2-3 : Tabular representation of a distinguishing test case for FSM S.

input\states	$\overline{1,2,3,4}$	$\overline{2,3}$	$\overline{2,4}$
a	-	-	*D* / 0 , *D* / 1
b	-	*D* / 0 , *D* / 1	-
c	$\overline{2,3}$ / 0 , $\overline{2,4}$ / 1	-	-

Here, the construction for the so-called *distinguishing machine* for a set *b* used by the A procedure [39] and by our incremental method, proposed in the following chapter, is included.

2.2 The Exact Algorithm A for Deriving a Distinguishing Test Case:

Derivation of the distinguishing machine $S_{b\text{-}dist}$ **for set *b***: Given a complete observable FSM $S = (S, I, O, h_S)$ and a subset *b* of *S*, the machine S_{b_dist} is constructed in the following way. States of S_{b_dist} are non-empty subsets of *S* union, the designated state *F* and *f* is the designated output that is not in the set *O*. The initial state of S_{b_dist} is state *b* and the machine has the smallest set of transitions which can be constructed using the following:

(1) A transition at state *F* under each input is a self-loop labeled with the output *f*.

(2) For each subset *g*, label a transition from state *g* under input *i* as an *undefined transition* (denoted as '-' at *g*), if for each $o \in O$, the non-empty *io*-successors of every two different states of the set *g* do not coincide and each non-empty *io*-successor of *g* is a singleton. The input *i* is an *undefined* input at *g*.

(3) There is a transition (g, i, o, d) where $d \neq F$ and *d* is not a singleton, if and only if for every $o' \in O$, the non-empty io'-successors of every two different states of the set *g* do not coincide and *d* is the non-empty *io*-successor of the set *g*.

(4) There is a transition (g, i, o, F) if and only if the *io*-successor of the set *g* is not empty and there exists $o' \in O$ such that the non-empty io'-successors of two different states of the set *g* coincide.

The procedure A [39] is used. It takes the FSMs S and S_{b_dist} as an input and returns a message "The set *b* is not adaptively distinguishable" when an adaptive distinguishing test case for the set *b* does not exist. In fact, A is mainly based on iteratively determining and deleting states with undefined inputs of S_{b_dist} to check the existence of a DTC. If a DTC for the set *b* exists, then A uses a simple procedure for constructing the test case.

Exact Algorithm (A):

First, the existence of a distinguishing test case for a complete observable nondeterministic FSM is checked.

Input: FSMs S and S$_{dist}$.

Output : The message "FSM S is not adaptively distinguishing" **or** the set *UN* of undefined input at deleted states.

$UN := \varnothing; k := 1;$

Step 1. For each state c of S$_{dist}$ with an undefined input, an undefined input $un^{\bar{c}}$ at c should be considered and this input should be added to the *UN*;

If there are no states with undefined transitions then

   Return the message "FSM S is not adaptively distinguishing"

 Step 2.

   Remove from S$_{dist}$ each state c with an undefined input $un^{\bar{c}}$ and all incoming transitions to this state;

   If the initial state of S$_{dist}$ is deleted then **Return** the set *UN* ;

  k++ and Go-to **Step 1** ;

After that a DTC is derived from derived from S and S$_{dist}$.

   ***Example* 2.1**. As an example, consider the machine S$_{dist}$ in Table 2-2 with the initial state $S = \{1, 2, 3, 4\}$ derived from the FSM S in Table 2-1. For the obtained machine S$_{dist}$ each of the states $\overline{2,4}$ and $\overline{2,3}$ has an undefined input, and the initial state does not have an undefined input. Thus, by applying the iterative process at Step 1, we delete from S$_{dist}$ states $\overline{2,4}$ and $\overline{2,3}$ and their incoming transitions, and then the initial state will have an undefined transition under the input c. As the initial state is deleted, the FSM S$_{dist}$ has no complete submachine and thus S is adaptively distinguishing and the set *UN* contains $un^{\overline{2,4}} = a$, $un^{\overline{2,3}} = b$, and $un^{\overline{1,2,3,4}} = c$. After applying procedure 1, and as a DTC exists for this example, Procedure 2 in [39] is used to derive the DTC shown in.

Chapter 3 : An Incremental Algorithm for Deriving Distinguishing Test Cases

In this chapter, we propose an incremental algorithm Inc for deriving a DTC for a set b of states of a given FSM. This algorithm considers the states of the set b; one after the other utilizing the following proposed composition operator.

3.1 Incremental Composition Operator:

In order to decide if a set of states has a distinguishing test case incrementally, we propose a special composition operator, denoted as $\otimes$, over two FSMs $S_{b\text{-}dist}$ and $S_{c\text{-}dist}$, where b and c are disjoint subsets of S, that produces the machine $S_{b\cup c\_dist}$. It is also noted that when b (or c) is a singleton, hereafter, $S_{b\text{-}dist}$ is used to denote S with the initial state b (c).

Given disjoint subsets b and c of states of a complete observable FSM S = (S, I, O, h_S) and the machines $S_{b\text{-}dist}$ and $S_{c\text{-}dist}$ derived according to construction rules (1) to (4) given in the previous chapter, the machine $S_{b\text{-}dist} \otimes S_{c\text{-}dist}$ is derived in the following way. The machine $S_{b\text{-}dist} \otimes S_{c\text{-}dist} = (Q, (b, c), I, O \cup \{ f \}, h)$ is the smallest machine Q = $(Q, (b, c), I, O \cup \{ f \}, h_Q)$ with the initial state (b , c) that can be derived using the rules given below. States of $S_{b\text{-}dist} \otimes S_{c\text{-}dist}$ are either pairs of (subsets of) states $S_{b\text{-}dist}$ and $S_{c\text{-}dist}$ where these subsets can both be non-empty, or only one of them is the empty subset, or a state of $S_{b\text{-}dist} \otimes S_{c\text{-}dist}$ can be the designated F state. The output f is the designated output that is not in the set O.

(a) A transition at the state F under each input is a self-loop labeled with the output f.

(b) There is a transition $((m, n), i, o, (d, g))$, $d, g \neq F$, in the FSM $S_{b\_dist} \otimes S_{c\_dist}$ if and only if

1) FSM $S_{b\text{-}dist}$ has no transition (m, i, f, F) and FSM $S_{c\text{-}dist}$ has no transition (n, i, f, F);

2) For every $o \in O$, the non-empty io-successors of m and n do not intersect ;

3) d is the non-empty io-successor of the set m and g is the non-empty io-successor of the set n.

(c) There is a transition $((m, n), i, o, (\varnothing, g))$, $g \neq F$, in the FSM $S_{b\_dist} \otimes S_{c\_dist}$ if and only if the non-empty *io*-successor g exists only for the state n and FSM $S_{b\text{-}dist}$ has no transition (m, i, f, F)

(d) There is a transition $((m, n), i, o, (d, \varnothing))$, $d \neq F$, in the FSM $S_{b\_dist} \otimes S_{c\_dist}$ if and only if the non-empty *io*-successor d exists only for the state m and FSM S_{c_dist} has no transition (n, i, f, F)

(e) There is a transition $((m, n), i, f, F)$ in the FSM $S_{b\text{-}dist} \otimes S_{c\_dist}$ if and only if there is a transition (m, i, f, F) in FSM S_b or there is a transition (n, i, f, F) in FSM S_{c_dist} ; or there exists $o \in O$ such that the non-empty *io*-successors of subsets m and n in FSM S intersect.

Each state (b, c) of the FSM Q obtained after applying construction rules (a) to (e) above is replaced by the union $b \cup c$. The obtained FSM is the FSM $S_{b \cup c\text{-}dist}$.

shows how $S_{\{1,2,3\}\text{-}dist} = S_{\{1,2\}\text{-}dist} \otimes S_{\{3\}\text{-}dist}$ is derived using the above rules.

Table 3-1: Tabular representation of FSM $S_{\{1,2,3\}\text{-}dist} = S_{\{1,2\}\text{-}dist} \otimes S_{\{3\}\text{-}dist}$.

Input\State	$\overline{1,2,3}$	$\overline{2,3}$	$\overline{2,4}$	$\overline{1,3}$	2	3	4
a	F/f	F/f	2 / 0 3 / 1	$\overline{2,3}/0$ $\overline{2,4}/1$	2 / 0	2 / 0, 4 / 1	3 / 1
b	$\overline{1,3}/0$ 2 / 1	2 /1 3 / 0	F/f	$\overline{1,3}/0$	2 / 1	3 / 0	2 / 1
c	$\overline{2,3}/0$ 4 / 1	3 / 0 4 / 1	2 / 0 2 / 1	2 / 0 4 / 1	3 / 0	4 / 1	2 / 1

3.2 Inc Algorithm

Here, we present the incremental algorithm Inc for deriving a DTC for a set of state b of an FSM is introduced. We also define the derivatives Inc(k) of Inc, where the set b is partitioned into k subsets, $k = 2, 3, 4$, and 8.

Incremental Algorithm (Inc). Deriving a distinguishing test case incrementally

Input : FSM S = (S, I, O, h_S), subset $b = \{1, \ldots, m\} \subseteq S, m \geq 3$

Output : The message "The set b is not adaptively distinguishing" or a distinguishing test case R for the set b

Step 1. Partition b into disjoint subsets $b_1 \ldots b_l$ where $|b_1| \geq 2$;

$k := 1$;

Derive the FSM $S_{b(k)\text{-}dist}$ using the rules (1) to (4) given before A algorithm in the previous chapter.

Call Procedure A for $S_{b(k)\text{-}dist}$;

If Procedure A returns the message "The set $b(k)$ is not adaptively distinguishing"

then **Return** the message "The set b is not adaptively distinguishing"

Step 2. While $k < l$

Derive the FSM $S_{b(k+1)\text{-}dist}$ using rules (1) to (4) given in A algorithm;

If $|b(k+1)| > 1$ and Procedure A returns the message "The set $b(k+1)$ is not adaptively distinguishing"

then **Return** the message "The set b is not adaptively distinguishing"

Derive the FSM $S_{c\text{-}dist} = S_{b(k)\text{-}dist} \otimes S_{b(k+1)\text{-}dist}$;

Call Procedure A for $S_{c\text{-}dist}$;

If Procedure A returns the message "The set c is not adaptively distinguishing"

then **Return** the message "The set b is not adaptively distinguishing"

k++ ;

$S_{b(k)\text{-}dist} = S_{c\text{-}dist}$;

    EndWhile

Step 3. Construct the test case R for the set c as given in [39];

End Inc

    ***Example* 3.1**. Considering the machine S in Table 2-1, the set $b = \{1, 2, 3, 4\}$, and the partition $b_1 = \{1, 2\}$, $b_2 = \{3\}$, and $b_3 = \{4\}$ of b. At Step 1 of Inc, starting with the pair b_1 obtained, by rules (1) to (4), the FSM $S_{\{1,2\}\text{-}dist}$ in Table 3-2. As the initial state $\overline{1,2}$ has an adaptive distinguishing test case for states 1 and 2, then, at Step 2, in the first iteration of the while-loop, the machine $S_{\{1,2,3\}\text{-}dist} = S_{\{1,2\}\text{-}dist} \otimes S_{\{3\}}$ in Table 3-1 is derived. As the set $\overline{1,2,3}$ has a DTC, in the second iteration of the loop, the machine $S_{\{1,2,3,4\}\text{-}dist}$ (in Table 3-3) is constructed, and afterwards at Step 3 a test case is constructed for the set $\overline{1,2,3,4}$ as given in [39] It is noted if at Step 1 there is no DTC for the pair $\{1, 2\}$, or if after the first iteration of the loop there is no DTC for the set $\{1, 2, 3\}$, then the incremental procedure would terminate declaring that there is no DTC for the considered set b. As another application of Inc, we could consider the partition $b_1 = \{1, 2\}$ and $b_2 = \{3, 4\}$ of the set b. In this case, after deriving $S_{\{1,2\}\text{-}dist}$ at Step 1, at Step 2, the machine $S_{\{3,4\}\text{-}dist}$ is derived and as the set $\overline{3,4}$ has a DTC, in the second iteration of the loop, the machine $S_{\{1,2,3,4\}\text{-}dist}$ is constructed and afterwards as the set $\{1, 2, 3, 4\}$ is adaptively distinguishing, a DTC for the set b is derived.

Table 3-2: Tabular representation of FSM $S_{\{1,2\}\text{-dist}}$.

Input\State	$\overline{1,2}$	$\overline{2,3}$	1	2	3	4	F
a	$\overline{2,3}$ / 0 2 / 1	F / f	2 / 1 3 / 0	2 / 0	2 / 0 4 / 1	3 / 1	F / f
b	1 / 0 2 / 1	2 / 1 3 / 0	1 / 0	2 / 1	3 / 0	2 / 1	F / f
c	$\overline{2,3}$ / 0	3 / 0 4 / 1	2 / 0	3 / 0	4 / 1	2 / 1	F / f

Table 3-3: Tabular representation of FSM $S_{\{1,2,3,4\}\text{-dist}} = S_{\{1,2,3\}\_\text{dist}} \otimes S_{\{4\}\text{-dist}}$.

Input\States	(1,2,3,4)	(2,3)	(2,4)	1	2	3	4	F
a	F / 0 , 1	F / 0 , 1	-	-	-	-	-	F / f
b	F / 0, 1	-	F / 1	-	-	-	-	F / f
c	$\overline{2,3}$ / 0 , $\overline{2,4}$ / 1	-	-	-	-	-	-	F / f

3.3 Inc(k) Implementations

As Inc starts from a given partition $b_1…, b_l$ of a set $b = \{1, 2,... s_m\} \subseteq S$ of the specification machine S, $| b | = m \leq n = /S/$, then it would be interesting to assess the performance of Inc considering various ways of partitioning the set b. We consider the following types of partitions and rename Inc accordingly to simplify discussions. Namely, hereafter, Inc is used when the set b is partitioned into $| b | - 1$ subsets as indicated in the algorithm, i.e., $b_1 = \{1, 2\}$ and for every $i = 2, …, l$, b_i is a singleton, i.e., $b_i = \{s_{i+1}\}$. We use Inc(k), $k = 2, 3, 4$, or 8, when b is partitioned into k subsets $\{b_1\}, … \{b_{k-1}\}, \{ b_k\}$. We experiment with two values of m, namely, $m = 5$ and $m = | S |$.

Chapter 4 : A Heuristic Approach for Deriving Distinguishing Test Cases

In this chapter, some rules that can be used for constructing a DTC for a set of states of a specification FSM are presented. The first approach, called H, uses the depth first search techniques for a given fixed height while appropriately utilizing hashing to speed up the search process. The second approach, called Hc, is the same as the first; however, it uses a cost function for ordering the inputs to be considered in the search. Experimental evaluation of these approaches is provided in Chapter 5.

4.1 Rules for Constructing a DTC

Given a specification FSM S, an approach is presented here for constructing a DTC for a given set b of states based on Depth-First-Search (DFS) traversal of a truncated successor tree of height $L \geq 1$ that is derived using the following rules. The truncated successor tree of height L is constructed using the following rules:

(a) The root node of the tree is labeled with the set b and an edge between two nodes of the tree is labeled by an input-output i/o pair.

(b) At each non-leaf (or non-terminal) node of the tree, there exist $|O|$ outgoing transitions only under the same input $i \in I$. A terminal leaf node has no outgoing transitions.

(c) Given a non-leaf (or non-terminal) node labeled with a set of states c and an input i, for each output $o \in O$, there is a transition $(c, i/o, d)$ where d is the io-successor of the set c or the empty set.

(d) Given a node labeled (with the subset) d with parent c and transition $(c, i/o, d)$, node d is called a *bad terminal node* if there are two different states of c where the non-empty io-successors of these states coincide. By definition, a bad terminal node has no outgoing transitions.

(e) Given a node d with parent c and transition $(c, i/o, d)$, node d is called a *good terminal leaf node* if the io-successor of c is a singleton or the empty set and d is not a bad node.

(f) Given a non-leaf node c such that for each input i there is an edge to a bad node, then the c is also a bad node.

(g) Given a node c at the L-th level of the tree starting from the initial root, if c is not labeled with a singleton or the empty set (i.e., c is not a good terminal node), then c is labeled as a bad node (due to the height); otherwise, c is a good terminal leaf node. This rule is added to limit the search up to a certain height L, where the value of L is appropriately set according to conducted experiments.

If there exists a finite tree derived using rules (a) to (g) such that each leaf node of the tree is a good terminal leaf node then this tree represents a DTC for the set b, written DTC_b. By definition, a subtree DTC_b with the root node labeled with the set c, denoted DTC_c, represents a DTC for the set c.

4.2 H and Hc Methods

Two DFS based versions of the above rules are implemented; both implementations use the same rules for an ordered list of inputs while exploring the tree. This list provides the order that will be used to derive the i-successors of a current subset c. The difference between these implementations is that the first version, hereafter named H, uses the (same) predefined order $i_1, i_2, \ldots, i_{k-1}, i_k, k = |I|$, of inputs at each node of the tree; while the second implementation, named Hc, uses a cost function that provides an order among the inputs of I at each node of the tree.

More precisely, at a current node c we consider the ordered list of inputs $i_1, i_2, \ldots, i_{k-1}, i_k, k = |I|$ and the following rules:

(h) Let i be the current input of the ordered list:

If there exists an output o such that there are two different states of c where non-empty io-successors of these states coincide or the i/o successor of c is a bad node, then:

(1) Node c is labeled as a *bad node under input i* ;

The input i is deleted from the ordered list of inputs attached to node c ;

If the list of inputs at c is empty (there are no more inputs to consider at c), then c is labeled as a *bad node* (i.e., node c is bad under all inputs), outgoing transitions of c are deleted.

(i) If node d is labeled as a bad node, consider the transition from the parent node c leading to the bad node d over the input i, then apply (1) above to node c.

The cost function for the Hc implementation is calculated as follows:

Hc Cost function: The order list of inputs at a node is labeled with the subset of states c for each input i in I. The cost of i under c is calculated as the total number of the *io*-successors of c over all outputs o in O.

On Speeding up the search process: Speeding up the search process is achieved by hashing the good nodes that lead us to solution. Then afterwards if during the search an *io*-successor of a current state c is found to be d, then no further search is carried out from d; node c is labeled as bad node under i, and handling node c is done as described in rules (h) and (i) above. Furthermore, if during the search, a DTC_d is established for a subset of states say d, then d is hashed as a good node and DTC_d is saved. In this case, if afterwards during the search node d is encountered again, no further search is carried out of d as it is considered as a good terminal leaf node since a DTC for the subset d is already derived.

Example **4.1**. As an application example of the H approach, consider the FSM S_1 in Table 4-1 with inputs $\{a, b\}$, outputs $\{0, 1\}$, and states $\{1, 2, 3, 4\}$. The H method is applied starting from the set of states $b = \overline{0,1}$ using the order $a/0$, $a/1$, $b/0$, $b/1$ for input/output pairs at tree nodes. First, as none of the non-empty $a0$ - successors of state 0 and $a/0$ - successors of state 1 coincide, then node n_2 labeled by the $a0$-sucessor of $\overline{0,1} = \overline{1,2}$ is added to the tree (Figure 4-1). Then at n_2, considering the input a, as the $a/0$-successors of states 1 and 2 coincide, then input a is considered as a bad input at $\overline{1,2}$, and we proceed to the second input b. Then, as none of the non-empty $b/0$-successors of state 1 and $b/1$-successors of 2 coincide, then $b/0$-successor $\overline{0,4}$ (node n_3) of $\overline{1,2}$ is derived. The $a/0$-sucessor of $\overline{0,4}$ is the singleton $\{1\}$, then as this $a/0$-sucessor is a terminal leaf node, the $a/1$-sucessor $\overline{1,3}$ is derived,

and the process repeats until the tree, which represents a DTC for $\overline{0,1}$, given in Table 4-1 is obtained.

Note that for this example, if hashing is used as illustrated above, it is observed that the tree with root node n_5 labeled by $\overline{1,3}$ represents a DTC for $\overline{1,3}$, and thus $\overline{1,3}$ is hashed as a DTC for $\overline{1,3}$, then later in the search as the $a/1$-sucessor of $\overline{1,0}$ is $\overline{1,3}$ (node n_{12}), thus, no further exploration is done from node n_{12} i.e., the nodes n_{13}, n_{14}, and n_{15}, are not generated, and this speeds up the search process. Further, if we assume that the maximum height L equals 2, then, while conducting the search, node n_3 becomes a leaf node and thus we do not obtain a DTC (of height 2) for the pair $\overline{0,1}$.

Table 4-1: Tabular representation of FSM S_1.

Input\State	0	1	2	3	4
a	1 / 0, 1 / 1	2 / 0, 3 / 1	2 / 0 , 3 / 1	3 / 2	3 / 1
b	2 / 0	0 / 0	4 / 0	3 / 0	4 / 2
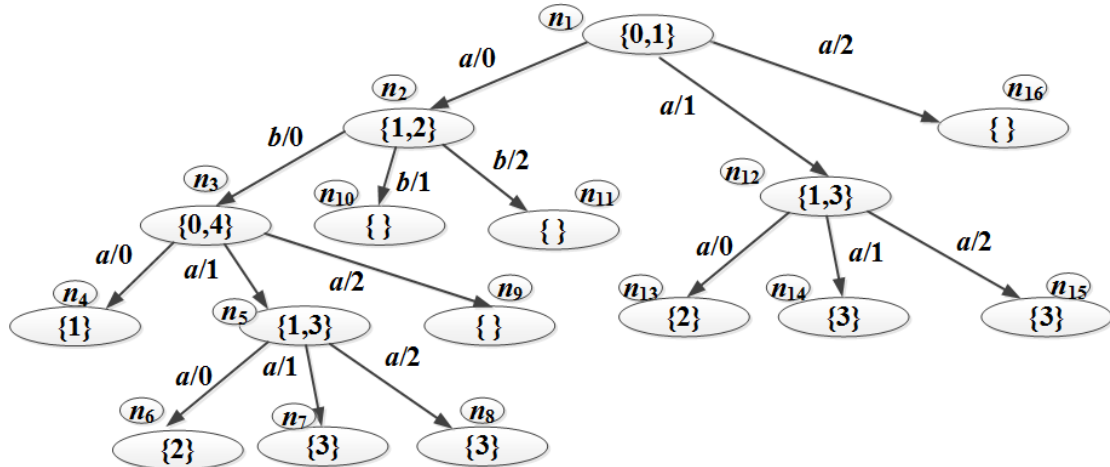
Figure 4-1: Tree obtained of applying the H method starting from the pair $\overline{0,1}$.

***Example* 4.2**. Here we apply the Hc method to the set machine described in previous example again considering the same set of state $b = \overline{0,1}$. First, at the initial node, as the cost of the a-sucessors of $\overline{0,1} = 4$ and that of the b-sucessors of $\overline{0,1} = 2$, the order b (namely, $b/0$; $b/1$) followed by a (namely, $a/0$; $a/1$) is used when

searching from a node labeled with $\overline{0,1}$. Afterwards, node n_3 is derived and again the order $b/0$; $b/1$; $a/0$; $a/1$ is established for this node, then the process repeats till the tree in

Figure 4-2, which also represents a DTC for the set $0,1$ is obatained. For this example, it is clear that Hc even for the height $L = 4$ derives much smaller tree than that of the H method.
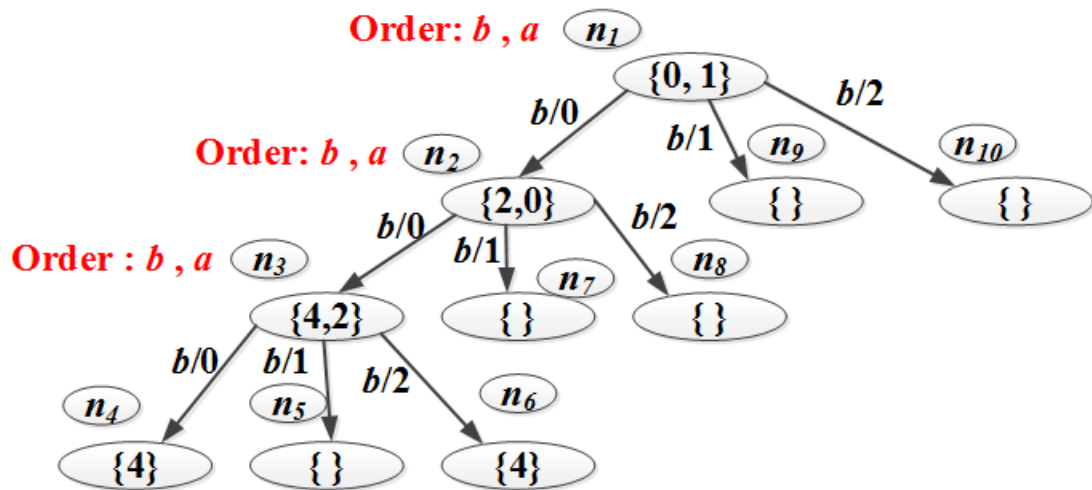

Figure 4-2: Tree obtained by applying the Hc method starting from the pair $\overline{0,1}$.

Chapter 5 : Experimental Evaluation

This chapter includes the experiments conducted for assessing the A, H, and Hc methods in terms of execution time and quality of obtained solutions (height of obtained DTCs). Experiments are conducted to study how often a DTC exists depending on some FSM attributes and a number of states to be distinguished. Thus, according to the conducted experiments, H and Hc significantly outperform A in terms of execution time. H and Hc execution time is then assessed over large size machines. The execution time of A and its incremental implementations are also compared as both always find a solution when a solution exists. It is also interestingly observed that, for the conducted experiments that both H and Hc never missed finding a solution when a solution existed, and never declared that there is no solution where in fact a solution exists. In other words, the maximum search space of H and Hc is set appropriately to obtain such results.

Experiments were conducted using randomly generated (complete) machines with various combinations of attributes including a number of states $|S|$, inputs $|I|$, outputs $|O|$, minimum *min* and maximum *max* degrees of non-determinism. The minimum or maximum degree of non-determinism represents the minimum or maximum number of transitions that are going from each state under each input of the generated machine. In addition, we also conducted experiments using some real machines and obtained the same pattern as random machines.

In general, depending on the assessment objective, as illustrated below, experiments are conducted to assess performance when:

$|b| = 2$, i.e. when the number of states to distinguish in the set b equals two, or

$b = S$ and thus $|b| = |S|$, i.e., when the objective is to distinguish all states of the machine.

In some cases, several values of $|b|$ are considered. It is noted that for each considered combination of attributes and a considered value of $|b|$, we run five different experiments and then calculate and depict the average of the obtained results in the corresponding figures. The experiments were obtained using an Intel(R)

Core$^{(TM)}$ i5 3337U CPU @1.80GHz with 4 GB RAM machine under Windows 7 (32 bit).

5.1 Existence of DTCs

Here the existence of a DTC depending on the number of state, outputs, and the degree of non-determinism is checked. Experiments are conducted using machines with the following combinations attributes $|S| = 40, \ 80, \ 120, \ |I| = 6, |O| = 4, 6, 15, 20, |S|/2, |S|/2 + 10$, $min = 1, 2$ and $max = 2$.

Existence when $|b| = 2$: DTCs were always obtained in all conducted experiments when the objective was to distinguish a randomly selected pair of states of the machine.

Existence when $b = S$: In general, according to the results depicted in Figure 5-1, as the number of outputs increases, for the same considered number of inputs, the possibility of finding a DTC for all states of the machine increases.
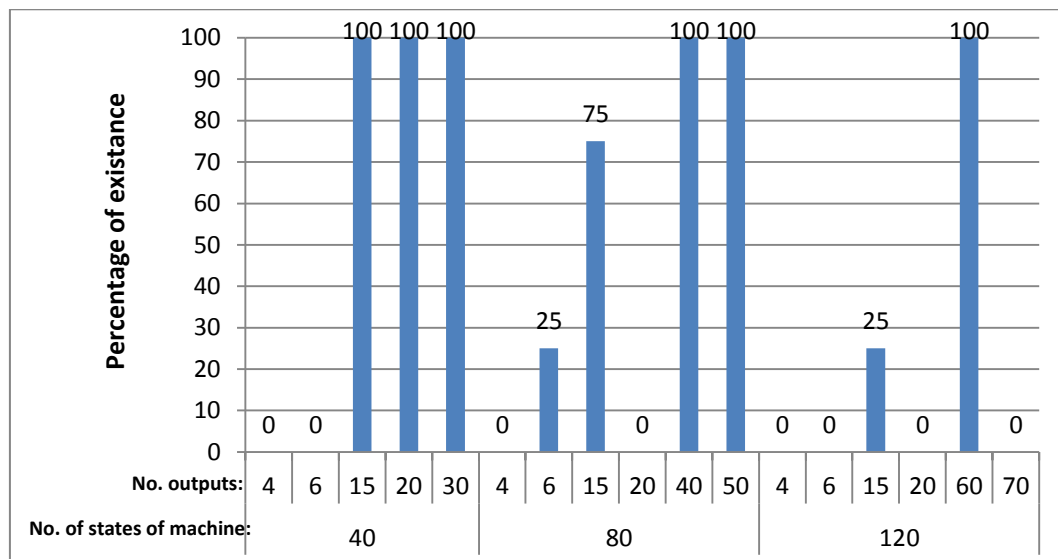
Figure 5-1: Existence of DTCs when $b = S$ for deterministic and nondeterministic FSMs.

5.2 A versus Incremental

Here the performance of A, Inc(k) described in previous sections, k could be 2, 3, 4, 8 states is assessed. It is noted that every point found in the figures below

corresponds to the average results of the experiments done for the machines with the same values of attributes.

i) Representative example: We will have two parts of representative results; results when we have DTC, and when there is no such DTC.

a) Results when a DTC exist: For experiments with distinguishing sequences, we use machine with the following specifications: $|S| = 50, |I| = 6, |O| = 40, min = 1, max = 2$ the result in Figure 5-2, Inc is the slowest one because of the exhaustive use of composition operator which is very expensive and consumes a lot of time. The fastest one is A, then Inc(2), Inc(3), Inc(4), then Inc(8). This means the less usage of composition operator, the faster the algorithm is when there is a DTC.
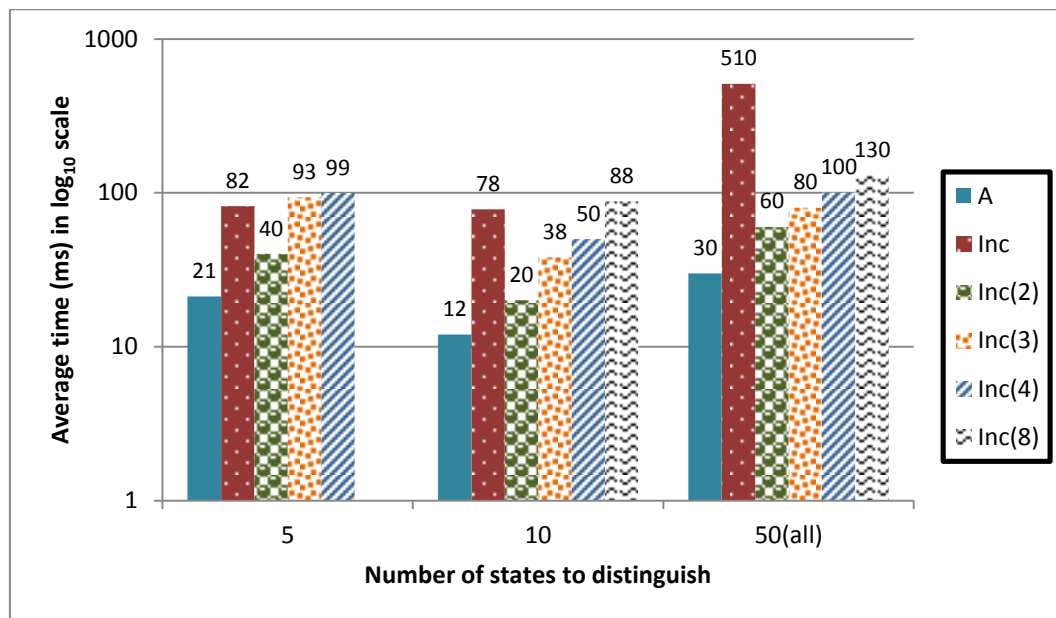
Figure 5-2: A vs Inc, Inc(2), Inc(3), Inc(4), Inc(8) when there is a DTC.

b) Results when there is no DTC: Here we use machine with $|S| = 50, |I| = 6, |O| = 5, min = 3, max = 3$. In Figure 5-3 results showed that Incremental outperforms A when the goal is to distinguish small numbers like $b = 5$, but when $b = 10$, not all incrementals have better performance because when we use Inc(2) which has the slowest performance, the composition operator consumes a lot of time, because it builds a tree here when the division is 2 which makes it very slow. Inc(3) also was slow. The order of algorithms is as follows: Inc, Inc(4), Inc(8) have the same

performance , then Inc(3), followed by A, then Inc(2). While when $b = S$; A, Inc(2), Inc(3), Inc(4), followed by Inc, then Inc(8). Inc(8) is the slowest one, which means the composition operator here is used a lot.
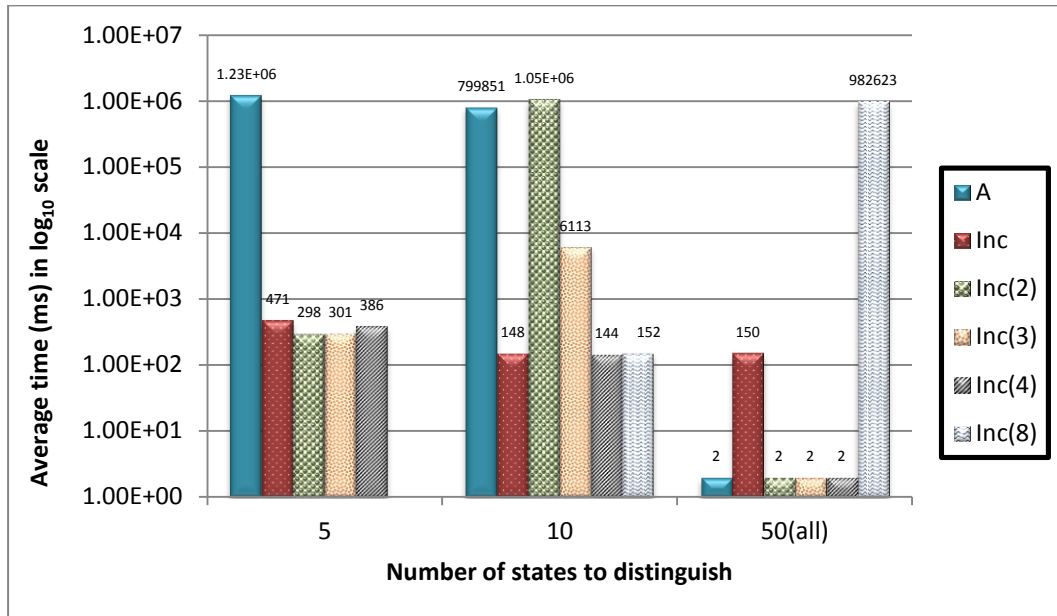

Figure 5-3: A vs Inc, Inc(2), Inc(3), Inc(4), Inc(8) when there is no a DTC.

ii) Summary of all conducted experiments: This summary reports results over all machines used when we have DTC, and when we don't have.

a) Results when a DTC exist: Machines with the following attributes are used: $|S| = 50, 100, 150, |I| = 6, |O| = 40, 45, min = 1, max = 2$. According to the results shown in Figure 5-4, when $|b| = 5$ or $b = S$ (all states of the machine), A outperforms all others followed by Inc(2), Inc(3), Inc(4), Inc(8), where Inc has the worst performance. The reason is that the composition operator used in the incremental implementations is expensive and thus when there is a DTC, the incremental takes more time than A. Thus, the less number of times the composition is performed, the better, i.e., Inc (2) is better than Inc (3), etc.
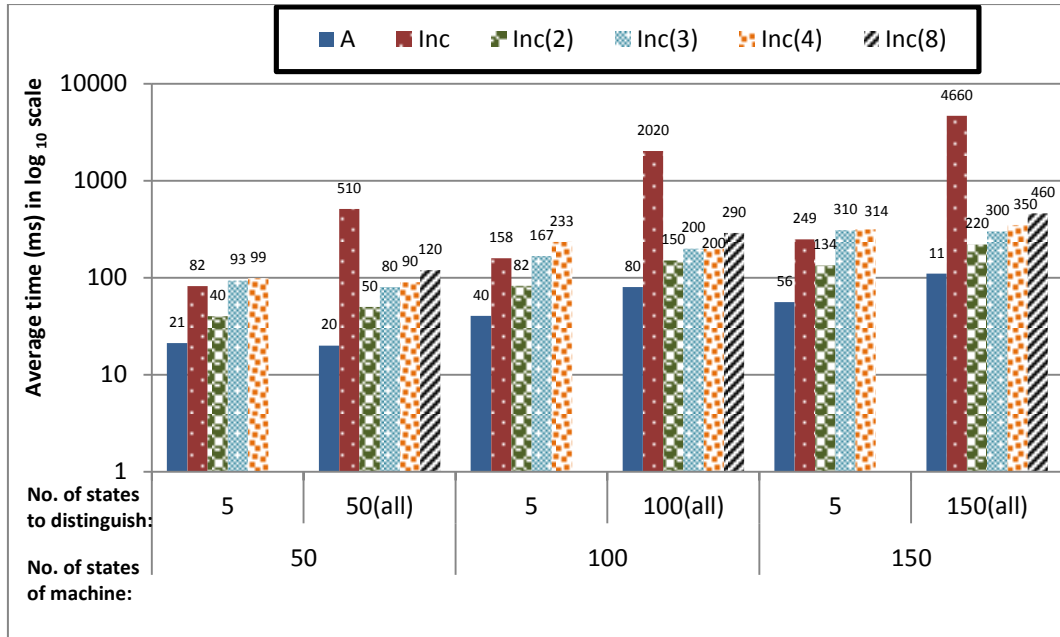
Figure 5-4: A and incremental when a DTC exists.

b) Results when there is no DTC: Here machines with the following attributes are used: $|S| = 50, 100, 150, |I| = 3, |O| = 3, min = 2, max = 2$, and we consider the cases when $|b| = 5$ and $b = S$. According to the obtained results in Figure 5-5, when $|b| = 5$ and there is no DTC for the set b, Inc(4), inc(3), Inc, Inc(2) have a comparable performance; followed by A. A did not produce results when $|S| = 150$. When $b = S$, A, Inc(2), Inc(3), Inc(4) obtained comparable performance, followed by Inc, followed by Inc(8).

Thus, it is clear that the incremental outperforms A when the objective is to distinguish a small number of states, here $|b| = 5$; however, when the objective is to distinguish a large number of states, namely, all states, A and Inc(k), $k = 2, 3,$ or 4 showed comparable results as usually all detect faster, than when considering $|b| = 5$, that there is no DTC. However, Inc and Inc(8) did not perform as good as others as Inc considers the states of the set b one by one and Inc(k) divides the set into many subsets; and thus both Inc and Inc(8) used the composition operator more than other methods.
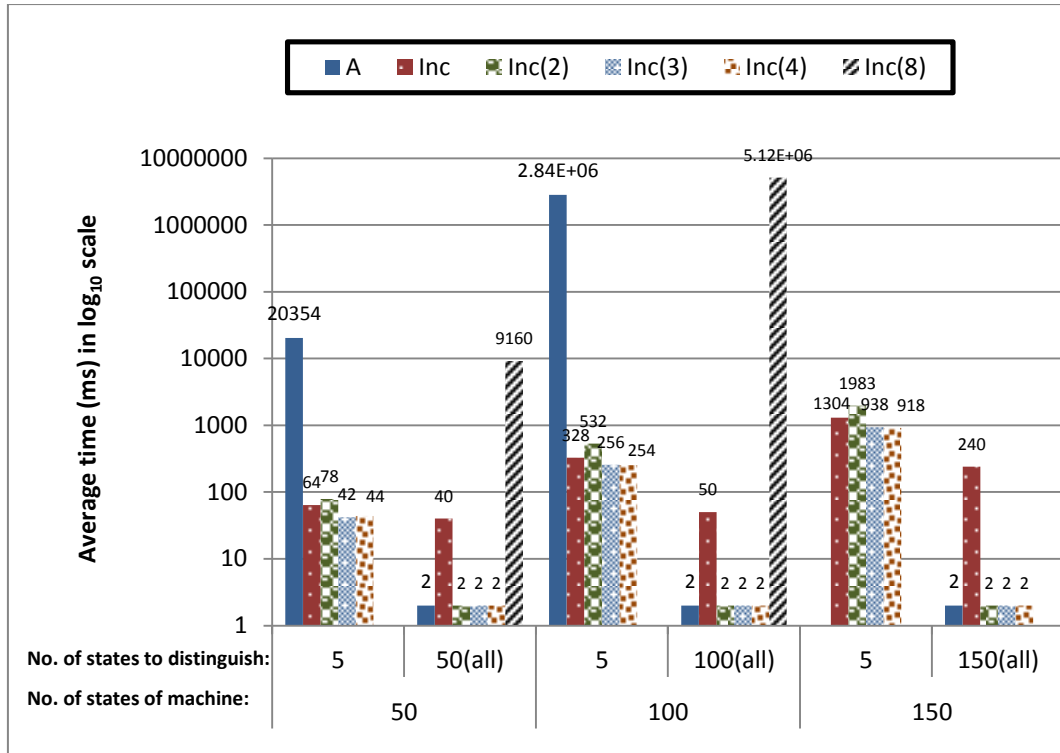
Figure 5-5: A and incremental when there is no a DTC.

5.3 Execution Time of A, H, and Hc Algorithms

i) Representative example: There are two parts of representative results; results when there is DTC, and when there is no such DTC.

a) Results when a DTC exist: Representative example: Here we used machine with: $|S| = 50$, $|I| = 6$, $|O| = 26$, *min* = 1, *max* = 2, and Figure 5-6 shows that the execution time increases as the number of states that we want to distinguish increases of experiments (FSMs) with distinguishing sequences. It shows that H, Hc performs much better than A.
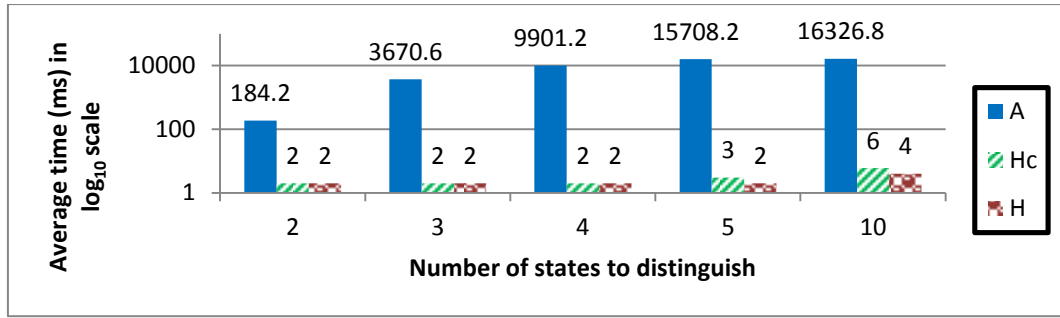
Figure 5-6: Average time vs number of states to distinguish (A vs H vs HC) when there is a DTC.

b) Results when there is no DTC: We used $|S| = 50$, $|I| = 6$, $|O| = 5$, *min* = 3, *max* = 3. Figure 5-7 shows the execution time as the number of states that I want to distinguish increases of experiments (FSMs) without distinguishing sequences. According to these experiments, H, Hc always outperforms A from 2 to 10 states; after that A has the same performance as H and Hc.
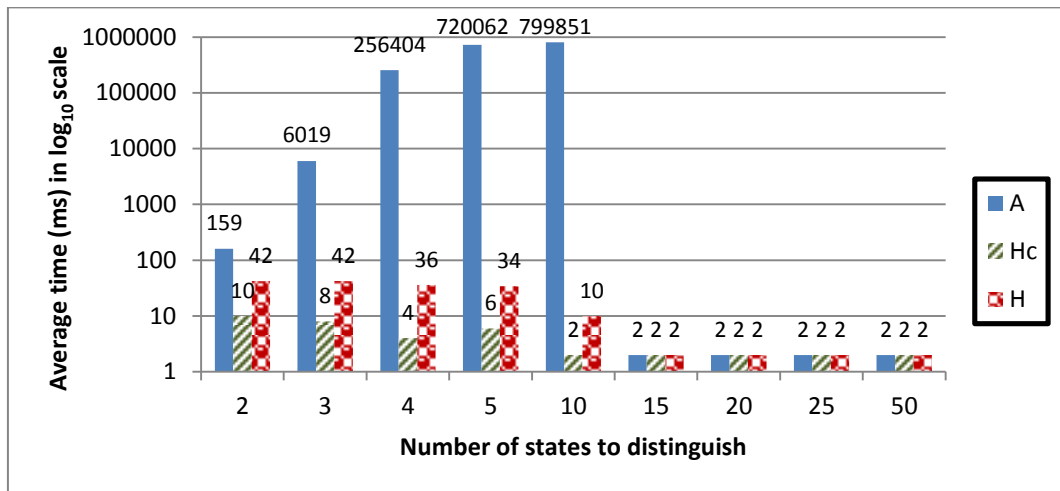

Figure 5-7: Average time vs number of states to distinguish (A vs H vs HC) when there is no a DTC.

ii) Summary of all conducted experiments: it reports results over all machines used when we have DTC, and when we don't have.

a) Results when a DTC exist: **Results when $|b| = 2$:** Here we used machines with the following specifications:$|S| = 50$, 70, 80, 100, 200, 250, 300, 350, 400, 450,

500, 550, 600, 650, $|I| = 10$, $|O| = 10$, $min = 5$, $max = 5$. From Figure 5-8 we can see that H, and Hc always outperform A when $|b| = 2$, and in general Hc is faster than H.
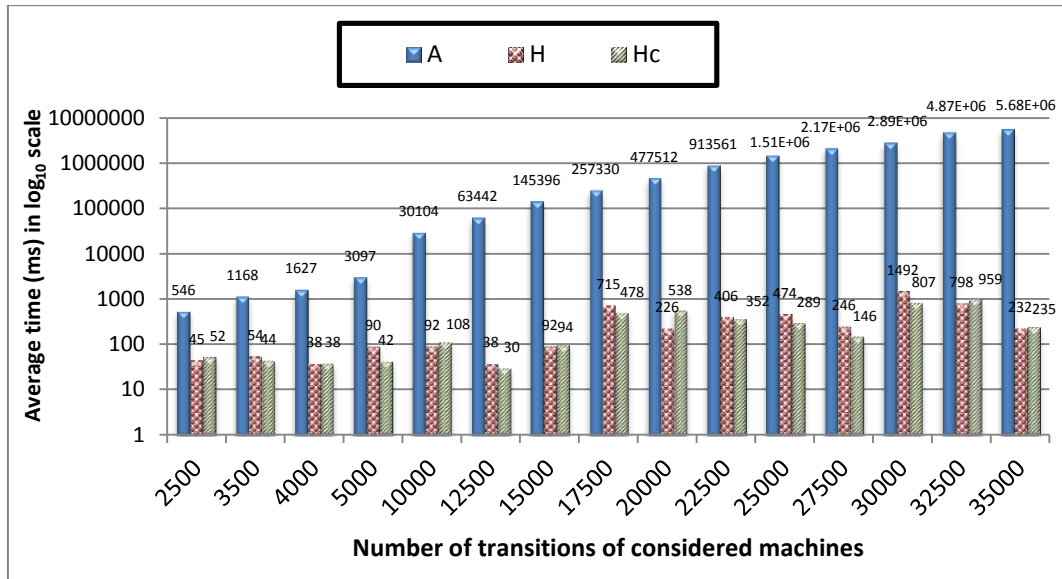

Figure 5-8: Average time vs number of transitions when there is a DTC and $|b| = 2$.

Results when $b = S$: In order to compare results when the DTC is over $b = S$, we used machines with:$|S| = 20, 40, 80, 120$, $|I| = 6$, $|O| = 4, 6, 15, 20$, $|S|/2$, $|S|/2 + 10$, $min = 1$, and $max = 2$. Figure 3-8 showed that H, and Hc always outperforms A when $b = S$. Also, Hc is faster than H. Figure 5-9 showed that H, and Hc always outperform A when $b = S$. Also, Hc is faster than H.
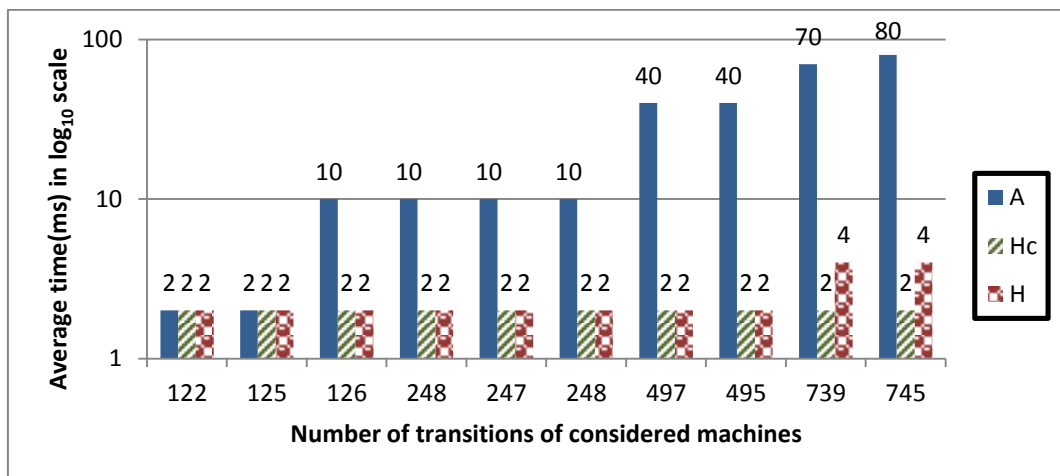

Figure 5-9: Average time vs. number of states when there is a DTC and $b = S$.

39

b) Results when there is no DTC: Here we used machines with the following specifications: $|S| = 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, |I| = 10, |O| = 10, min = 7, max = 7$.

Results when $|b| = 2$: Figure 5-10 showed that when we want to find DTC for $|b| = 2$, H, and Hc outperforms A, while the execution time for A increases dramatically when the number of transitions increases.
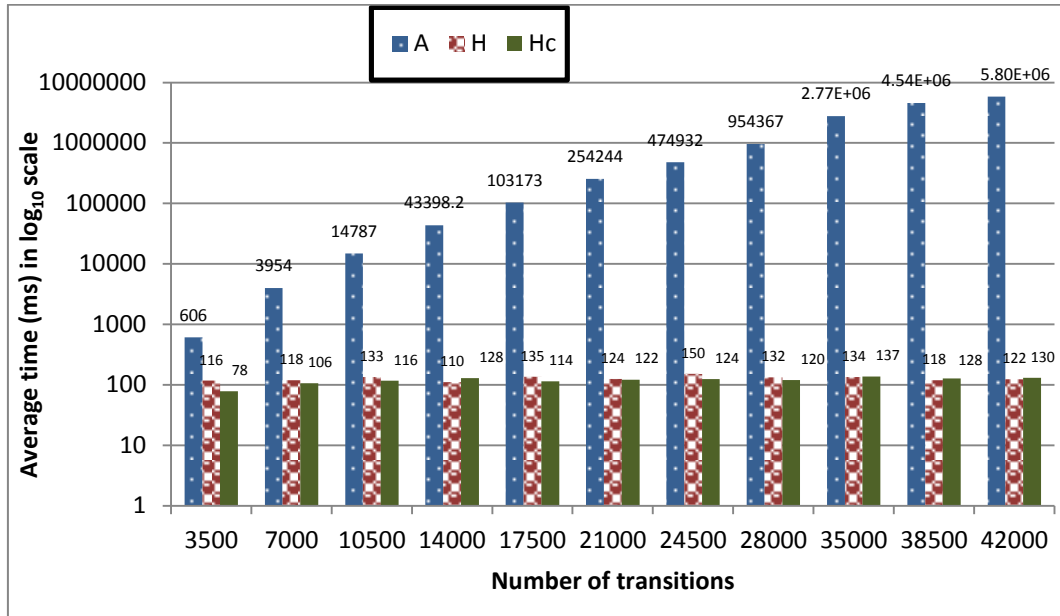

Figure 5-10: Average time vs number of states when there is no a DTC and $|b| = 2$.

Results when $b = S$: All H, Hc, and A all have the same performance here.

5.4 Comparison between H and Hc Over Large Machines

Here we assess the performance of H and Hc considering large size machines. We consider machines with the following attributes: $|S| = 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000, 11000, 12000, 13000, 14000, 15000, 16000, 17000, 18000, 19000, 20000, |I| = 10, |O| = 10, min = 5, max = 5,$ and $|b| = 2$. According to the obtained results depicted in Figure 5-11, it is clear that both methods can handle big machines; however, Hc almost always outperforms H in terms of execution time.
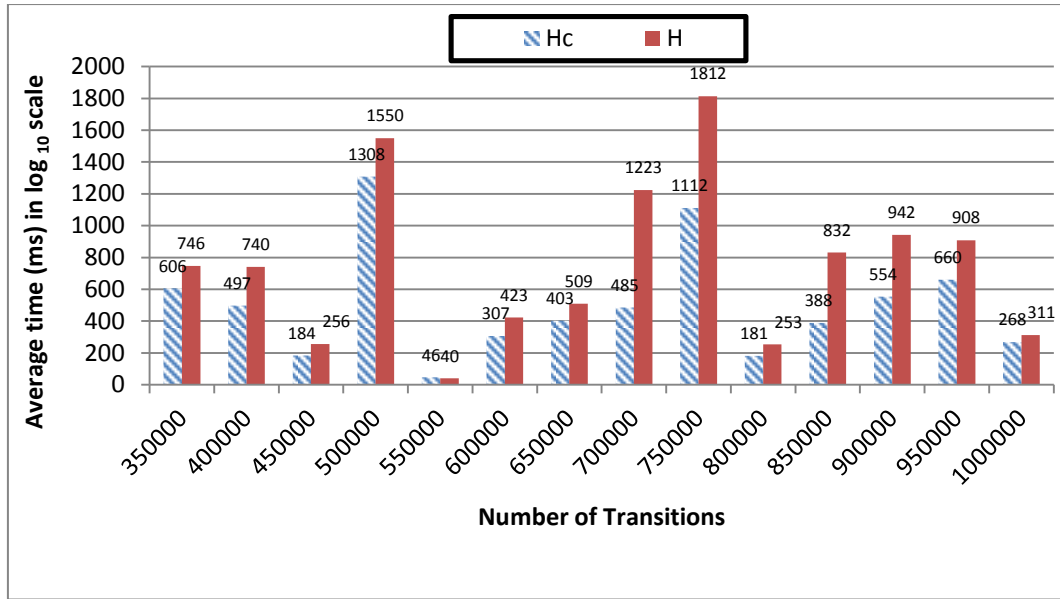
Figure 5-11: Average time vs number of transitions when there is a DTC.

5.5 Comparison of A, H, and Hc in Terms of the Quality of Obtained Solutions

A drawback of the H and Hc methods is that, by construction, they might miss a solution when a solution exists; i.e., they might declare that there is no DTC when in fact, there is a DTC. This is due to the fact that the search of these methods is carried out up to a fixed depth. In order to avoid encountering this problem, the value L is appropriately set, based on the conducted experiments, in such a way that H and Hc always find a solution when such a solution exists. More precisely, for small and medium size machines, up to 40000 transitions, the value of L is set according to the longest DTC obtained by A over all conducted experiments. For large size machines, we experimented with larger depth values and had set L appropriately according to the longest obtained length.

In order to compare the results of A, H, Hc in terms of average length of obtained adaptive sequences (the quality of obtained solutions) when $| b | = 2$, machines with the following combinations of attributes are used, $| S | = 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 650, | I | = 10, | O | = 10, min = 5, max = 5$. The average length of obtained sequences using the A, H, and Hc are 5.25, 5.7, and 5.3, respectively; thus, A, H and Hc return sequences of comparable length. We can see from Figure 5-12, if we want to compare 5 states, the gap between H, and Hc will

increase. And so on for 10 and 15 states. This means Hc does not only enhance the time, but it also enhances the length.
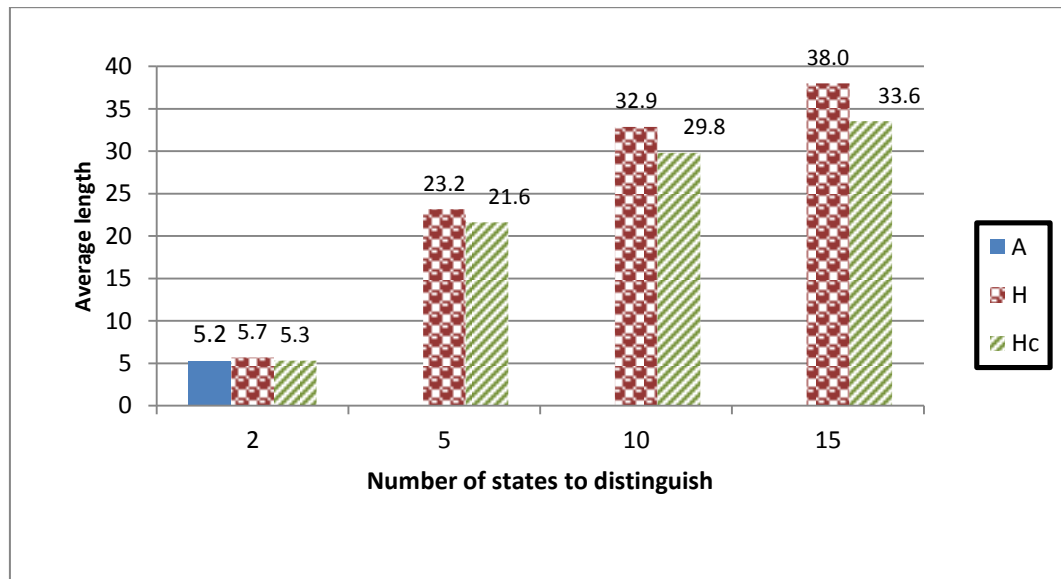

Figure 5-12: Average length when number of states to distinguish equals $| b | = 2, 5,$ 10, and 15.

In order to compare the quality of solutions when $b = S$, machines with the following attributes are used: $| S | = 20, 40, 80, 120, | I | = 6, | O | = 4, 6, 15, 20, | S | / 2, | S | / 2 + 10, min = 1,$ and $max = 2$. For all conducted experiments, A, H, and Hc return DTCs of average length 2.1 when a DTC exists. In other words, when $b = S$ and a DTC exist, the DTC length is usually small.

5.6 Experiments with Real Machines

In this part, we compare between A, Inc, IncA(2), H, Hc in terms of execution time using real FSMs taken from the ACM/SIGDA benchmarks [45].

Table 5-1: Real machines.

Name	Number of states	Number of transitions	Number of inputs	Number of outputs
s1488	48	12288	256	64
log	17	8704	512	17
s1a	20	5120	20	256
s386	13	1664	128	11
pma	24	3728	244	24
sse	16	5264	128	22

5.6.1 Existence of DTCs: Here we check the existence of a DTC depending on the number of states. Experiments were conducted using these machines: s386, s1a, log, s1488, sse, pma.

Existence when | b | = 2: DTCs were always obtained over all conducted experiments when the objective was to distinguish a randomly selected pair of states of the machine. Except for s1a, it doesn't have sequence.

Existence when | b / = S: According to the results depicted in Figure 5-13 when | b | = S, the possibility of finding a DTC for all states of the machine decreases.
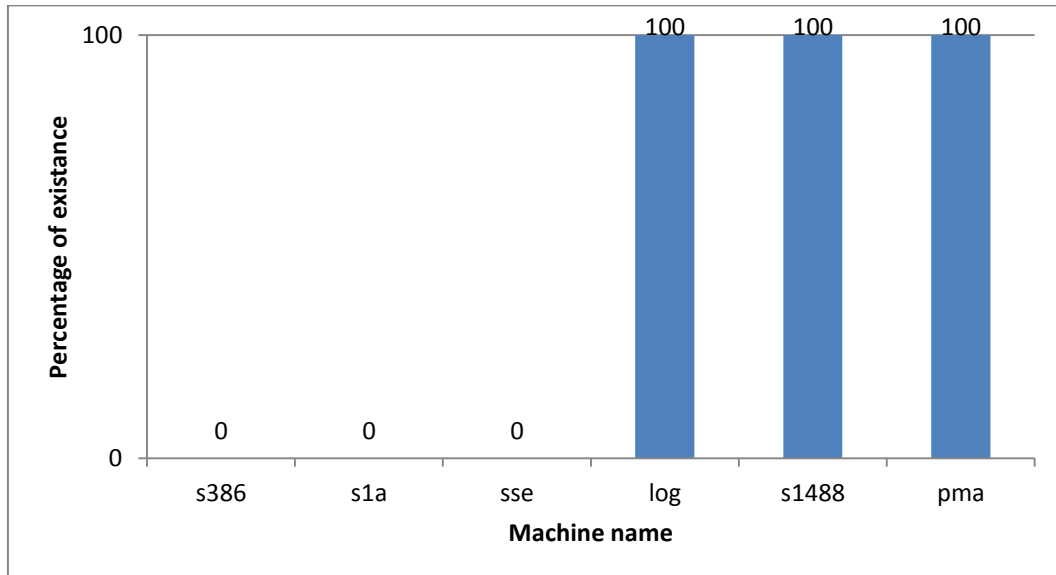

Figure 5-13: Existence of a DTCs when | b | = S.

5.6.2 A versus Incremental: Here we assess the performance of A, Inc and Inc(k) described in previous sections.

i) Representative example: We will have two parts of representative results; results when we have DTC, and when there is no such DTC.

a) Results when a DTC exist: Here we use machine "log" with specifications $| S | = 17$, number of transitions $| T | = 8704$, $| I | = 512$, $| O | = 17$. Figure 5-14 shows that when $| b | = 5$ or $| b | = 10$ or $| b | = S$. A has the best performance over all algorithms, then Inc(2), followed by, Inc(3), then Inc(4), then Inc(8), Inc comes at the last.
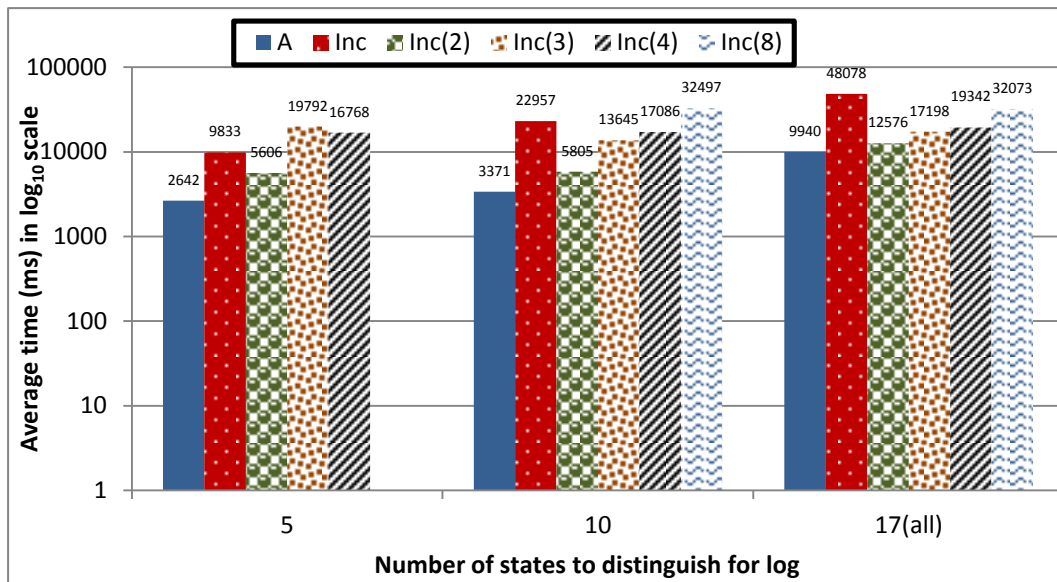

Figure 5-14: Average time in (ms) versus number of states that we want to distinguish with a DTC for "log" machine.

b) Results when there is no DTC: we use machine "s1a", $| S | = 20$, number of transitions $| T | = 5120$, $| I | = 20$, $| O | = 256$. It gives us results with no sequence. Figure 5-15 showed when $| b | = 5$, Inc(k) outperform A, but after that A has the best performance, because other Incrementals divide the subset b into more smaller parts and that means the composition operator will be used a lot.
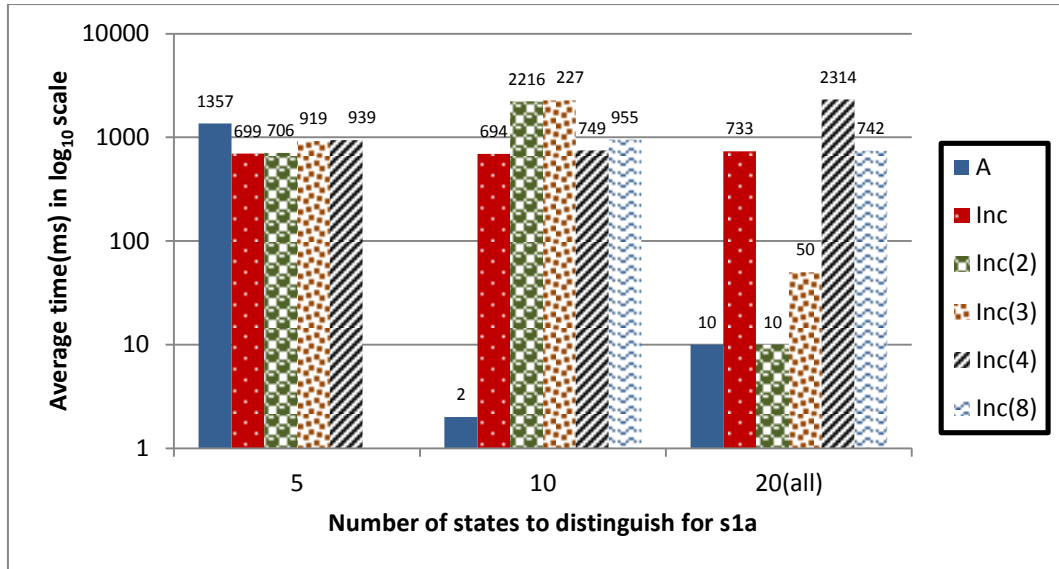
Figure 5-15: Average time in (ms) versus number of states that we want to distinguish when there is no a DTC for "s1a" machine.

ii) Summary of all conducted experiments: it reports results over all machines used when we have DTC, and when we don't have.

a) Results when a DTC exist: According to the results shown in Figure 5-16, when | b | = 5 or $b = S$ (all states of the machine), A outperforms all others followed by Inc(2), Inc(3), Inc(4), Inc(8), where Inc has the worst performance. The reason is that the composition operator used in the incremental implementations is expensive and thus when there is a DTC, the incremental takes more time than A. Thus, the less number of times the composition is performed the better, i.e., Inc(2) is better than Inc(3), etc.
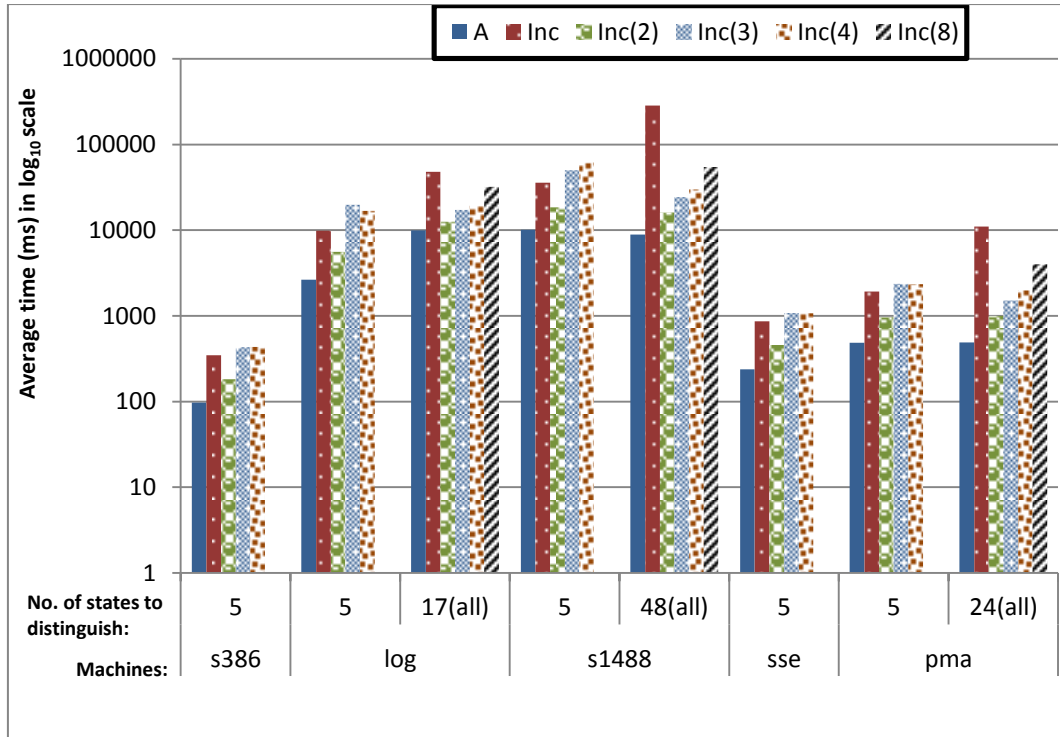
45

Figure 5-16: Average time versus number of states to distinguish when a DTC exists.

b) Results when there is no DTC: Machines that are used here have a small number of states, so the arrangements of algorithms for small machines will have some differences than those random machines. So for s386 and sse, they both have the same performance. We can see that from Figure 3-17, when $|b| = 5$ and there is no DTC for the set b, A, followed by Inc(2), followed by Inc, followed by Inc(3), and Inc(4). Inc(3) and Inc(4) both have comparable performance. When $b = S$, (A, Inc(2)) have comparable performance; followed by Inc(3), and Inc(4); they both have comparable performance, followed by Inc, then followed by Inc(8). We can see here that A is better than the Incremental algorithms because of the exhaustive use of composition operator in the incremental algorithms. For s1a, we can see that from Figure 5-17, when $|b| = 5$ and there is no DTC for the set b, Inc(4), inc(3), Inc, Inc(2) have the comparable performance; followed by A. When $b = S$, A, Inc(2), obtained comparable performance, followed by Inc(3), followed by Inc, Inc(8), followed by Inc(4). It is clear that more usage of composition operator means more execution time of the algorithm. In addition, the bigger the machine is, the more useful incremental algorithms are.
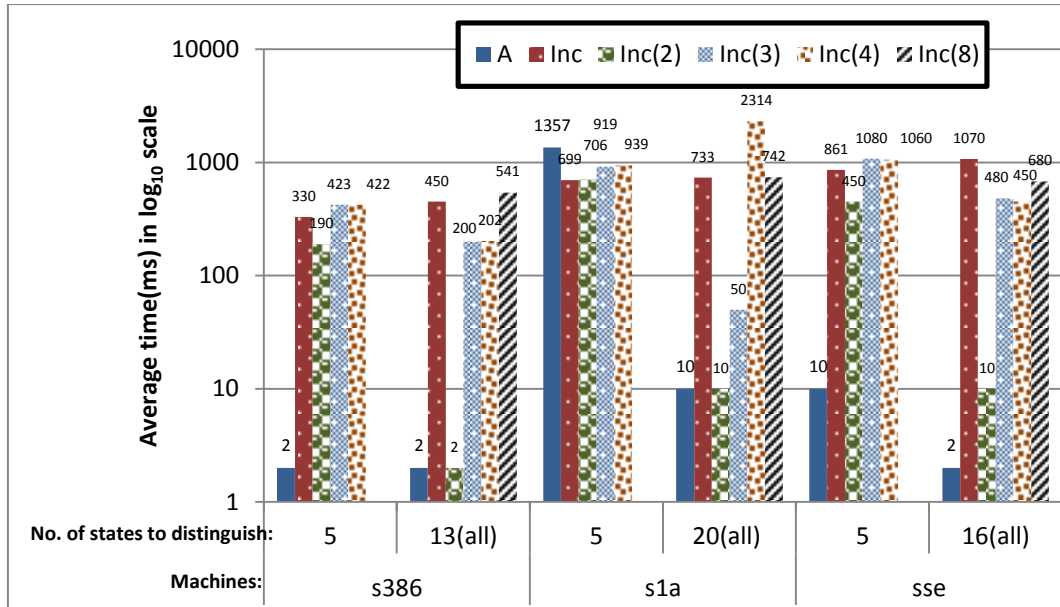
Figure 5-17: Execution time versus number of states to distinguish when there is no a DTC.

5.6.3 Execution time of A, H, and Hc algorithms:

Here we compare the execution time for the three algorithms A, H, and Hc.

i) Representative example: We will have two parts of representative results; results when we have DTC, and when there is no such DTC.

a) Results when a DTC exist: we use machine" log", with specifications $| S | = 17$, $| T | = 8704$, $| I | = 512$, $| O | = 17$, Figure 5-18 shows that H, Hc have the best performance.
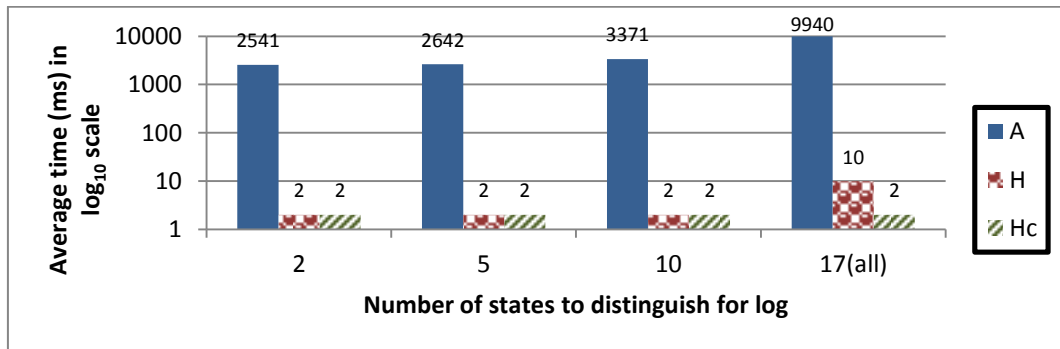

Figure 5-18: Average time in (ms) versus number of states that we want to distinguish when there is a DTC.

*b) Results when there is no DTC***:** We used machine"s1a" $| S | = 20$, $| T | =$ 5120, $| I | = 20$, $| O | = 256$. It gives us results with no sequence. Figure 5-19 shows that H, and Hc always have the best performance.
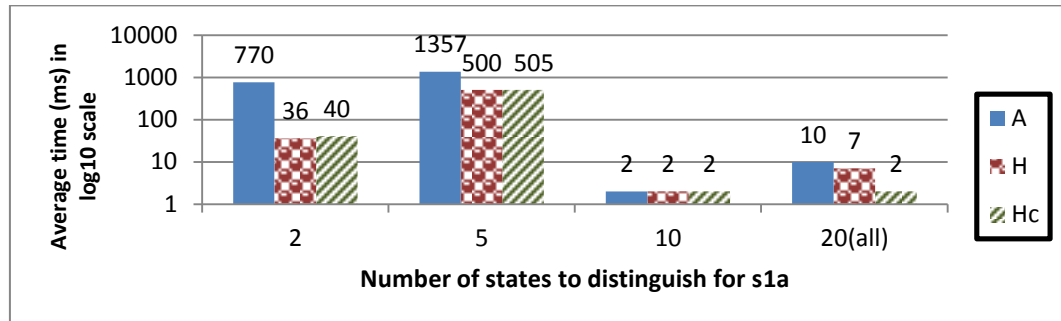
Figure 5-19: Average time in (ms) versus number of states that we want to distinguish when there is no a DTC.

*ii) Summary of all conducted experiments***:** In the following we report results over all machines with or without DTCs.

*a) Results when a DTC exists***:** As we can see here from Figure 5-20, H and Hc always outperform A when there is DTC.
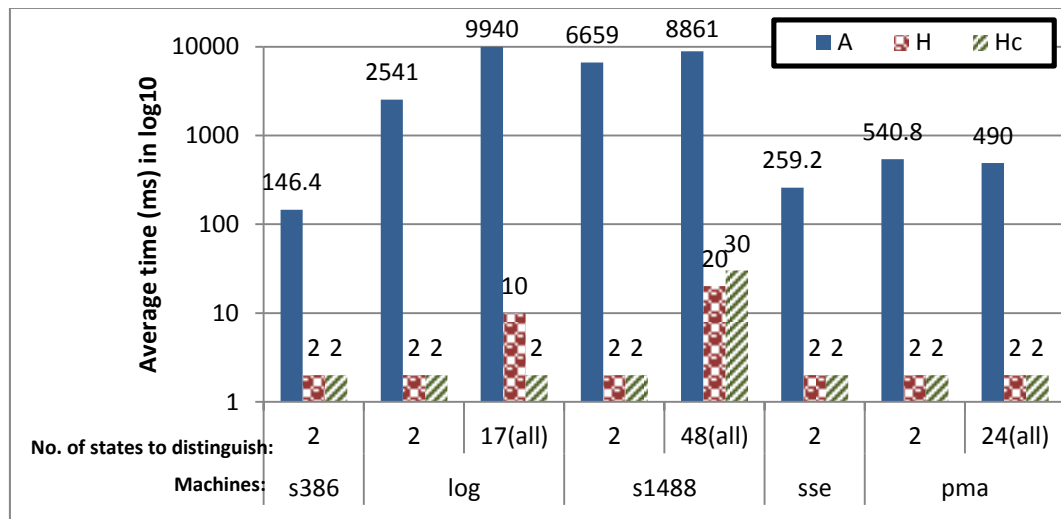
Figure 5-20: Average time versus number states to distinguish when there is a DTC.

b) Results when there is no DTC: From Figure 5-21, when $|b| = 2$ or $|b| = 5$ for s1a and there is no DTC for the set b, H and Hc outperform A, but for s386 they are all have the same performance. Also when $b = S$, A, H, and Hc can have the same performance as s386 and sse, or faster than A as s1a.

Figure 5-21: Average time in versus number to distinguish when there is no a DTC.

Chapter 6 : Conclusion and Summary of Obtained Results

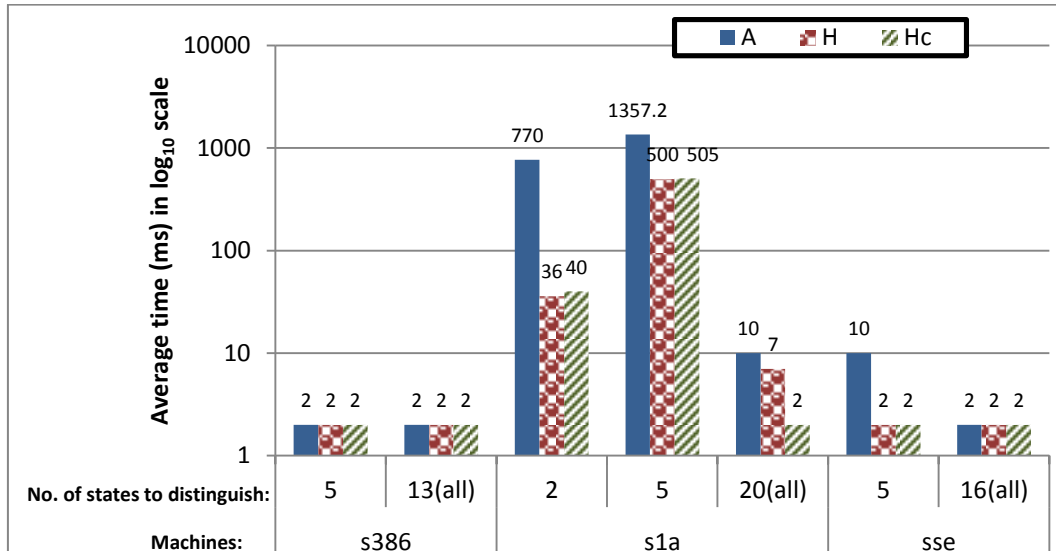This thesis targets the problem of deriving adaptive distinguishing test cases (DTCs) for a subset b of states of an observable nondeterministic finite state machine (FSM) with $|S|$ states. An incremental approach named Inc is proposed based on the non-incremental approach named A given in [39]. Inc considers the states of the set b incrementally one after the other while checking the existence of a DTC. Experiments are conducted to assess A in comparison with the proposed algorithm Inc and many of its implementations named Inc(k), $k = 2, 3, 4, 8$, where the subset b is partitioned into k increments.

In addition, in this thesis, an efficient heuristic approach, called H, is proposed for deriving DTCs. The approach searches for a solution using a special traversal of a successor tree based on some established rules and an appropriate use of hashing to speed up the search process. An implementation of the H approach, called Hc, that utilizes a cost function while conducting the search is also provided. Comprehensive experiments are conducted to assess and compare the performance of the proposed work. A summary of all the conducted experiments is reported below:

- A versus Inc: When $|b| = 5$ and a DTC exists, A and Inc(k) have comparable performance (execution time), followed by Inc. However, when a DTC does not exist, all Inc(k) have comparable performance, followed by A. When $|b| = |S|$ and a DTC exists, A and Inc(k) have comparable performance, followed by Inc. When a DTC does not exist, A, Inc(2), Inc(3), Inc(4) have comparable performance, followed by Inc, followed by Inc(8).

- H versus Hc for very large machines: When $|b| = 2$ and a DTC exists, Hc usually outperforms H in terms of execution time. However, when a DTC does not exist, both H and Hc have comparable execution time as both converge fast to a solution. When $|b| = S$ (if a DTC or it does not exist), H and Hc both are comparable as they converge very fast to a solution.

- In terms of quality of obtained solutions (length of obtained DTC) as assessed for small and medium size machines, all methods almost have comparable

performance. For large machines, in most cases, Hc outperforms H. Also, Hc enhances the quality of obtained solutions. When the number of states to distinguish increases, Hc provides solutions with better lengths than H.

- By construction, H and Hc might miss a solution even when a solution exists; however, according to the conducted experiments, it is shown that these methods are scalable to very large machines. On the contrary, A and Inc do not miss solutions, yet in general, for example when DTCs exist, these methods are not scalable to very large machines.

It is clear from the above analysis that there is no clear indication that using a particular method is always better than using the others. However, the above analysis helps a test engineer in selecting an appropriate method according the above analysis.

By construction, the incremental approach Inc(k) handles each of the subsets corresponding to the k partitions of the set b independently of each other. Thus, the development of parallel implementations of Inc(k) could be straightforward and it would be interesting to develop and assess such implementations considering recent state-of-the-art parallel technologies. In this case, the previous related work summarized in the Introduction section could be a good start for such a work.

References

[1]   D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines-a survey," *Proceedings of the IEEE,* vol. 84, no. 8, pp. 1090-1123, 1996.

[2]   T. Repasi, "Software testing - State of the art and current research challanges," *International Symposium on Applied Computational Intelligence and Informatics*, Timisoara, pp. 47-50, 2009.

[3]   G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*. John Wiley and Sons, 2013.

[4]   P. C. Jorgensen, *Software testing: a craftsman's approach*: CRC press, 2013.

[5]   A. Abdurazik, P. Ammann, W. Ding, and J. Offutt, "Evaluation of three specification-based testing criteria," in *Proceedings of the Sixth IEEE International Conference on Engineering of Complex Computer Systems*, pp. 179-187, 2000.

[6]   R. Lai, "A survey of communication protocol testing," *Journal of Systems and Software*, vol. 62, no. 1, pp. 21-46, 2002.

[7]   G. V. Bochmann and A. Petrenko, "Protocol testing: review of methods and relevance for software testing," in *Proceedings of the 1994 ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 109-124, 1994.

[8]   R. Dorofeeva, K. El-Fakih, S. Maag, A. R. Cavalli, and N. Yevtushenko, "FSM-based conformance testing methods: a survey annotated with experimental evaluation," *Information and Software Technology*, vol. 52, no. 12, pp. 1286-1297, 2010.

[9]   A. Benharref, R. Dssouli, M. A. Serhani, A. En-Nouaary, and R. Glitho, "New approach for EFSM-based passive testing of web services," *Testing of Software and Communicating Systems*, pp. 13-27, 2007.

[10]   S. Hallé and R. Villemaire, "Runtime monitoring of web service choreographies using streaming XML," in *Proceedings of the ACM symposium on Applied Computing*, pp. 2118-2125, 2009.

[11]   G. Morales, S. Maag, A. Cavalli, W. Mallouli, E. M. De Oca, and B. Wehbi, "Timed extended invariants for the passive testing of web services," in *IEEE International Conference on Web Services (ICWS)*, pp. 592-599, 2010.

[12]   J. Simmonds, "Dynamic analysis of webservices," Ph.D. thesis Graduate Department of Computer Science, University of Toronto, 2011.

[13]   M. Haydar, A. Petrenko, and H. Sahraoui, "Formal verification of web applications modeled by communicating automata," in *International Conference on Formal Techniques for Networked and Distributed Systems*, pp. 115-132, 2004.

[14]   T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE Transactions on Software Engineering*, vol. 4, no. 3, pp.178-187, 1978.

[15]   A. D. Friedman and P. R. Menon, *Fault Detection in Digital Circuits*. Prentice Hall, 1971.

[16]   A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: principles, techniques, and tools*. Addison-Wesley Longman Publishing Co., Inc., p. 796, 1986.

[17]   F. Belli, "Finite state testing and analysis of graphical user interfaces," in *Proceedings of the 12th International Symposium on Software Reliability Engineering*, pp. 34-43, 2001.

[18] I. Burdonov, A. Kossatchev, A. Petrenko, and D. Galter, "Kvest: Automated generation of test suites from formal specifications," in *FM'99—Formal Methods*, ed: Springer, pp. 608-621, 1999.

[19] R. Binder, *Testing object-oriented systems: models, patterns, and tools*: Addison-Wesley Professional, 2000.

[20] Y. Dong, Z. Li, Y. Cheng, and H. Zhao, "A model driven testing solution for embedded system with Simulink/stateflow model," *in Proceedings of the 2nd International Conference on Trustworthy Systems and Their Applications*, pp. 24-29, 2015.

[21] K. El-Fakih and N. Yevtushenko, "Test translation for embedded finite state machine components," *The Computer Journal*, vol. 59, pp. 1805-1816, 2016.

[22] W. Grieskamp, N. Kicillof, K. Stobie, and V. Braberman, "Model-based quality assurance of protocol documentation: tools and methodology," *Softw. Test. Verif. Reliab.,* vol. 21, no. 1, pp. 55-71, 2011.

[23] E. F. Moore, "Gedanken-experiments on sequential machines," *Automata studies*, Priceton Univeristy Press, volume 34, pp. 129-153, 1956.

[24] R. Alur, C. Courcoubetis, and M. Yannakakis, "Distinguishing tests for nondeterministic and probabilistic machines," *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pp. 363-372, 1995.

[25] A. Gill, "State-identification experiments in finite automata," *Information and Control,* volume 4, issues 2-3, pp. 132-154, 1961.

[26] C. Güniçen, K. Inan, U. C. Türker, and H. Yenigün, "The relation between preset distinguishing sequences and synchronizing sequences," *Formal Aspects of Computing,* vol. 26, pp. 1153-1167, 2014.

[27] C. Güniçen, G.-V. Jourdan, and H. Yenigün, "Using Multiple Adaptive Distinguishing Sequences for Checking Sequence Generation," in *Testing Software and Systems*, ed: Springer, pp. 19-34, 2015.

[28] R. M. Hierons and U. C. Türker, "Distinguishing sequences for partially specified FSMs," in *NASA Formal Methods*, pp. 62-76, 2014.

[29] Z. Kohavi, *Switching and finite automata theory*. New York: McGraw-Hill, 1978.

[30] N. Kushik, K. El-Fakih, and N. Yevtushenko, "Preset and adaptive homing experiments for nondeterministic finite state machines," *Proceedings of the Sixteenth International Conference on Implementation and Application of Automata*, Lecture Notes in Computer Science 6807, pp. 215-224, 2011.

[31] N. Kushik, K. El-Fakih, N. Yevtushenko, and A. R. Cavalli, "On adaptive experiments for nondeterministic finite state machines," *Int. J. Softw. Tools Technol. Transf.,* vol. 18, no. 3, pp. 251-264, 2016.

[32] D. Lee and M. Yannakakis, "Testing finite-state machines: State identification and verification," *IEEE Transactions on Computers*, volume 43, issue 3, pp. 306-320, 1994.

[33] A. Mathur, *A Foundations of Software Testing.* Addison Wesley, 2008.

[34] N. Spitsyna, K. El-Fakih, and N. Yevtushenko, "Studying the separability relation between finite state machines," *Software Testing, Verification and Reliability*, volume 17, issue 4, pp. 227-241, 2007.

[35] U. C. Türker and H. Yenigün, "Hardness and inapproximability of minimizing adaptive distinguishing sequences," *Formal Methods in System Design,* vol. 44, pp. 264-294, 2014.

[36]   U. C. Türker, T. Ünlüyurt, and H. Yenigün, "Effective algorithms for constructing minimum cost adaptive distinguishing sequences," *Information and Software Technology,* vol. 74, pp. 69-85, 2016.

[37]   A. Bianco and L. De Alfaro, "Model checking of probabilistic and nondeterministic systems," in *Foundations of Software Technology and Theoretical Computer Science*, pp. 499-513, 1995.

[38]   R. M. Hierons, "Testing from a nondeterministic finite state machine using adaptive state counting," *IEEE Transactions on Computers,* vol. 53, pp. 1330-1342, 2004.

[39]   K. El-Fakih, N. Yevtushenko, and N. Kushik, "Adaptive distinguishing experiments for nondeterministic finite state machines: Test Case Derivation and Constructive Approach for Establishing the Tight Upper Bound," Submitted to *Formal Aspects of Computing Journal*, Submission Date: 14/1/2017.

[40]   M. N. Sokolovskii, "Diagnostic experiments with automata," *Cybernetics and Systems Analysis,* vol. 7, pp. 988-994, 1971.

[41]   K. El-Fakih, A. R. Haddad, N. Aleb, and N. Yevtushenko, "Heuristics for deriving distinguishing experiments of nondeterministic finite state machines," *Applied Soft Computing,* vol. 49, pp. 1175-1184, 2016.

[42]   R. M. Hierons and U. Turker, "Parallel algorithms for generating distinguishing sequences for observable non-deterministic FSMs," *ACM Transactions on Software Engineering and Methodology*, vol. 26, pp. 1-37, 2017

[43]   K. El-Fakih, G. Barlas, M. Ali, and N. Yevtushenko, "Parallel algorithms for reducing derivation time of distinguishing experiments for nondeterministic finite state machines," *International Journal of Parallel, Emergent and Distributed Systems*, pp. 1-14, 2017.

[44]   A. Petrenko, N. Yevtushenko, "Adaptive testing of deterministic implementations specified by nondeterministic FSMs," *International Conference on Testing Software and Systems,* pp. 162-178, 2011.

[45]   F. Brglez. (2017, Jan. 15). *ACM/SIGMOD benchmark dataset*. Available: http://www.cbl.ncsu.edu/benchmarks/Benchmarks-upto-1996.html

Vita

Ayat was born on January 26, 1988, in Saudi Arabia. She studied at Al Najah National University in Nablus, Palestine, where she graduated in 2010 with a Bachelor's Degree in Computer Engineering.

Mrs. Ayat worked as a PHP programmer for a year. After that, she moved to the United Arab Emirates in 2012, where she worked as a computer engineer at Sharjah American International School (SAIS) for a year and half. Then she joined the Master's program in Computer Engineering at the American University of Sharjah.