

**THE UPPER BOUND FOR THE LENGTH OF THE SHORTEST
HOMING SEQUENCES**

by
BERK CIRISCI

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfilment of
the requirements for the degree of Master of Science

Sabanci University
May 2019

THE UPPER BOUND FOR THE LENGTH OF THE SHORTEST
HOMING SEQUENCES

Approved by:

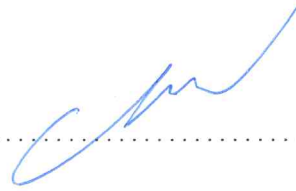
Assoc. Prof. Dr. Hüsnü Yenigün
(Thesis Supervisor)



Asst. Prof. Dr. Kamer Kaya



Asst. Prof. Dr. Uraz Cengiz Türker



Date of Approval: July 19, 2019

BERK CIRISCI 2019 ©

All Rights Reserved

ABSTRACT

THE UPPER BOUND FOR THE LENGTH OF THE SHORTEST HOMING SEQUENCES

BERK CIRISCI

COMPUTER SCIENCE AND ENGINEERING M.A. THESIS, MAY 2019

Thesis Supervisor: Assoc. Prof. Dr. Hüsnü Yenigün

Keywords: Finite State Machines, Homing Sequences, Isomorphic FSM, Hibbard's
Upper Bound

Homing sequences are special input sequences that are used by various techniques of finite state machine based testing. Using a shorter homing sequence is typically preferred since it would yield a shorter test sequence. Finding a shortest homing sequence is known to be an NP-hard problem. The upper bound of shortest homing sequences is also a problem studied in the literature. A tight upper bound for the length of shortest homing sequence for a finite state machine with n states is known to be $n(n-1)/2$. However, the known examples of finite state machines hitting to this upper bound also use $n-1$ input symbols, i.e. the size of the input alphabet also grows with the number of states. Is this upper bound reachable for a finite state machine with a constant number of inputs? In this work, we use an experimental analysis and we answer this question negatively. By exhaustively enumerating all finite state machines with two input symbols and two output symbols, we experimentally compute the upper bound for the length of the shortest homing sequence for finite state machines with 10 or less states. In order to make this computation feasible in practice, we apply several techniques to eliminate from our search those finite state machines which would not affect the result of the computation.

ÖZET

EN KISA ÖZGÜDÜM DİZİLERİNİN UZUNLUĞUNUN ÜST SINIRI

BERK ÇİRİŞCİ

BİLGİSAYAR MÜHENDİSLİĞİ VE BİLİMİ YÜKSEK LİSANS TEZİ, TEMMUZ
2019

Tez Danışmanı: Doç. Dr. Hüsnü Yenigün

Anahtar Kelimeler: Sonlu Durum Makineleri, Özgüdümlü Dizileri, Eşbiçimli Sonlu Durum Makinesi, Hibbard'ın Üst Sınırı

Özgüdümlü dizileri, çeşitli sonlu durum makinesi bazlı testlerde kullanılan ilginç girdi dizilerindedir. Daha kısa özgüdümlü dizileri kullanmak, daha kısa test dizileri sağlayacağı için genellikle tercih edilir. En kısa özgüdümlü dizisini bulmanın NP-zor bir problem olduğu bilinmektedir. En kısa özgüdümlü dizisinin üst sınırı da literatürde çalışılan bir problemdir. n durumlu bir sonlu durum makinesi için sıkı üst sınırın $n(n-1)/2$ olduğu bilinmektedir. Bununla birlikte, bu sınıra ulaşan sonlu durum makinelerinin bilinen bütün örneklerinin hepsi $n-1$ girdi sembolu kullanmaktadır ve bu durum, girdi alfabesi durum sayısı ile birlikte büyüyor demektir. Peki bu üst sınıra sabit sayıda girdili bir sonlu durum makinesi ile ulaşılabilir mi? Bu çalışmada deneysel bir analiz yaptık ve soruya negatif bir şekilde cevap verdik. Bütün 2 girdili, 2 çıktılı sonlu durum makinelerini etraflıca sayıp, deneysel olarak 10 ya daha az durumlu sonlu durum makineleri için en kısa özgüdümlü dizisinin üst sınırını hesapladık. Bu hesaplamayı pratikte uygulanabilir kılmak adına sonucu etkilemeyen sonlu durum makinelerini elemek için çeşitli teknikler uyguladık.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my thesis advisor, Husnu Yenigun. Without his knowledge, patience, kind and encouraging personality I would not be at this stage of my career to defend this thesis of mine and continue to this academic path as a Ph.D. student. I'm so honored to work with him and I wish someday I can work with students who admire me as much as I admire him.

I also would like to thank Kamer Kaya for his support who always tried to help whenever he can. I am very thankful for his patience to my numerous questions and his kind-hearted answers. I'm so glad that I took both courses of Kemal Inan who was the first person makes me love my field and subject.

Even though I know that I can't remunerate their efforts for making me who I am today, I want to present my thanks with all my heart to my parents; my mother and my father. I hope I can be supportive and caring parents for my children as they are. I would like to thank to my grandmothers for their helps when I was raising to become this person who writes these words. Rest in peace grandma.

Finally, I would like to thank all my friends, whoever stayed with me as a fellow traveler on this exciting journey. I really appreciate your existence whenever I need you. I hope we can always be supportive to each other for our entire lives. Cheers.

*to my mother and father
anne ve babama*

TABLE OF CONTENTS

LIST OF TABLES	ix
LIST OF FIGURES	x
1. INTRODUCTION	1
2. PRELIMINARIES	6
3. EXHAUSTIVE NON-ISOMORPHIC FSM GENERATION	14
3.1. Non-isomorphic Unary Automaton Selection	15
3.2. Non-isomorphic Binary Automaton Generation	17
3.3. Non-isomorphic FSM Generation	19
4. EXPERIMENTS	26
5. CONCLUSION & FUTURE WORK	37
BIBLIOGRAPHY	39

LIST OF TABLES

Table 3.1. Usage of output functions in experiments according to their types to generate binary FSMs	23
Table 4.1. Number of all and non-isomorphic FSMs according to their number of states and inputs (Harary & Palmer, 2014)	27
Table 4.2. Experimental results for number of eliminated automata and FSMs according to their number of states	28
Table 4.3. Experimental results for shortest homing sequences according to number of states	29

LIST OF FIGURES

Figure 1.1. An FSM M hits to Hibbard's Bound	3
Figure 2.1. An automaton A_0 and two FSMs M_0, M_1	8
Figure 2.2. An FSM M'	11
Figure 2.3. The Homing tree of M' given in Figure 2.2	12
Figure 3.1. Two unary automata $A, B \in \mathcal{U}$ and a binary automaton $C = A \oplus B$	18
Figure 3.2. A unary automaton A and an isomorphic automaton A_π where $\pi(0) = 2, \pi(1) = 0, \pi(2) = 1$	19
Figure 3.3. Two unary FSMs $M_A, M_B \in \mathcal{M}$ and a binary FSM $M_C = M_A \oplus M_B$	20
Figure 3.4. A unary FSM M and an isomorphic FSM M_π where $\pi(0) = 2, \pi(1) = 0, \pi(2) = 1$	21
Figure 3.5. Structure of FSM hits to upper bound when $p = 1$	24
Figure 4.1. Two unary FSMs $M_A, M_B \in \mathcal{M}$ and a binary FSM $M_C = M_A \oplus M_B$ which hits to $H(4, 2, 2)$	29
Figure 4.2. Two unary FSMs $M_A, M_B \in \mathcal{M}$ and a binary FSM $M_C = M_A \oplus M_B$ which hits to $H(5, 2, 2)$	30
Figure 4.3. Two unary FSMs $M_A, M_B \in \mathcal{M}$ and a binary FSM $M_C = M_A \oplus M_B$ which hits to $H(6, 2, 2)$	31
Figure 4.4. Two unary FSMs $M_A, M_B \in \mathcal{M}$ and a binary FSM $M_C = M_A \oplus M_B$ which hits to $H(7, 2, 2)$	32
Figure 4.5. Two unary FSMs $M_A, M_B \in \mathcal{M}$ and a binary FSM $M_C = M_A \oplus M_B$ which hits to $H(8, 2, 2)$	33

1. INTRODUCTION

Testing is the most widely used method for system validation by the industry. In practice, testing is usually applied manually in an ad hoc manner. Such an approach is very expensive and it itself is open to errors. Therefore, many systematic and automated methods are proposed in the literature for testing.

Model Based Testing (MBT) (Broy, Jonsson, Katoen, Leucker & Pretschner, 2005) is one such approach, where the requirements of the system are specified by using a model. When this specification model is given in a formal notation, it can be used to generate test cases automatically. For the specification of interactive systems, usually state-based models, such as State-Charts (Harel & Politi, 1998) or Finite State Machines (FSM) (Kohavi, 1978), are used.

When the abstract behavior of an interactive system is modeled by using an FSM, there are various methods to construct a test sequence from this FSM model (Chow, 1978; Gonenc, 1970; Hennine, 1964; Hierons & Ural, 2006; Lee & Yannakakis, 1996; Moore, 1956; Simao & Petrenko, 2010; Ural, Wu & Zhang, 1997). These methods construct a test sequence, called a *checking sequence*, which gives 100% fault coverage under certain assumptions, such as an upper bound on the number of states of the implementation.

These methods construct checking sequences by using some special sequences. These special sequences are typically used to identify the states of the implementation. For instance, distinguishing sequences or characterizing sets used in a checking sequence identify the initial state. In other words, when a distinguishing sequence is applied in a checking sequence, by looking at the output sequence produced by the implementation as a response to the application of the distinguishing sequence, one can tell the state of the implementation *before* the application of the sequence.

On the other hand, synchronizing sequences and homing sequences are used to identify the final state of the implementation. In other words, when a homing sequence is applied, by looking at the output sequence produced by the implementation as a response to this homing sequence, one can tell the state of the implementation

reached *after* the application of the sequence.

The checking sequence construction methods make use of these special sequences. Therefore, the existence of these special sequences and the length of these special sequences are important for the applicability and the scalability of the checking sequence construction methods. The following results are known for these special sequences¹.

Both preset and adaptive distinguishing sequences are considered in the literature. For preset distinguishing sequence, the existence check problem is PSPACE-complete (Lee & Yannakakis, 1994), whereas the existence check for an adaptive distinguishing sequence can be handled in time $O(pn \log n)$ time. Here n and p are the number of states and the number of input symbols of the FSM, respectively. Upper bounds are also known for the length of preset and adaptive distinguishing sequences. For preset distinguishing sequences, this upper bound is exponential. There are FSMs where the length of the shortest preset distinguishing sequence is exponential (e.g. see Theorem 2.1 of (Lee & Yannakakis, 1994) and Theorem 2.11 of (Krichen, 2005)).

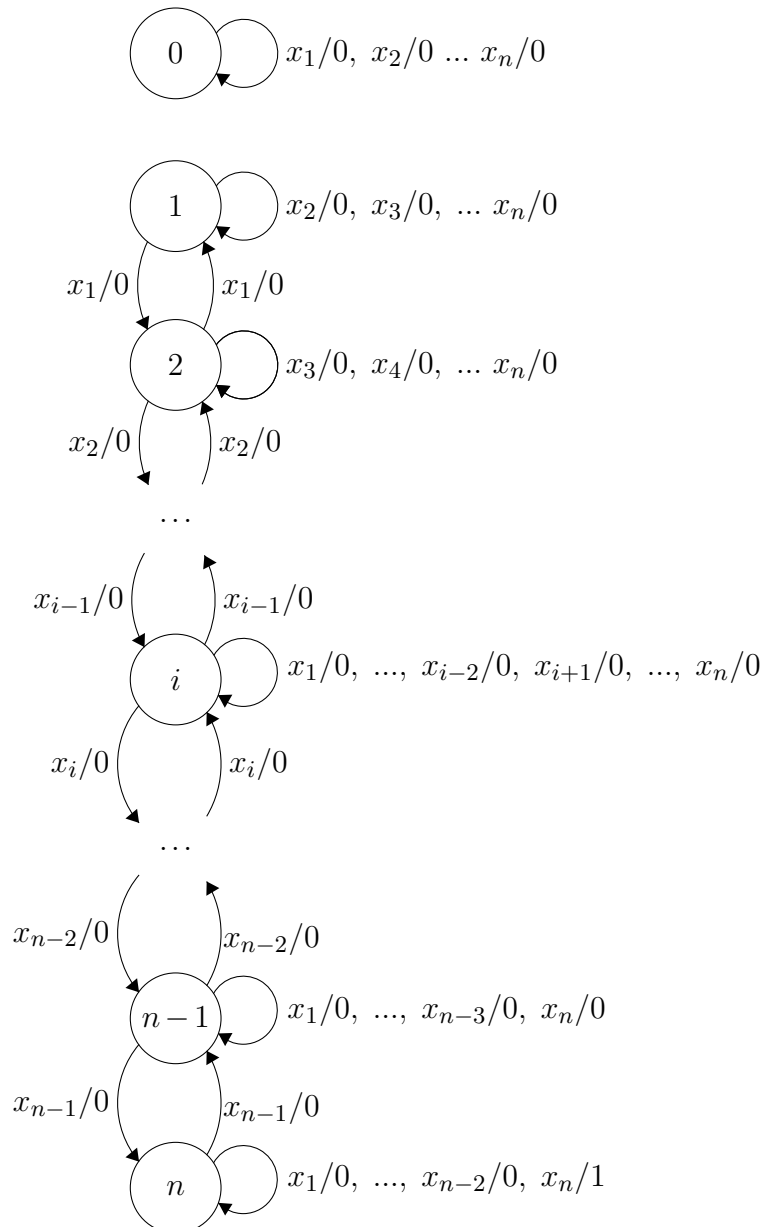
There is a wide literature for synchronizing sequences, possibly because of the existence of an interesting open problem in the field. Firstly, checking the existence of a synchronizing sequence can be handled in time $O(pn^2)$ (Eppstein, 1990). Although finding shortest synchronizing sequence is an NP-hard problem (Eppstein, 1990), computing a synchronizing sequence is a polynomial time problem, for which several algorithms exist (see e.g. (Eppstein, 1990; Kudłacik, Roman & Wagner, 2012; Roman, 2009; Roman & Szykuła, 2015; Trahtman, 2004)). The interesting open problem about synchronizing sequences is related to the upper bound of the shortest synchronizing sequences. The well-known *Černý conjecture* claims that the length of a shortest synchronizing sequence is at most $(n - 1)^2$ for an FSM with n states (Černý, 1964; Černý, Pirická & Rosenauerová, 1971). If this conjectured upper bound is correct, it is also known to be tight since there are FSMs with shortest synchronizing sequence of length $(n - 1)^2$.

For the class of FSMs¹ considered in this work, there always exists a homing sequence. Finding a shortest homing sequence is known to be NP-hard (Eppstein, 1990; Sandberg, 2005). Unlike synchronizing sequences, there is not much work on the computation of homing sequences. There is only a recent work (Çirisci, Emek, Sorguç, Kaya & Yenigün, 2019) where the authors actually use/adapt synchronizing sequence construction algorithms for computing homing sequences.

¹These results given here apply to the class of deterministic, complete, and minimal FSMs. Please refer to Section 2 for the formal definition of these terms.

The upper bound for the length of shortest homing sequences, which is the main topic of this work, is also known. (Hibbard, 1961) showed that, for an FSM with n states, the length of the shortest homing sequence can be at most $n(n-1)/2$. We will call this upper bound as *the Hibbard bound*. The Hibbard bound is also known to be tight. Hence there are FSMs hitting to the Hibbard bound, i.e. FSMs where the length of the shortest homing sequence is equal to $n(n-1)/2$. You can find the corresponding FSM in the Figure 1.1.

Figure 1.1 An FSM M hits to Hibbard's Bound



The Hibbard bound expression does not depend on the number of input symbols of the FSM. For the FSMs hitting to the Hibbard bound (see Figure 1.1), on the other hand, the number of input symbols is the same as the number of states of the FSM. It is possible to ask the following question at this point: Is there an FSM with a

constant number of input symbols which hits to the Hibbard bound?

In this work, we attempt to answer this question and we start from the simplest possible form of it, i.e. we ask the following question: Is there an FSM with two input symbols and two output symbols (which we will call as *binary FSMs*), that hits to the Hibbard bound?

We answer this question negatively, i.e. no binary FSM hits to the Hibbard bound. In this case, the next question is then the following: What is the upper bound for the length of the shortest homing sequence of binary FSMs? In this work we also attempt to answer this question by using an experimental approach.

In order to compute the upper bound of the length of the shortest homing sequence for binary FSMs with n states, we essentially enumerate all binary FSMs with n states, and compute the shortest homing sequence of each FSM considered. For the number of states 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 (respectively), we found the upper bound for the length of the shortest homing sequences to be 0, 1, 3, 6, 9, 13, 18, 24, 31, 38 (respectively), an integer sequence which does not exist in the OEIS library (OEIS, 2019).

Note that, this experimental study is computationally challenging. If we consider FSMs with n states, p input symbols, and o output symbols, there are $(no)^{(np)}$ FSMs, not considering the isomorphism. Even when we restrict ourselves to binary FSMs with 10 states, two input symbols, and two output symbols (the largest FSM size considered in our study), there are 20^{20} such FSMs! Several theoretical and practical techniques are applied in our experimental study to reduce the number of FSMs taken into account (without affecting the outcome of the computation for the upper bound) and to speed up the computation.

Suppose that we simply rename the states and/or the input symbols and/or the output symbols of an FSM M to get another FSM M' . FSMs M and M' that can be obtained from one another by such a renaming are called *isomorphic*. The answer to the homing sequence related problems (such as the existence of a homing sequence, the length of the shortest homing sequence, etc.) will be the same for isomorphic FSMs. A great deal of FSMs can be eliminated in our search, if we can guarantee to consider at least one FSM from each isomorphism class.

A similar experimental work (Kisielewicz & Szykuła, 2013) exists in the literature which enumerates all automata (not FSMs), by considering these isomorphism classes. (Kisielewicz & Szykuła, 2013) performs this experimental study to verify/falsify the Černý conjecture mentioned above. We follow the approach of (Kisielewicz & Szykuła, 2013) to generate non-isomorphic automata as much as

possible, and we construct FSMs from the generated automata. We also employ a multi-core parallel computation approach to speed-up the computation.

The rest of the thesis is organized as follows. Section 2 introduces the notation we use throughout the thesis and gives background information. Section 3 explains how we generate all non-isomorphic binary FSMs with a given number of states. We provide the theoretical results yielding the conservative reductions in this section. The experimental study that we performed is given in detail in Section 4. In Section 5, we conclude the paper and provide some future directions for our work.

2. PRELIMINARIES

A *Deterministic Finite Automaton (DFA)* (or simply an *automaton*) is a triple $A = (S, X, \delta)$ where S is a finite set of *states*, X is a finite set of *alphabet* (or *input*) symbols, and $\delta : S \times X \rightarrow S$ is a *transition function*. When δ is a total (resp. partial) function, A is called *complete* (resp. *partial*). In this work we only consider complete DFAs unless stated otherwise.

A *Deterministic Finite State Machine (FSM)* is a tuple $M = (S, X, Y, \delta, \lambda)$ where S is a finite set of *states*, X is a finite set of *alphabet* (or *input*) symbols, Y is a finite set of *output symbols*, $\delta : S \times X \rightarrow S$ is a *transition function*, and $\lambda : S \times X \rightarrow Y$ is an *output function*. In this work, we always consider *complete* FSMs, which means the functions δ and λ are total functions.

Note that given an automaton $A = (S, X, \delta)$, one can extend A by using a set of output symbols Y and an output function $\lambda : S \times X \rightarrow Y$ to obtain an FSM $M = (S, X, Y, \delta, \lambda)$ and this extension is represented as $M = A \uplus \lambda$. Reversely, each FSM M has an underlying automaton. The automaton of an FSM $M = (S, X, Y, \delta, \lambda)$ will be denoted as $M|_A$ where we simply have $M|_A = (S, X, \delta)$. Hence, for an FSM $M = (S, X, Y, \delta, \lambda)$ we have $M = M|_A \uplus \lambda$.

Two DFAs $A = (S, X, \delta)$ and $A' = (S', X', \delta')$ are called *isomorphic* if there exist bijections $f : S \rightarrow S'$ and $g : X \rightarrow X'$ such that $\forall x \in X$ and $\forall s \in S, f(\delta(s, x)) = \delta'(f(s), g(x))$. Intuitively, A and A' are isomorphic, if one can simply rename the states and input symbols of A to obtain A' . A and A' are called *state-isomorphic* if A and A' isomorphic when g is taken as the identity function, which implies A and A' share the same alphabet. In this case, renaming only the states of A (but keeping the input symbols unchanged) is sufficient to get A' . A and A' are called *input-isomorphic* if A and A' isomorphic when f is taken as the identity function, which implies A and A' share the set of states. In this case, renaming only the input symbols of A (but keeping the state names unchanged) is sufficient to get A' .

Two FSMs $M = (S, X, Y, \delta, \lambda)$ and $M' = (S', X', Y', \delta', \lambda')$ are called *isomorphic* if there exist bijections $f : S \rightarrow S'$, $g : X \rightarrow X'$, and $h : Y \rightarrow Y'$ such that $\forall x \in X$ and

$\forall s \in S, f(\delta(s, x)) = \delta'(f(s), g(x))$ and $h(\lambda(s, x)) = \lambda'(f(s), g(x))$. Intuitively, M and M' are isomorphic, if one can simply rename the states, input symbols, and output symbols of M to obtain M' . M and M' are called *state-isomorphic* if M and M' isomorphic when g and h are taken as the identity functions, which implies M and M' share the same input alphabet and the same set of output symbols. In this case, renaming only the states of M (but keeping the input symbols and the output symbols unchanged) is sufficient to get M' . M and M' are called *input-isomorphic* if M and M' isomorphic when f and h are taken as the identity function, which implies M and M' are defined over the same set of states and the same of output symbols. In this case, renaming only the input symbols of M (but keeping the state names and the output symbols unchanged) is sufficient to get M' . Finally, M and M' are called *output-isomorphic* if M and M' isomorphic when f and g are taken as the identity function, which implies M and M' are defined over the same set of states and the same input alphabet. In this case, renaming only the output symbols of M (but keeping the state names and the input symbols unchanged) is sufficient to get M' .

An automaton and an FSM can be visualized as a graph, where the states correspond to the nodes and the transitions correspond to the edges of the graph. For an automaton the edges of the graph are labeled by input symbols, whereas for an FSM the edges are labeled by an input and an output symbol. In Figure 2.1a and Figure 2.1b, an example automaton and an example FSM are given.

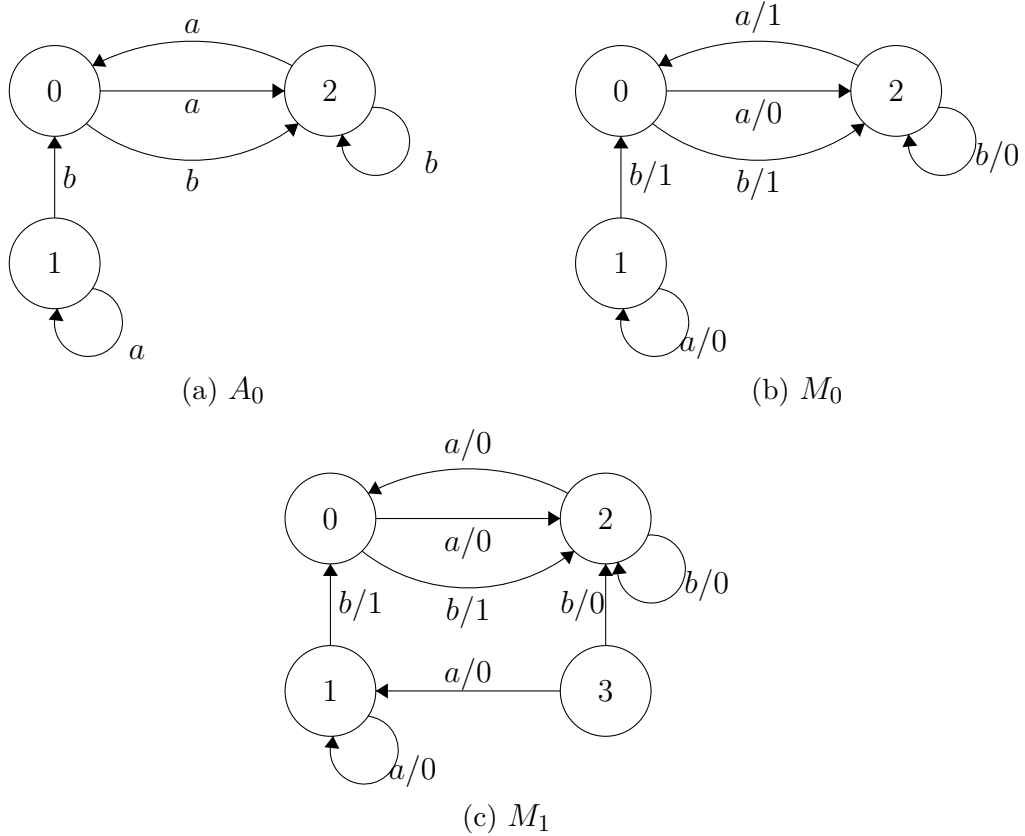
$|Z|$ expresses the number of the elements in a given set Z , i.e. the cardinality of Z . For the cardinality of certain components of FSMs, we will use the following symbols consistently throughout this thesis:

- The number of states $|S|$ will be represented as n .
- The number of inputs $|X|$ will be represented as p .
- The number of outputs $|Y|$ will be represented as o .

In this work, we will consider FSMs with $o = 2$. We will typically consider the set of output symbols as $Y = \{0, 1\}$. In other words, the outputs labeling the transitions will either be a 0 or a 1. Given a set of input symbols $X' \subseteq X$, we use $C_{X'}$ as the number of 1s seen in the output labels of the transitions with input symbols in X' . More formally $C_{X'} = |\{s \in S | \lambda(s, x) = 1, x \in X'\}|$. Now we will introduce some special subsets of output functions as follows:

- If $\lambda(s, x) = 0$ for all $s \in S, x \in X' \subseteq X$ (i.e. $C_{X'} = 0$), we call such an output function λ as *all-zeroes for X'* . When it is clear from the context, we call an

Figure 2.1 An automaton A_0 and two FSMs M_0, M_1



output function which is all-zeroes for X' simply as *all-zeroes*. In Figure 2.1c, the output function of M_1 is all-zeroes for $\{a\}$ since $C_{\{a\}} = 0$.

Note that, considering output-isomorphism, an all-zeroes for X' output function means every transition with input $x \in X'$ has the same output (not necessarily the output 0).

- If $\lambda(s, x) = 0$ for all $s \in S$ and $x \in X' \subset X$, except for just one pair $(s', x') \in S \times X'$, $\lambda(s', x') = 1$ (i.e. $C_{X'} = 1$), then λ is called as *single-one for X'* . Again, when it is clear from the context, we call an output function which is single-one for X' simply as *single-one*.

In other words, for an output function which is single-one for X' , every transition with input $x \in X'$ has the same output except for one of them. In Figure 2.1b, the output function of M_0 is a single-one for $\{a\}$ since $C_{\{a\}} = 1$.

- If an output function is not all-zeroes for X' and it is not single-one for X' , then it will be called as *multi-one for X'* (or simply *multi-one* when X' is clear from the context).

In Figure 2.1c, output function of M_1 is multi-one for $\{b\}$ since $C_{\{b\}} = 2$.

An *input sequence* (or a *word*) $\bar{x} \in X^*$ is a concatenation of zero or more input symbols. More formally, an *input sequence* \bar{x} is a sequence of input symbols $x_1x_2\dots x_k$ for some $k \geq 0$ where $x_1, x_2, \dots, x_k \in X$. As can be seen from the definition, an input sequence may have no symbols; in this case it is called the *empty sequence* and denoted by ϵ . We use the notation $x^\ell = xx\dots x$ to denote an input sequence consisting of ℓ copies of the input symbol $x \in X$.

For both automata and FSMs, the transition function δ is extended to input sequences as follows. For a state $s \in S$, an input sequence $\bar{x} \in X^*$ and an input symbol $x \in X$, we let $\bar{\delta}(s, \epsilon) = s$, $\bar{\delta}(s, x\bar{x}) = \bar{\delta}(\delta(s, x), \bar{x})$. Similarly, the output function of FSMs is extended to input sequences as follows: $\bar{\lambda}(s, \epsilon) = \epsilon$, $\bar{\lambda}(s, x\bar{x}) = \lambda(s, x)\bar{\lambda}(\delta(s, x), \bar{x})$. By abusing the notation we will continue using the symbols δ and λ for $\bar{\delta}$ and $\bar{\lambda}$, respectively.

Finally for both automata and FSMs, the transition function δ is extended to a set of states as follows. For a set of states $S' \subseteq S$ and an input sequence $\bar{x} \in X^*$, $\delta(S', \bar{x}) = \{\delta(s, \bar{x}) \mid s \in S'\}$.

Given an FSM $M = (S, X, Y, \delta, \lambda)$ and two states $s_i, s_j \in S$, an input sequence $\bar{x} \in X^*$ is said to *separate* s_i and s_j if $\lambda(s_i, \bar{x}) \neq \lambda(s_j, \bar{x})$. In this case, \bar{x} is called a *separating sequence* for s_i and s_j .

Given an FSM $M = (S, X, Y, \delta, \lambda)$ and a subset of states $S' \subseteq S$, an input sequence $\bar{x} \in X^*$ is said to *separate* S' , if there exist two states $s_i, s_j \in S'$ such that \bar{x} separates s_i and s_j .

An FSM $M = (S, X, Y, \delta, \lambda)$ is said to be *minimal* if for any two different states $s_i, s_j \in S$, there exists a separating sequence for s_i and s_j .

Definition 1. For an FSM $M = (S, X, Y, \delta, \lambda)$ and a subset of states $S' \subseteq S$, a *Homing Sequence (HS)* for S' is an input sequence $\bar{x} \in X^*$ such that for all states $s_i, s_j \in S'$, $\lambda(s_i, \bar{x}) = \lambda(s_j, \bar{x}) \implies \delta(s_i, \bar{x}) = \delta(s_j, \bar{x})$. An HS for S is called an HS for M .

Intuitively, an HS \bar{x} is an input sequence such that for all states output sequence to \bar{x} uniquely identifies the final state. In other words, if the current state of an FSM is not known, then a homing sequence can be applied to the FSM and the output sequence produced by the FSM will tell us the final state reached. A homing sequence is also called a *homing word* in the literature. For FSM M_0 given in Figure 2.1b, the input sequence aa is an HS.

If M is minimal, then there certainly exists an HS for M (Kohavi, 1978). However,

there can be more than one HS for an FSM M . An input sequence \bar{x} is a *shortest HS for M* if there does not exist a shorter HS than \bar{x} for M . There can be multiple shortest HS for M as well.

In this work, we are interested in the upper bound for the length of the shortest homing sequences. For a minimal, complete, deterministic FSM M , let $|M|$ denote the length of the shortest HS for M and let $Q(n,p,o)$ be the set of all minimal, complete, deterministic FSMs with n states, p input symbols and o output symbols. We use the notation $H(n,p,o)$ to denote the upper bound of the length of the shortest HS of all FSMs in $Q(n,p,o)$. Formally, we define

$$H(n,p,o) = \max\{|M| : M \in Q(n,p,o)\}$$

Note that, by definition, $H(n,p,o)$ is a tight bound, i.e. there exists an FSM hitting to this bound.

As mentioned above, an HS \bar{x} is used to identify the final state of an FSM M . In other words, if we do not know the current state of M , we can apply the sequence \bar{x} to M and by looking at the output sequence by M as a response to \bar{x} , we can tell the final state reached. An automaton does not have the notion of an output symbol. Hence, nothing is observed as a reaction when an input sequence is applied to an automaton. However, in some cases, it is still possible to find an input sequence that can be used to identify the final state of an automaton. We now define input sequences that can be used for this purpose.

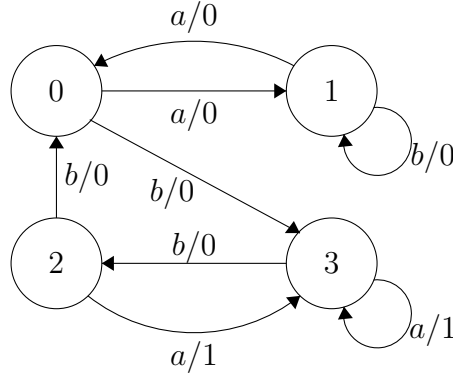
Definition 2. For an automaton $A = (S, X, \delta)$, a Synchronizing Sequence (SS) of A is an input sequence $\bar{R} \in X^*$ such that $|\delta(S, \bar{R})| = 1$.

A synchronizing sequence is also called a *reset sequence* in the literature. An automaton does not necessarily have an SS. It is known that the existence of an SS for an automaton can be checked in polynomial time (Eppstein, 1990).

The algorithms and our explanations will use the concept of uncertainty vector. Intuitively, an uncertainty vector of an FSM M is a collection of set of states of the FSM M . If one does not know anything about the current state of M , based on the application of an input sequence applied to M , we can infer some information, while still being uncertain about the current state. Basically, an uncertainty vector keeps such information.

Formally, an *initial state uncertainty vector for an input sequence $\bar{x} \in X^*$* is a partitioning $\pi(\bar{x}) = \{p_1, p_2, \dots, p_m\}$ of the states of M such that two states s, s' of M

Figure 2.2 An FSM M'



will belong to the same partition p_i in $\pi(\bar{x})$ iff $\bar{\lambda}(s, \bar{x}) = \bar{\lambda}(s', \bar{x})$. The states in the same block p_i give the same output to \bar{x} and hence cannot be distinguished by \bar{x} . On the other hand, a *current state uncertainty vector* (or simply *uncertainty vector*) for an input sequence \bar{x} is defined as $\sigma(\bar{x}) = \{\bar{\delta}(p_i, \bar{x}) | p_i \in \pi(\bar{x})\}$. Intuitively, the states in the same block of $\sigma(\bar{x})$ are the current states of the states that could not be distinguished from each other by \bar{x} .

The *successor tree* of an FSM $M = (S, X, Y, \delta, \lambda)$ is a tree where the nodes are labeled by the uncertainty vectors and the edges are labeled by the input symbols from X . The root of the successor tree is labeled by the uncertainty vector $\{S\}$. From each node of the successor tree there is an outgoing edge labeled by each of the input symbols $x \in X$. If the path from the root to a node is labeled by the sequence of input symbols \bar{x} , then the label of this node is $\sigma(\bar{x})$.

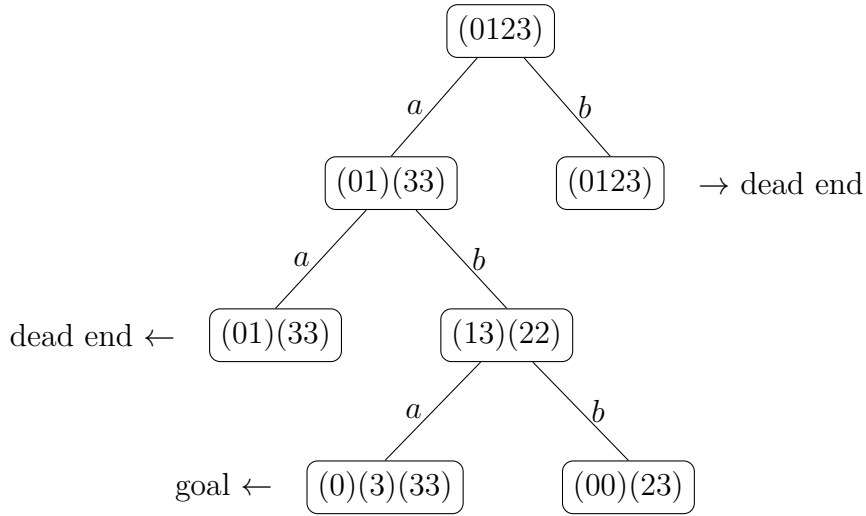
Homing Tree is a special case of successor tree, where certain nodes are pruned. There are two conditions to prune subtree at a certain node.

1. Let N be a node in the successor tree at level l_N , and N' be another node at level $l_{N'}$. Let uncertainty vector $P = \{p_1, p_2, \dots, p_m\}$ be the label of N , $P' = \{p'_1, p'_2, \dots, p'_{m'}\}$ be the label of N' . If for each $p_i \in P$, there exists a $p'_j \in P'$ such that $p_i \subseteq p'_j$ and $L_N \leq L_{N'}$, then the subtree at node N' can be pruned. (*dead end*)
2. Let N be a node with an uncertainty vector $P = \{p_1, p_2, \dots, p_m\}$ such that $|p_i| = 1$, for all $p_i \in P$. The subtree at node N can be pruned. (*goal*)

Note that, when a node is pruned by the condition (2) given above, that node gives us an HS. The label of the path from the root to that node is an HS for the FSM.

In Figure 2.3, the Homing tree for FSM M' in Figure 2.2 is given. As can be seen from the tree, aba is an HS for FSM M' .

Figure 2.3 The Homing tree of M' given in Figure 2.2



Recall that the problem of finding a shortest HS of an FSM is NP-hard. Therefore, unless $P = NP$, we have to use exponential time algorithms to compute a shortest HS. An easy brute-force algorithm to find a shortest HS is to construct the Homing Tree of an FSM in a breadth-first manner. The first goal node constructed by the algorithm will give us a shortest HS. Although this is an exponential time algorithm, this is the algorithm we use to compute the length of the shortest HS of an FSM.

In this work, we essentially consider all FSMs $M \in Q(n, 2, 2)$ to compute $H(n, 2, 2)$. However, this does not mean that we really take each and every FSM $M \in Q(n, 2, 2)$, and compute the length of the shortest HS of FSM M . This would be practically infeasible, even for small state sizes we used in our work. Furthermore, we are not aware of any direct method of enumerating the FSMs in $Q(n, 2, 2)$, which consists of only minimal FSMs. Hence, the only way to generate FSMs in $Q(n, 2, 2)$ is to generate FSMs with n states, 2 input symbols and 2 output symbols, and check if they are minimal. Of course, this makes the required computation even more expensive.

In order to be able to complete the required computation, we use several theoretical and practical improvements. These improvements are explained in Section 3 in detail. The theoretical methods employed to speed-up the search are based on skipping an FSM M (or sometimes a set of FSMs all together), whenever we understand that the shortest HS of M cannot hit to the upper bound $H(n, 2, 2)$. Since we do not know $H(n, 2, 2)$ in the beginning of the search, we start by a conservative *estimate/conjecture* H_n for which we know $H_n \leq H(n, 2, 2)$. During the search if we ever come across an FSM $M \in Q(n, 2, 2)$ such that $|M| > H_n$, we simply update the current conjecture H_n as $|M|$. During our search, if for an FSM M (or for a set

of FSMs) we understand that $|M| < H_n$, we skip M . Note that, it is sometimes possible to understand that $|M| < H_n$, without actually computing the shortest HS for M . For example, if we can find an upper bound for $|M|$ which is smaller than H_n , we surely have $|M| < H_n$.

Above, we defined an SS over an automaton. We can also define an SS for an FSM as well in the following way. An input sequence is an SS for an FSM M , if it is an SS for the automaton $M|_A$ of M . Note that, if an input sequence \bar{x} is an SS for an FSM M , it is also an HS for M . Therefore, if an SS \bar{x} has length strictly shorter than H_n , then M can be skipped, i.e. M does not have a chance of hitting $H(n, 2, 2)$, and it does not even have chance of improving the current conjecture. In fact, not only M , but any M' such that $M'|_A = M|_A$ can be skipped. The way we generate FSMs allows an easy way of skipping such FSMs all together. As we will explain in Section 3, we generate FSMs by considering an automaton augmenting it with several output functions. Hence, when we see that an automaton A has an SS shorter than the current conjecture H_n , we skip all possible FSMs that could have been generated from A by augmenting it with different output functions.

Note that knowing an upper bound for the length of an SS for an FSM M , also gives us an upper bound for the length of an HS for M as well. We will apply several ideas along this line. One of them is based on the smallest subset of states that one can reach by simply applying a sequence of inputs consisting of the same input symbol. The details will be explained later, but here we only introduce the terminology used for this purpose. Let $M = (S, X, Y, \delta, \lambda)$ be an FSM with only one input symbol, i.e. $X = \{a\}$. Consider the sequence of set of states $\delta(S, a^1), \delta(S, a^2), \delta(S, a^3), \dots$. It is easy to see that $\delta(S, a^i) \subseteq \delta(S, a^{i+1})$. However, there exists an integer ℓ such that for any $k \geq \ell$, $\delta(S, a^k) = \delta(S, a^{k+1})$. We call the smallest such integer ℓ the *reduction-threshold* of M and $\delta(S, a^\ell)$ is called the *reduction-set* of M .

3. EXHAUSTIVE NON-ISOMORPHIC FSM GENERATION

Hibbard (1961) states the upper bound for the length of shortest homing sequences as $n(n-1)/2$, and shows that this bound is tight by providing a class of FSMs (see Figure 1.1) hitting to this upper bound. As can be seen in Figure 1.1, these FSMs have two output symbols but the number of input symbols grows linearly with the number of states. Hence, using our notation, we can state Hibbard's upper bound as $H(n, n-1, 2) = n(n-1)/2$.

It is easy to see that $H(n, p, 2) = H(n, n-1, 2) = n(n-1)/2$ for any $p \geq n$ (just consider adding new input symbols to the FSM in Figure 1.1 as self looping transitions with output 0).

On the hand it is not immediately clear if we have $H(n, p, 2) = H(n, n-1, 2) = n(n-1)/2$ when $p < n-1$. Our claim is that $H(n, p, 2) < n(n-1)/2$ when $p < n-1$. To support this claim, we generate all the FSMs with n states, two input symbols and two output symbols.

Note that, the number of FSMs with n states, p input symbols and o output symbols is $(n \times o)^{(n \times p)}$. Even for the small FSM sizes that we consider in our work, the number of FSMs that needs to be considered reaches to 20^{20} for the largest FSM size of 10 states, 2 input symbols, 2 output symbols in our study. This is too big of a number of FSMs to be enumerated in practice. Therefore, we employ several techniques to reduce the number of FSMs considered, without affecting the outcome of the analysis. We explain the details of all these techniques in the section.

Even after eliminating some class of FSMs, there will be some FSMs for which we explicitly have to compute a shortest HS. We compute such shortest HS by using the exponential brute-force algorithm explained in Section 2. This is acceptable for our purposes, since the size of the FSM considered are quite small, and we can afford an exponential time algorithm for such small FSMs.

To support our claim we need to generate all the binary FSMs with n states and 2 output symbols. We can produce a binary FSM by just superimposing 2 unary

FSMs with n states and 2 output symbols. For creating all non-isomorphic binary FSMs, we also need to use the methods described in Kisielewicz & Szykuła (2013) beside superimposition. As we can form a binary FSM by just superimposing 2 unary FSMs, we need a set of unary FSMs. We can obtain this set of unary FSMs by extending unary automata with additional set of output symbols and output function. Therefore, we need the set of unary automata which we had already from one of the previous works. The generation phase of unary automata set is out of our scope and we are getting that set as an input at beginning of our program. During this entire generation process, we have some opportunities to not consider some automata or FSMs and since we are generating binary FSMs by using unary automata as building blocks, each elimination in one stage means, there won't be any elements generated from the corresponding eliminated element in next stages. As an example if a unary automaton is eliminated, there won't be any unary or binary FSM generated using this corresponding automaton. In the following subsections, we describe how we use the opportunities for elimination more detailed by introducing theorems.

3.1 Non-isomorphic Unary Automaton Selection

Here in this section, the techniques that we used to eliminate some set of unary automata, will be explained with showing the according theorems. To eliminate, we need a set of unary automaton which we are getting it as an input. This set is generated with brute force and the isomorphic ones eliminated before we use it. We will not go into details about the generation of corresponding non-isomorphic unary automaton set since it is out of our scope but the details of elimination process will be seen in the upcoming parts of this paper.

Theorem 1. *Let $M = (S, X, Y, \delta, \lambda)$ be an FSM and $M|_A = (S, X, \delta)$. If a sequence \bar{x} is a synchronizing sequence for $M|_A$ then $\bar{x} \in X^*$ is a homing sequence for FSM M .*

Proof. Since $\delta(s, \bar{x}) = s'$ for all $s \in S$ and for some $s' \in S$, for all states $s_i, s_j \in S$, $\lambda(s_i, \bar{x}) = \lambda(s_j, \bar{x}) \implies \delta(s_i, \bar{x}) = \delta(s_j, \bar{x}) = s'$ \square

Using Theorem 1, we eliminated every automaton with a shorter synchronizing

sequence than the current conjecture of $H(n, p, o)$ by finding their shortest synchronizing sequence which is NP-hard. As the goal is finding $H(n, p, o)$, the homing sequences that can be found for the possible FSMs generated by adding output to an automaton which has shorter synchronizing sequence than the current conjecture of $H(n, p, o)$ can only decrease the length of shortest homing sequence for corresponding FSM if length of corresponding homing sequence is less than the synchronizing sequence of that automaton. This technique can also be applied when generating binary automata.

But to eliminate some of the unary automata, those automata don't need to have a synchronizing sequence. We can still find a limit for the possible homing sequence length after reaching the reduction-set of corresponding automaton according to the following theorems.

Theorem 2. (Hibbard, 1961) *For an FSM with n states and a subset S' of k states, there exists an input sequence \bar{x} with length at most $n - k + 1$ such that \bar{x} separates S' .*

Corollary 1. *For an FSM with n states, to separate two states, an input sequence with length at most $n - 1$ is enough.*

Proof. Consider Theorem 2 when $k = 2$. □

Theorem 3. *For an FSM M with n states, and a subset S' of k states of M , there always exists an HS for S' with length at most $((k - 1) \times (n + 1)) - ((k \times (k + 1))/2) + 1$.*

Proof. After every sequence of inputs which separates a single state from the others makes the number of elements in partition decrease. So for the first state can be separated after applying a sequence with length at most $n - k + 1$. The second state can be homed after applying a sequence with length at most $n - (k - 1) + 1 = n - k + 2$ and so on. Finally, the last two states in a partition will be separated with an input sequence with length at most $n - 2 + 1$. Hence the sum of the length of all these sequences is

$$\sum_{i=k}^2 (n - i + 1) = ((k - 1) \times (n + 1)) - ((k \times (k + 1))/2) + 1$$

□

Note that, Theorem 3 gives the Hibbard's bound when $k = n$. However, when

$k < n$, we can obtain a better upper bound for the length of the shortest HS of the given FSM. In other words, suppose that we are given a unary automaton with input alphabet $X = \{x\}$, where ℓ is the reduction-threshold. Hence, we reach to the reduction-set of the automaton by using the sequence x^ℓ . Assume that the cardinality of the reduction-set of the automaton is k . Then it is easy to see that, any minimal FSM which is generated using this automaton will have a homing sequence of length at most $\ell + (((k-1) \times (n+1)) - ((k \times (k+1))/2) + 1)$. Therefore, if this number is less than the current conjecture for $H(n, p, o)$, we can eliminate the given automaton, without considering any FSM that could be generated using this automaton.

3.2 Non-isomorphic Binary Automaton Generation

A binary automaton is generated by superimposing two unary automata as follows. Let $A = (S, X_a, \delta_a)$ and $B = (S, X_b, \delta_b)$ two unary automata. A binary automaton $C = (S, X_a \cup X_b, \delta)$ can be constructed as

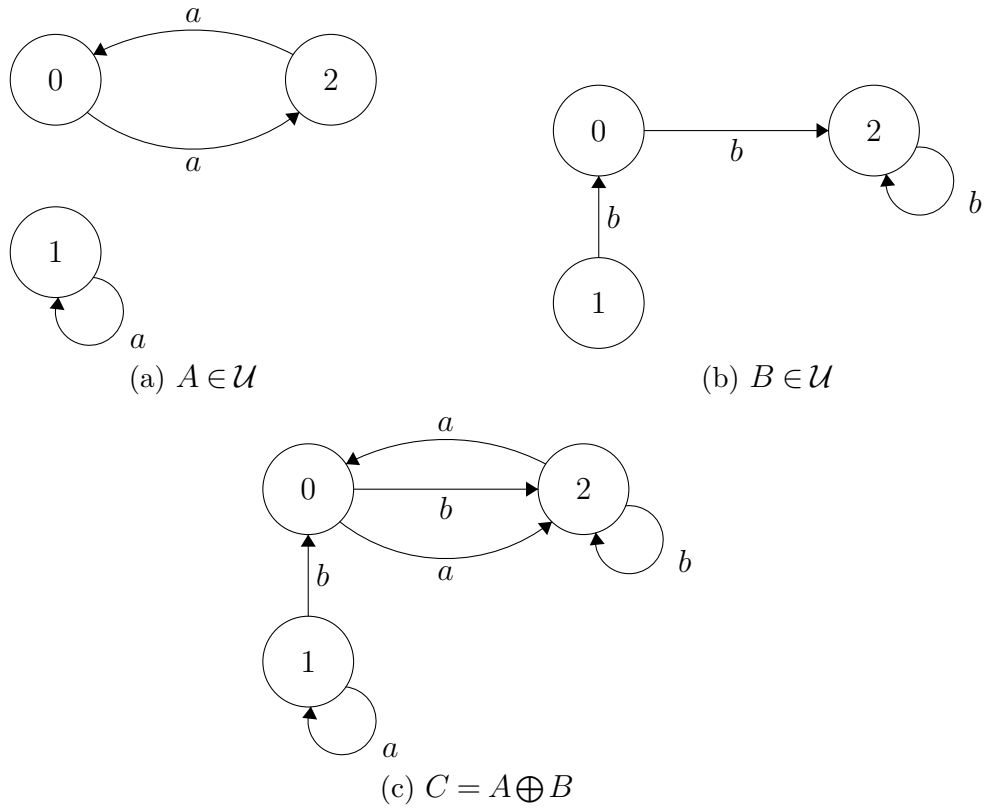
- $\delta(s, x) = \delta_a(s, x)$ for every $s \in S, x \in X_a$
- $\delta(s, x) = \delta_b(s, x)$ for every $s \in S, x \in X_b$

We will show this superimposition as $C = A \oplus B$. In Figure 3.1, an example superimposition is given.

In order to create all possible binary automata, we use the unary automata collection \mathcal{U} we have. As explained in Section 3.1, some of the unary automata are eliminated, because they do not stand a chance to be used as the underlying automaton of an FSM which hits to the upper bound we are after. Let \mathcal{U} be the all non-isomorphic unary automata with n states and let \mathcal{U}' be the subset of \mathcal{U} consisting of those automaton that could not be eliminated by using the techniques given in Section 3.1.

In order to form a binary automaton, we will consider pairs A, B of unary automata in \mathcal{U}' and superimpose them. However, it is not sufficient to generate every possible binary automaton by simply considering the superimposition of every unary automaton pair $A, B \in \mathcal{U}'$. In other words, if we just consider automata $C = A \oplus B$ automata for all $A, B \in \mathcal{U}$, there will be some binary automaton not formed/generated using this approach. Instead, one has to consider the permutations of the states of

Figure 3.1 Two unary automata $A, B \in \mathcal{U}$ and a binary automaton $C = A \oplus B$



the unary automata as well. However, permuting the states of one of the unary automaton is sufficient for this purpose. Therefore, we consider the states of the unary automaton A as fixed, and we rename the names of the states of the unary automaton B . This corresponds to creating all isomorphic unary automata for B , and pairing it with the unary automaton A for superimposition as explained below.

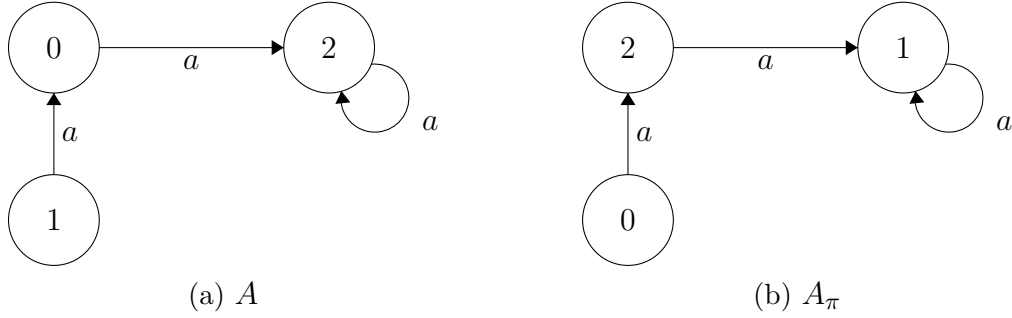
Given a unary automaton $A = (S, X, \delta)$ with n state, and a bijection π from S to S (i.e. a permutation on S), we can create an isomorphic unary automaton $A_\pi = (S, X, \delta_\pi)$ by taking $\delta_\pi(\pi(s), x) = \pi(\delta(s, x))$, for all $s \in S, x \in X$. Note that one can get $n!$ different isomorphic automata A_π in this way.

In Figure 3.2a and Figure 3.2b, you can find an example unary automaton A and an isomorphic automaton A_π which is generated by renaming the states of A .

The following theorem states that this approach is sufficient to generate at least one binary automaton from every isomorphism class.

Theorem 4. *For every binary automaton C , there is an isomorphic binary automaton which can be created as $A \oplus B_\pi$ where $A = (S, X_a, \delta_a)$ and $B = (S, X_b, \delta_b)$ are two unary automata and B_π is the permutation of B with bijection $\pi : S \rightarrow S'$ (Kisielewicz & Szykula, 2013).*

Figure 3.2 A unary automaton A and an isomorphic automaton A_π where $\pi(0) = 2, \pi(1) = 0, \pi(2) = 1$



Using Theorem 4, we generate binary automaton by getting automata pairs from our non-isomorphic unary automata set if they are not eliminated with theorems in previous subsection.

As in non-isomorphic unary automaton generation phase, we eliminate binary automata from our constructed binary automaton set if their synchronizing sequence length is less than the current conjecture of $H(n, p, o)$ by using Theorem 1.

We are also eliminating some of the isomorphic binary automata which is generated according to Theorem 4, using symmetry properties of automata. You can find the details in the paper of Kisielewicz & Szykuła (2013).

3.3 Non-isomorphic FSM Generation

In non-isomorphic FSM generation part, similarly to the automaton generation part, first we are generating unary FSMs, eliminate some of them according to some techniques described below and create binary FSMs with remaining unary FSMs.

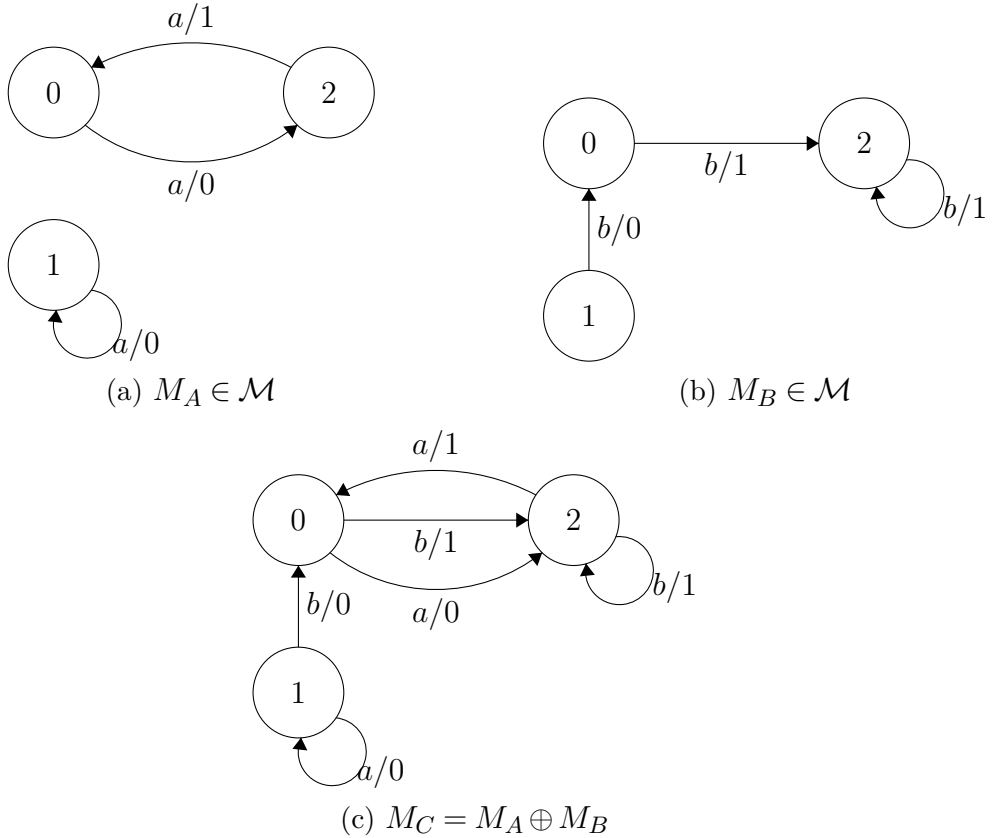
We are generating binary FSMs using similar methods with binary automata generation. A binary FSM is generated by superimposing two unary FSMs from our unary FSM set \mathcal{M} as follows. Let $M_A = (S, X_a, Y, \delta_a, \lambda_a)$ and $M_B = (S, X_b, Y, \delta_b, \lambda_b)$ two unary FSMs. A binary FSM $M_C = (S, X_a \cup X_b, Y, \delta, \lambda)$ can be constructed as

- $\delta(s, x) = \delta_a(s, x)$ for every $s \in S, x \in X_a$
- $\delta(s, x) = \delta_b(s, x)$ for every $s \in S, x \in X_b$

- $\lambda(s, x) = \lambda_a(s, x)$ for every $s \in S, x \in X_a$
- $\lambda(s, x) = \lambda_b(s, x)$ for every $s \in S, x \in X_b$

We will show this superimposition as $M_C = M_A \oplus M_B$. In Figure 3.3, an example superimposition is given.

Figure 3.3 Two unary FSMs $M_A, M_B \in \mathcal{M}$ and a binary FSM $M_C = M_A \oplus M_B$

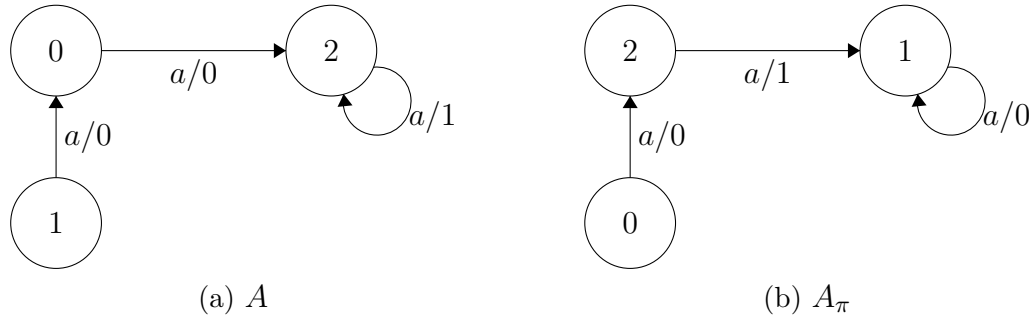


Given a unary FSM $M = (S, X, Y, \delta, \lambda)$ with n state, and a bijection π from S to S (i.e. a permutation on S), we can create an isomorphic unary FSM $M_\pi = (S, X, Y, \delta_\pi, \lambda_\pi)$ by taking $\delta_\pi(\pi(s), x) = \pi(\delta(s, x))$ and $\lambda_\pi(\pi(s), x) = \lambda(s, x)$ for all $s \in S, x \in X$. Note that one can get $n!$ different isomorphic FSM M_π in this way.

In Figure 3.4a and Figure 3.4b, you can find an example unary FSM M and an isomorphic FSM M_π which is generated by renaming the states of M and the outputs accordingly.

Before starting to elimination of process, we know that if the possible shortest homing sequence of a unary FSM is less than our current conjecture, we are eliminating this FSM from our set to generate binary FSMs. Also from the remaining unary FSMs with no homing sequences, a subset of them can be eliminated since some output functions are unnecessary according to the theorems described below.

Figure 3.4 A unary FSM M and an isomorphic FSM M_π where $\pi(0) = 2, \pi(1) = 0, \pi(2) = 1$



In the binary FSM generation part, outputs will be added to transitions. Our first claim is that while adding these outputs we don't need to make experiments with $o = 1$

Theorem 5. *Let $M = (S, X, Y, \delta, \lambda)$ be an FSM. There doesn't exist any minimal FSM M when $o = 1$.*

Proof. Since $o = 1$, $\lambda(s_i, x) = \lambda(s_j, x)$ for any input symbol $x \in X$ and for any two different states $s_i, s_j \in S$. Therefore, for any input sequence $\bar{x} \in X^*$, $\lambda(s_i, \bar{x}) = \lambda(s_j, \bar{x})$. Hence there doesn't exist any input sequence $\bar{x} \in X^*$, $\lambda(s_i, \bar{x}) \neq \lambda(s_j, \bar{x})$. □

Then half of the FSMs can be eliminated if there was no other elimination method by using the Theorem 6 below. You can find the experimental numbers of eliminated FSMs using Theorem 6 in Section 4 in Table 4.2.

Theorem 6. *Let $M_1 = (S, X, Y_1, \delta, \lambda_1)$ and $M_2 = (S, X, Y_2, \delta, \lambda_2)$ be two output-isomorphic FSMs. Then a sequence $\bar{x} \in X^*$ is an HS for FSM M_1 if and only if \bar{x} is an HS for M_2 .*

Proof. Let $\bar{x} \in X^*$ be an HS for M_1 . This means $\forall s_1, s_2 \in S$, if $\lambda_1(s_1, \bar{x}) = \lambda_1(s_2, \bar{x})$ then $\delta(s_1, \bar{x}) = \delta(s_2, \bar{x})$. Since there is a bijection $g : Y_1 \rightarrow Y_2$ such that $\forall x \in X$ and $\forall s \in S, g(\lambda_1(s, x)) = \lambda_2(s, x)$, $\lambda_1(s_1, \bar{x}) = \lambda_1(s_2, \bar{x})$ if and only if $\lambda_2(s_1, \bar{x}) = \lambda_2(s_2, \bar{x})$. As the transition function δ is the same for both M_1 and M_2 , if $\lambda_2(s_1, \bar{x}) = \lambda_2(s_2, \bar{x})$, then $\delta(s_1, \bar{x}) = \delta(s_2, \bar{x})$. Hence, \bar{x} is an HS for M_2 . □

Theorem 7. *Let $M_1 = (S, X, Y_1, \delta, \lambda_1)$ and $M_2 = (S, X, Y_2, \delta, \lambda_2)$ are two FSMs where $Y = \{0, 1\}$ and $\lambda_1(s, x) = 1 - \lambda_2(s, x)$ for every $s \in S, x \in X$. Then a sequence $\bar{x} \in X^*$ is a homing sequence for FSM M_1 if and only if \bar{x} is a homing sequence for M_2 .*

Proof. Having $\lambda_1(s, x) = 1 - \lambda_2(s, x)$ means that, we have $g(\lambda_1(s, x)) = \lambda_2(s, x)$ for the bijection $g : \{0, 1\} \rightarrow \{0, 1\}$, where $g(0) = 1$ and $g(1) = 0$. Therefore, M_1 and M_2 are output-isomorphic FSMs. Then the result follows by using Theorem 6. \square

Using Theorem 7, it can be concluded that, for $Y = \{0, 1\}$, there is no need to create FSMs $|(s, x)|\lambda(s, x) = 0| > |(s, x)|\lambda(s, x) = 1|$ for all $x \in X$ and $s \in S$ or simply considering FSMs with more 0 than 1 in total as a result of $\lambda(s, x)$ for all $x \in X$ and $s \in S$ is unnecessary.

Some large amount of FSMs can be excluded according to the following Theorem 8.

Theorem 8. *Let $X' \subseteq X$ be a subset inputs and $M_1 = (S, X, Y, \delta, \lambda_1)$, $M_2 = (S, X, Y, \delta, \lambda_2)$ be two FSMs such that:*

- $\lambda_2(s_i, x) = \lambda_2(s_j, x)$ for all $s_i, s_j \in S$, $x \in X'$, i.e. all output symbols are the same for the transitions in M_2 with input symbols in X' . Note that, this would be the case if λ_2 is all-zeroes for X' .
- $\lambda_1(s, x) = \lambda_2(s, x)$ for all $x \in X \setminus X'$ and for all $s \in S$. In other words, M_1 and M_2 have the same output function for the transitions with the input symbols in $X \setminus X'$.

If a sequence $\bar{x} \in X^$ is a homing sequence for M_2 , then \bar{x} is a homing sequence for FSM M_1 .*

Proof. If \bar{x} is an HS for M_1 , following properties should be satisfied.

- $\lambda_1(s_i, \bar{x}) \neq \lambda_1(s_j, \bar{x})$ for any $s_i, s_j \in S$ or
- if $\lambda_1(s_i, \bar{x}) = \lambda_1(s_j, \bar{x})$ for any $s_i, s_j \in S$ then $\delta(s_i, \bar{x}) = \delta(s_j, \bar{x})$

Let \bar{x} is an HS for M_2 . Then \bar{x} satisfies the first property above since for any input sequence $\bar{x}' \in X^*$ and for any two states $s_i, s_j \in S$, if $\lambda_2(s_i, \bar{x}') \neq \lambda_2(s_j, \bar{x}')$ then $\lambda_1(s_i, \bar{x}') \neq \lambda_1(s_j, \bar{x}')$. That's because, for some input symbols and for all states M_1 and M_2 have the same output functions and for all remaining transitions M_2 produce the same output symbol (properties of M_1 and M_2 in Theorem 8). Also, \bar{x} satisfies the second property as both M_1 and M_2 uses the same transition function δ , which implies if $\lambda_1(s_i, \bar{x}) = \lambda_1(s_j, \bar{x})$ then $\delta(s_i, \bar{x}) = \delta(s_j, \bar{x})$. Hence \bar{x} is an HS for M_1 . \square

We use Theorem 8 in our process as follows. If we have two FSMs $M_1 =$

$(S, X, Y, \delta, \lambda_1)$, $M_2 = (S, X, Y, \delta, \lambda_2)$ satisfying the premises of Theorem 8, we only need to consider M_2 if M_2 is minimal since M_1 does not have the chance of having a longer shortest HS than M_2 . If M_2 is not minimal, we have to generate all binary FSMs which can be created by changing all-zero output of M_2 with all the other allowed output functions. By this method, if all of the generated FSMs are minimal, we just need to consider $2 \times ((n \times o)^n - n^n) \times n^n$ FSMs rather than $(n \times o)^{(n \times p)}$ FSMs for our experiments with using no other elimination method. You can find the experimental numbers of eliminated FSMs using Theorem 8 in Section 4 in Table 4.2..

You can find the types of used or eliminated output functions while superimposing two unary FSMs to generate binary FSM by using Theorem 8 in Table 3.1.

Table 3.1 Usage of output functions in experiments according to their types to generate binary FSMs

Types	all-zeroes	single-one	multi-one
all-zeroes	Not Used Theorem 5 implied	Used Round 1	Used Round 2
single-one	Used Round 1	Only the ones that are not eliminated by Theorem 8 - Round 1	Only the ones that are not eliminated by Theorem 8 - Round 1 and Round 2
multi-one	Used Round 2	Only the ones that are not eliminated by Theorem 8 - Round 1 and Round 2	Only the ones that are not eliminated by Theorem 8 - Round 2

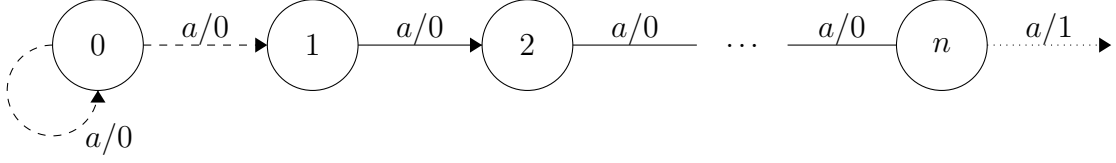
As a note, we didn't do any experiment for $p = 1$ because of the Theorem 9.

Theorem 9. $H(n, 1, 2) = n - 1$

Proof. According to Corollary 1, for 2 states in an FSM, an input sequence with length at most $n - 1$ is enough to separate them and it is true for every $s, s' \in S$. The input sequence with length $n - 1$ can only be x^{n-1} since $X = \{x\}$ in a unary FSM. Therefore one of the prefixes of x^{n-1} must be enough to separate each and every state in a unary FSM, so $H(n, 1, 2) = n - 1$. □

The bound $H(n, 1, 2) = n - 1$ is actually tight. You can find the structure of FSMs that are hitting to this upper bound when $X = \{a\}$ and $Y = \{0, 1\}$ in Figure 3.5. One of the dashed transitions is enough for the setting. The dotted transition can be connected to any state of the FSM.

Figure 3.5 Structure of FSM hits to upper bound when $p = 1$



When we first start to experiments for a given n , we are setting current conjecture of $H(n, 2, 2)$ as $H(n-1, 2, 2)$ using Theorem 10.

Theorem 10. $H(n, p, 2) \leq H(n+1, p, 2)$

Proof. For $p = 1$, it can be proven by using Theorem 9. For $p > 1$, let $M = (S, X, Y, \delta, \lambda)$ be an FSM when shortest homing sequence of M is \bar{x} , $|\bar{x}| = H(n, p, o)$. Construct $M' = (S \cup \{s_{n+1}\}, X, Y, \delta', \lambda')$ where for every $s \in S, x \in X$:

- $\delta'(s, x) = \delta(s, x)$
- $\lambda'(s, x) = \lambda(s, x)$

Let $\bar{x} = x\bar{x}'$ be an HS for M such that $|\bar{x}| = H(n, p, 2)$. For some $s \in S$, we set $\delta'(s_{n+1}, x) = \delta'(s, x)$. This makes sure that \bar{x} will also be an HS for M' .

However, we also need to have M' as a minimal FSM. To this end, let $x' \in X \setminus \{x\}$ an input symbol other than x and set the transition and the output of s_{n+1} for x' such that for all $s \in S$, $(\delta'(s_{n+1}, x'), \lambda'(s_{n+1}, x')) \neq (\delta'(s, x'), \lambda'(s, x'))$. In this way, s_{n+1} will either produce a different output or it will go into a different state with any $s \in S$ under input x' . Hence, we will be able to distinguish s_{n+1} from any other state, which makes sure that M' is minimal, since M is minimal.

Note that setting the transition and the output of s_{n+1} for x' such that for all $s \in S$, $(\delta'(s_{n+1}, x'), \lambda'(s_{n+1}, x')) \neq (\delta'(s, x'), \lambda'(s, x'))$ is possible for any x' , because there are $n \times o$ options but there is only n transitions in M with input x' . Therefore non-existing pair $(\delta'(s_{n+1}, x'), \lambda'(s_{n+1}, x'))$ can always be found.

Since M' constructed as above is minimal and \bar{x} is an HS for M' , we conclude that $H(n, p, 2) = |\bar{x}| \leq H(n+1, p, 2)$.

□

Theorem 11. Consider an FSM with n states, and let \bar{x} be an input sequence and $\sigma(\bar{x}) = \{p_1, p_2, \dots, p_m\}$ be the uncertainty vector for \bar{x} . Let k_1, k_2, \dots, k_m be the cardinalities of p_1, p_2, \dots, p_m , respectively. The length of a shortest homing sequence

for M is at most

$$|\bar{x}| + \sum_{\ell=1}^m \sum_{i=k_\ell}^2 (n-i+1) = |\bar{x}| + \sum_{\ell=1}^m (((k_\ell - 1) \times (n+1)) - ((k_\ell \times (k_\ell + 1))/2) + 1)$$

Proof. Maximum length of a homing sequence for a given partition is given in Theorem 3. Summation of these lengths for each partition will be resulted as the formula above. If we try to prove by induction, for $m = 1$, it can be proved using Theorem 3. If we assume that formula is correct for $m - 1$, then we have:

$$|\bar{x}| + \underbrace{\sum_{\ell=1}^{m-1} \sum_{i=k_\ell}^2 (n-i+1)}_{\text{by induction hypothesis}} + \underbrace{\sum_{i=k_m}^2 (n-i+1)}_{\substack{\text{upper bound for the length} \\ \text{of an HS for a single partition} \\ p_m \text{ by Theorem 3}}} = |\bar{x}| + \sum_{\ell=1}^m \sum_{i=k_\ell}^2 (n-i+1)$$

Since for any sequence $\bar{x} \in X^*$, for every $p_i \in \sigma(\bar{x})$, $|p_i| \geq |\delta(p_i, \bar{x})|$, summation above is correct for worst case. □

For an uncertainty vector, adding the length given by Theorem 11 to the length of the input sequence in the path to that uncertainty vector would provide an upper bound for the shortest homing sequence of the FSM.

When constructing the homing tree of a unary FSM M , we obtain a unary tree as expected. While trying to find a homing sequence for M by constructing the homing tree, let us assume that we come across to a uncertainty vector which is seen before, which means there is no need to check the rest of the homing tree (pruned) since there is no homing sequence of this unary FSM which is also known as dead end. However, for unary FSM M , if we consider the uncertainty vector obtained by the input sequence taking M to the reduction-set, the reduction-threshold added to the length given in Theorem 11 can be used as an upper bound for the length of the HS of M . In case this upper bound is lower than the current conjecture for $H(n, p, o)$, one can skip further analysis of M .

4. EXPERIMENTS

In this section, We will explain the experimental study we have conducted to find an upper for the shortest homing sequence for an FSM with given states and inputs when the number of inputs is the less than number of states of corresponding FSM.s

The experiments were performed on a machine with Intel(R) Xeon(R) CPU E7-4870 CPU and 50GB of memory, using Ubuntu 16.04.2. The code was written in C/C++ and compiled using gcc with -o3 option enabled and the times elapsed are measured in terms of microseconds.

In our program, first we get non-isomorphic unary automata set as input. In unary automata generation phase we eliminate some of those automata using the help of Theorem 3. After unary automata generation phase, we generate unary FSMs to use Theorem 3 more efficiently as Theorem 11. Then we produce binary automata to find their synchronizing sequences and compare their length with current conjecture of $H(n, p, o)$. The remaining binary automata generates FSMs with non-eliminated additional outputs by theorems in FSM generation phase and we are done with the generation. Finally, we find shortest homing sequence of each FSM by using a homing tree if they are minimal and updating the current $H(n, p, o)$ accordingly. You can find the general algorithm as pseudocode at the end of this section.

We generated all the non-isomorphic FSMs with number of states $n \in \{3, 4, 5, 6, 7, 8, 9, 10\}$, the number of input symbols $p = 2$ and the number of output symbols $o = 2$ if they were not eliminated with the techniques mentioned in Section 3. Some isomorphic FSMs to previous one with same preferences are also generated as our algorithm doesn't eliminate some of isomorphic FSMs but they didn't change the $H(n, 2, 2)$ values as expected.

Table 4.1 gives the number of all possible FSMs and number of non-isomorphic FSMs for each $n \in \{3, 4, 5, 6, 7, 8, 9, 10\}$ and $p = \{1, 2\}$.

When we add outputs to FSMs, by the help of Theorem 6, we don't assign more

Table 4.1 Number of all and non-isomorphic FSMs according to their number of states and inputs (Harary & Palmer, 2014)

# of States (n)	$p = 1$		$p = 2$	
	All (n^n)	Non-isomorphic	All (n^{2n})	Non-isomorphic
3	27	7	729	74
4	256	19	65,536	1,474
5	3125	47	$9,76 \times 10^6$	41,876
6	46,656	130	2.17×10^9	1.54×10^6
7	823,543	343	6.78×10^{11}	6.83×10^7
8	1.67×10^7	951	2.81×10^{14}	3.54×10^9
9	3.87×10^8	2,615	1.50×10^{17}	2.09×10^{11}
10	1.00×10^{10}	7,318	1.00×10^{20}	1.39×10^{13}

1's than 0's in total when the set of outputs $Y = \{0, 1\}$. And for all the n, p values, the FSMs hits to the bound have $n - 1$ 0's and single 1 as the result of the output function which makes sense as intuitively it makes harder to identify the final states from the outputs produced with the high percentage of same output symbol in transitions.

As a summary, we are doing all our eliminations according to rules below.

- **Rule 1:** For all unary automata, add the length of an input sequence \bar{x} which reaches to reduction-threshold and the result of the formula in Theorem 3 for the partition reached after applying \bar{x} and check if the summation is less than current conjecture of $H(n, 2, 2)$. If it so eliminate that automaton.
- **Rule 2:** For all unary FSMs with allowed outputs using Theorem 7, add the length of an input sequence \bar{x} which reaches to reduction-threshold and the result of the formula in Theorem 11 for the partition reached after applying \bar{x} . Then eliminate that automaton if the summation is less than current conjecture of $H(n, 2, 2)$.
- **Rule 3:** For all binary automata that are generated by superimposition of 2 not eliminated unary automata (one of them is permuted), check if there is already an another automaton which is isomorphic to one that recently generated.
- **Rule 4:** For all binary automata that are not eliminated by Rule3, find their shortest synchronizing sequence and eliminate that automaton if the length of shortest synchronizing sequence of corresponding automaton is less than current conjecture of $H(n, 2, 2)$.
- **Rule 5:** Before generating binary FSMs that are generated by superimposition

of 2 not eliminated unary FSM (one of them is permuted), check each unary FSM whether their bound obtained by Theorem 11 is still greater than or equal to current conjecture of $H(n, 2, 2)$, else eliminate them.

- **Rule 6:** For each binary FSM $M_1 = (S, X, Y, \delta, \lambda_1)$, don't generate the binary FSM $M_2 = (S, X, Y, \delta, \lambda_2)$ where $\lambda_1(s, x) = 1 - \lambda_2(s, x)$ by implying Theorem 7.
- **Rule 7:** With remaining unary FSMs, generate binary FSMs according to Table 3.1.

You can find the number of eliminated automata and FSMs for each $n \in \{3, 4, 5, 6, 7, 8, 9, 10\}$ in Table 4.2 according to rules above.

Table 4.2 Experimental results for number of eliminated automata and FSMs according to their number of states

# of States(n)	# of unary automata			# of unary FSMs		# of binary automata		# of binary FSMs	
	Rule 1	Rule 2	Rule 5	Rule 4	Rule 3	Rule 6	Rule 7		
3	0	0	149	96	144	72	116		
4	3	43	1343	2155	2376	1741	1869		
5	13	266	26587	47677	40206	43111	81015		
6	50	1530	1090836	1135638	1206120	875240	2636457		
7	140	9137	21251273	51505987	44478974	28622016	108953271		
8	515	45712	542721008	1354592452	1409695814	565719206	3006556852		
9	1645	205999	16071514863	52717190332	32430768070	8676554491	78029236785		
10	4877	1144946	372846887604	2595548013299	1454954966204	607306831891	7816703036086		

For each n value, when we start to experiments, we knew the bound for FSMs with $n - 1$ states ($H(n - 1, 2, 2)$) so we set that bound as our current conjecture of $H(n, 2, 2)$ using Theorem 10 to eliminate the automata or FSMs accordingly. As we update current conjecture of $H(n, p, o)$ faster, we eliminate more automata and FSMs. Therefore, we selected a group of FSMs which have $n - 1$ 0's and single 1 in their output function to find their shortest homing sequences first since all the FSMs that hit to bound are from this set yet. Still we couldn't prove that the FSMs that hit to bound should be always from the set with single 1's in their output function.

Table 4.3 gives the number of FSMs that are generated, elapsed time and the $H(n, 2, 2)$ for each $n \in \{3, 4, 5, 6, 7, 8, 9, 10\}$.

Table 4.3 Experimental results for shortest homing sequences according to number of states

# of States(n)	# of generated binary FSMs	Elapsed Time	$H(n, 2, 2)$	# of FSMs Hits to $H(n, 2, 2)$	# of Non-Isomorphic FSMs Hits to $H(n, 2, 2)$
3	72	0m 0.046s	3	19	12
4	364	0m 0.06s	6	21	11
5	8877	0m 0.094s	9	140	74
6	175336	0m 0.566s	13	132	96
7	2396409	0m 12.374s	18	58	40
8	67528615	4m 59.910s	24	68	40
9	8676554491	142m 22.719s	31	18	8
10	607306831891	12210m 30.236s	38	62	18

In the following figures, you will see some example FSM structures which hits to $H(n, 2, 2)$.

Figure 4.1 Two unary FSMs $M_A, M_B \in \mathcal{M}$ and a binary FSM $M_C = M_A \oplus M_B$ which hits to $H(4, 2, 2)$

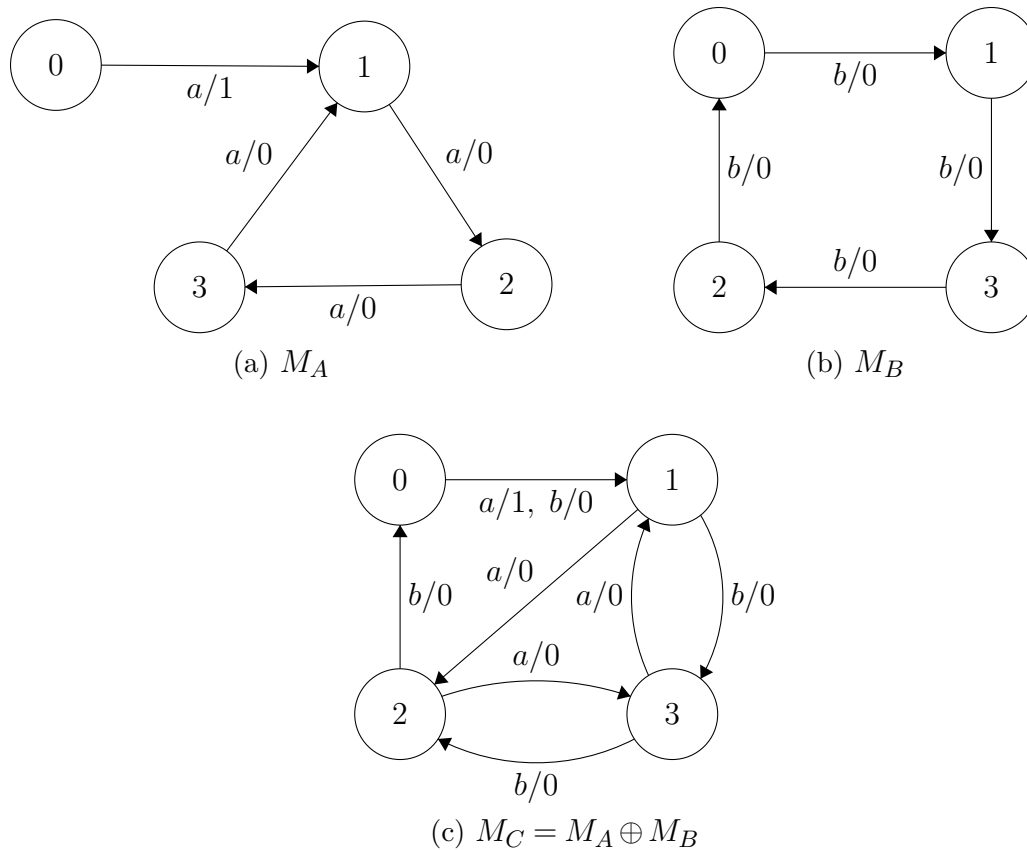
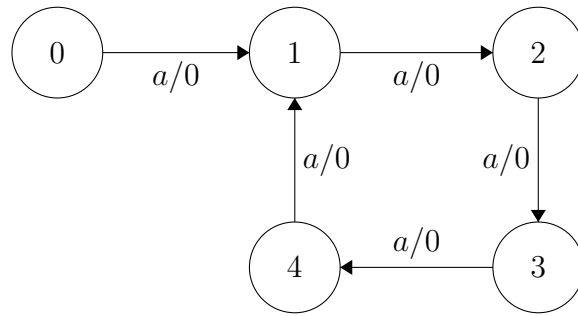
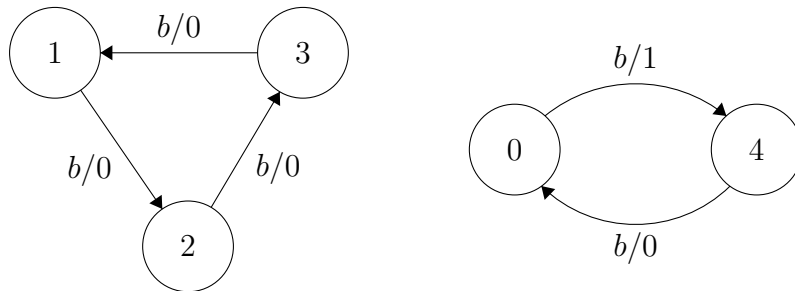


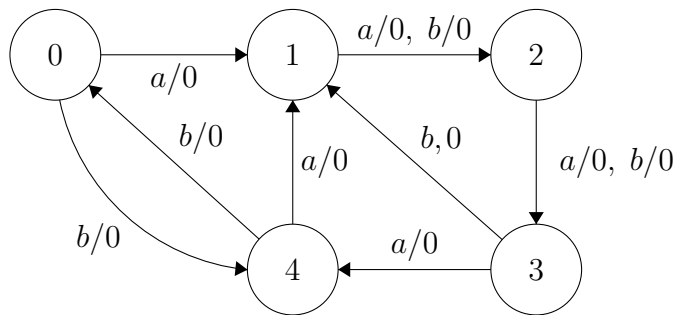
Figure 4.2 Two unary FSMs $M_A, M_B \in \mathcal{M}$ and a binary FSM $M_C = M_A \oplus M_B$ which hits to $H(5, 2, 2)$



(a) M_A

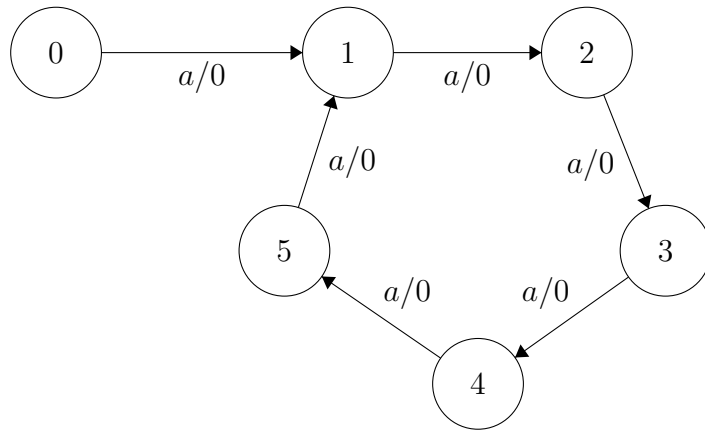


(b) M_B

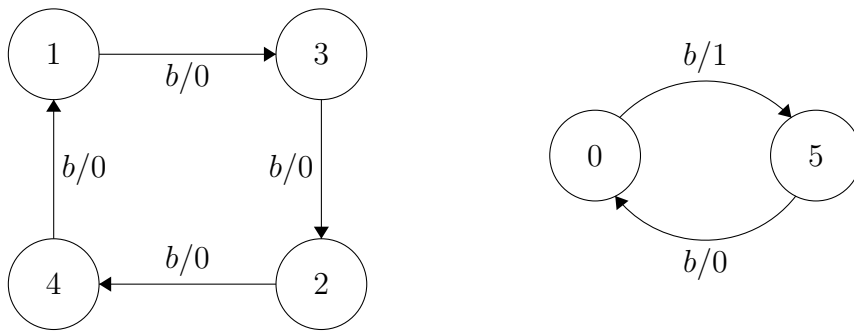


(c) $M_C = M_A \oplus M_B$

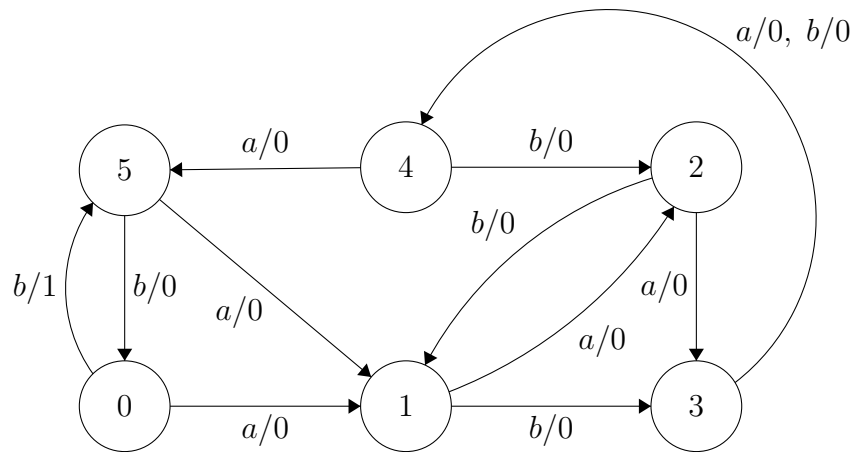
Figure 4.3 Two unary FSMs $M_A, M_B \in \mathcal{M}$ and a binary FSM $M_C = M_A \oplus M_B$ which hits to $H(6, 2, 2)$



(a) M_A

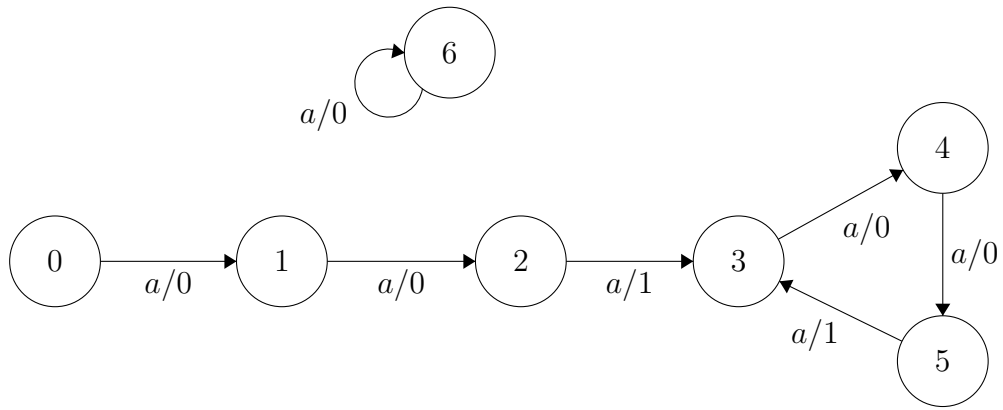


(b) M_B

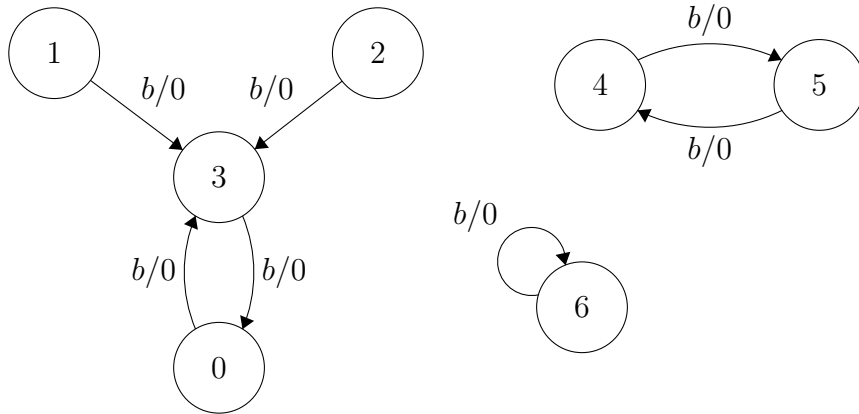


(c) $M_C = M_A \oplus M_B$

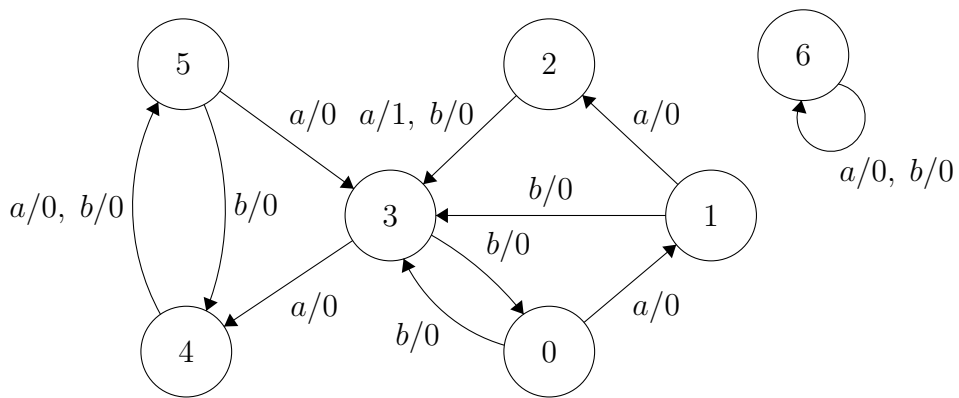
Figure 4.4 Two unary FSMs $M_A, M_B \in \mathcal{M}$ and a binary FSM $M_C = M_A \oplus M_B$ which hits to $H(7, 2, 2)$



(a) M_A

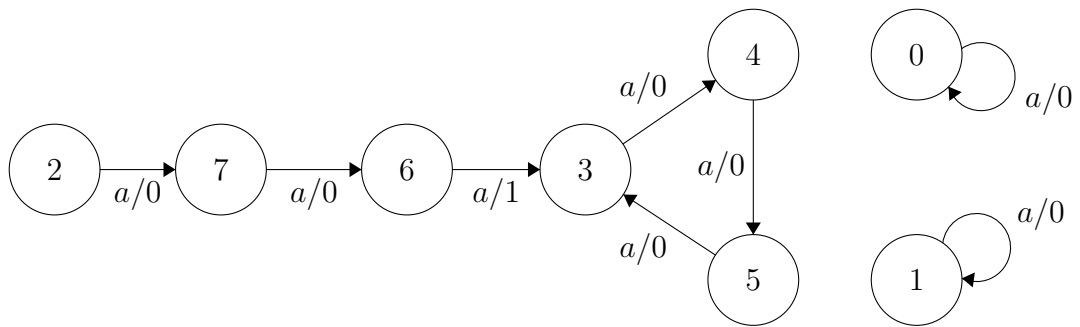


(b) M_B

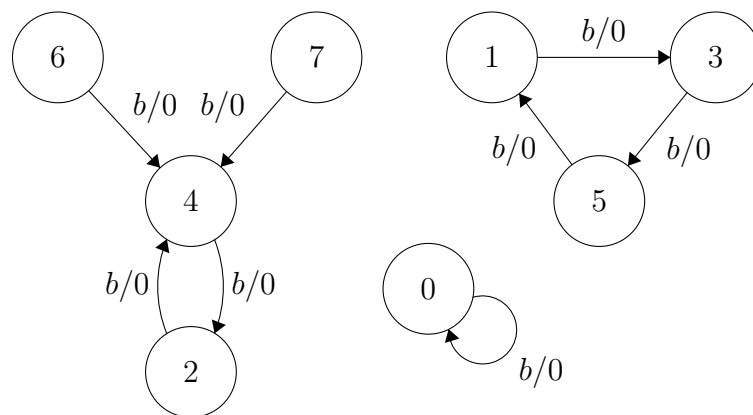


(c) $M_C = M_A \oplus M_B$

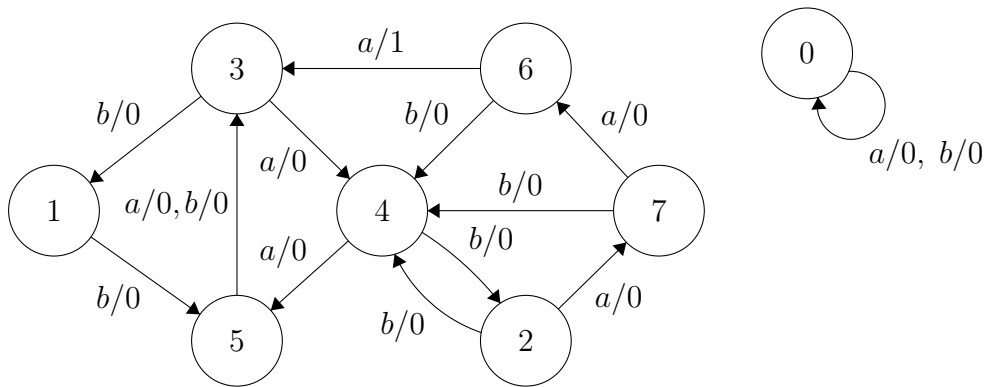
Figure 4.5 Two unary FSMs $M_A, M_B \in \mathcal{M}$ and a binary FSM $M_C = M_A \oplus M_B$ which hits to $H(8, 2, 2)$



(a) M_A



(b) M_B



(c) $M_C = M_A \oplus M_B$

You can find the pseudocode of our algorithm below.

Algorithm 1: Phase 1 of General Algorithm

input : \mathcal{U} \triangleright \mathcal{U} is set of all non-isomorphic unary automata with n states
input : $H(n-1, 2, 2)$ \triangleright upper bound for shortest homing sequence of all FSMs with $n-1$ states, 2 inputs and 2 outputs
output: \mathcal{U}' \triangleright \mathcal{U}' will be the set of non-isomorphic unary automata which can possibly hit to $H(n, 2, 2)$ with some output function
output: Λ^{S1} \triangleright Λ^{S1} will be the set of single-one output functions which can possibly hit to $H(n, 2, 2)$ with according automata
output: Λ^{M1} \triangleright Λ^{M1} will be the set of multi-one output functions which can possibly hit to $H(n, 2, 2)$ with according automata
output: H_n \triangleright H_n is the current conjecture of $H(n, 2, 2)$

- 1 $H_n = H(n-1, 2, 2)$
- 2 $\mathcal{U}' = \emptyset, \Lambda^{S1} = \emptyset, \Lambda^{M1} = \emptyset$
- 3 **forall** $A \in \mathcal{U}$ **do**
- 4 **if** A can not be eliminated by Theorem 3 after possible synchronization **then**
- 5 $\mathcal{U}' = \mathcal{U}' \cup \{A\}$
- 6 $\Lambda_A^{S1} = \emptyset$
- 7 $\Lambda_A^{M1} = \emptyset$
- 8 **forall** $\lambda \in \Lambda$ \triangleright Λ is set of allowed output functions for unary automata **do**
- 9 $M = A \uplus \lambda$ \triangleright Addition means adding outputs to transitions according to selected unary automaton and output function
- 10 **if** M can not be eliminated by Theorem 3 after possible homing **then**
- 11 **if** λ is a single-one **then**
- 12 $\Lambda_A^{S1} = \Lambda_A^{S1} \cup \{\lambda\}$
- 13 **if** λ is a multi-one **then**
- 14 $\Lambda_A^{M1} = \Lambda_A^{M1} \cup \{\lambda\}$
- 15 $\Lambda^{S1} = \Lambda^{S1} \cup \{(A, \Lambda_A^{S1})\}$
- 16 $\Lambda^{M1} = \Lambda^{M1} \cup \{(A, \Lambda_A^{M1})\}$

Algorithm 2: Phase 2 of General Algorithm (Round 1)

input : \mathcal{U}'
input : Λ^{S1}
input : H_n
output: Updated H_n

- 1 **forall** $A, A' \in \mathcal{U}'$ **do**
- 2 **forall** permutations π **do**
- 3 $A^2 = A \oplus A'_\pi$ \triangleright Unary automata superimposition to generate binary automaton.
- 4 **if** not (A^2 has synchronizing sequence with length less than H_n) **then**
- 5 Let Λ_A^{S1} be the set of single-one output functions for automaton A that could not be eliminated by Theorem 6 and Theorem 8, i.e. $(A, \Lambda_A^{S1}) \in \Lambda^{S1}$
- 6 **forall** $\lambda \in \Lambda_A^{S1}$ \triangleright for each such single-one output function λ **do**
- 7 $M = A \uplus \lambda$ \triangleright Extending A with outputs to generate unary FSM M
- 8 $M' = A'_\pi \uplus \lambda^Z$ \triangleright λ^Z is all-zeroes
- 9 $M^2 = M \oplus M'$
- 10 **if** M^2 is minimal **then**
- 11 Find shortest homing sequence \bar{x} of M^2
- 12 **if** $|\bar{x}|$ of M^2 is greater than H_n **then**
- 13 $H_n = |\bar{x}|$
- 14
- 15 **else**
- 16 **forall** $\lambda \in \Lambda_A^{S1} \cup \Lambda_A^{M1}$ **do**
- 17 $M' = A'_\pi \uplus \lambda_\pi$
- 18 $M^2 = M \oplus M'$
- 19 **if** M^2 is minimal **then**
- 20 Find shortest homing sequence \bar{x} of M^2
- 21 **if** $|\bar{x}|$ of M^2 is greater than H_n **then**
- 22 $H_n = |\bar{x}|$
- 23
- 24
- 25
- 26 **forall** $\lambda \in \Lambda_{A'}^{S1}$ \triangleright for each such single-one output function λ **do**
- 27 $M = A \uplus \lambda^Z$ \triangleright λ^Z is all-zeroes
- 28 $M' = A'_\pi \uplus \lambda_\pi$
- 29 $M^2 = M \oplus M'$
- 30 **if** M^2 is minimal **then**
- 31 Find shortest homing sequence \bar{x} of M^2
- 32 **if** $|\bar{x}|$ of M^2 is greater than H_n **then**
- 33 $H_n = |\bar{x}|$
- 34
- 35 **else**
- 36 **forall** $\lambda \in \Lambda_A^{S1} \cup \Lambda_A^{M1}$ **do**
- 37 $M = A \uplus \lambda$
- 38 $M^2 = M \oplus M'$
- 39 **if** M^2 is minimal **then**
- 40 Find shortest homing sequence \bar{x} of M^2
- 41 **if** $|\bar{x}|$ of M^2 is greater than H_n **then**
- 42 $H_n = |\bar{x}|$
- 43
- 44
- 45
- 46

Here, we create binary FSMs by superimposing 2 unary FSMs M, M' which are generated by extending all the permutations of an automaton A' with all-zeroes outputs an another automaton A with single-one outputs respectively for all $A, A' \in \mathcal{U}'$ to find their shortest homing sequence.

For all non-minimal binary FSMs that are generated by process explained above, we change all-zeroes outputs of A' with remaining output functions and repeat the same process above by superimposing unary FSMs and finding their shortest homing sequence.

Here, we create binary FSMs by superimposing 2 unary FSMs M, M' which are generated by extending all the permutations of an automaton A' with single-one outputs an another automaton A with all-zeroes outputs respectively for all $A, A' \in \mathcal{U}'$ to find their shortest homing sequence.

For all non-minimal binary FSMs that are generated by process explained above, we change all-zeroes outputs of A with remaining output functions and repeat the same process above by superimposing unary FSMs and finding their shortest homing sequence.

5. CONCLUSION & FUTURE WORK

We know that upper bound for the length of a shortest homing sequence for minimal FSMs with n states is $n(n-1)/2$ which was found by Hibbard (1961). This bound is actually a tight bound but all the examples that hits to this bound have $n-1$ input symbols. Therefore, we were curious about the upper bound for the FSMs with number of input symbols less than $n-1$ ($p < n-1$). We did an experimental research as we generated all the FSMs that can hit to bound, eliminate some of them with the explained theorems in the paper and find the shortest homing sequences of remaining ones for $n \in \{3, 4, 5, 6, 7, 8, 9, 10\}$, $p = 2$ and $o = 2$. As a result of the research, it is certain that Hibbard's bound is not a tight bound for FSMs with $p < n-1$. Although such an experimental work does not show this claim for all n , it at least shows that $n(n-1)/2$ is not a tight bound for all number of states n , when the number of inputs is less than $n-1$.

In the experiments we could consider small state sizes only, since there are $(n \times o)^{(n \times p)}$ FSMs for given number of states n , input symbols p and output symbols o . In addition, finding shortest homing sequence for a given FSM is known to be an NP-hard problem. Even tough for the state sizes we consider, finding the shortest homing sequences for one FSM can easily be handled using an exponential time algorithm, this algorithm is still at a critical point in our code, both performance and correctness purposes. That's why we double checked the shortest homing sequences of FSMs with 2 different algorithms for FSMs up to 7 states. After increasing our confidence in the implementation of the shortest homing sequence algorithm in this way, we omit this double-check for the experiments in FSMs with 8 or more states, due to the huge FSM set size for these number of states.

Currently, we do not know the generating function for the sequence $H(n, 2, 2)$ we obtained by our experiments. The sequence $H(n, 2, 2)$ is an interesting sequence since there was no such sequence in OEIS (2019). Another interesting finding is the structure of the FSMs that hit to this bound. When Hibbard stated his $n(n-1)/2$ bound, he also provided a class of FSMs with a certain structure (see Figure 1.1) that hits to this bound. Unfortunately, we could not identify such a class of FSMs

with a regular structure hits to the bounds for $H(n, 2, 2)$.

We believe that $H(n, p, o') < H(n, p, 2)$ where $o' > 2$. Intuitively it makes harder to separate states with less output symbols. Hence, we can conclude that $H(n, 2, o) = n - 1$ by using Theorem 9 if we accept $H(n, p, o') < H(n, p, 2)$. However, we don't have a formal proof for this claim. Similarly, we also believe that all the FSMs that hit $H(n, p, o)$ have single-one output function for entire set of input symbols X or more formally $C_X = 1$. Although the experiments that we have done so far shows this claim to be true, we don't have a proof for this idea either.

For future work, one can find more elimination methods for FSMs to check FSMs with larger number of states, make a deeper analysis for the FSMs that hit to the bound. Trying to prove the ideas explained in the previous paragraph would be a great improvement since the number of FSMs will be eliminated after those assumptions will be huge. Obviously, an ultimate solution to the problem would be a mathematical proof for the upper bound on the length of shortest homing sequences for FSMs with a constant number of inputs.

BIBLIOGRAPHY

- Broy, M., Jonsson, B., Katoen, J., Leucker, M., & Pretschner, A. (Eds.). (2005). *Model-Based Testing of Reactive Systems, Advanced Lectures*, volume 3472 of *Lecture Notes in Computer Science*. Springer.
- Černý, J. (1964). Poznámka k homogénnym experimentom s konečnými automatmi. *Matematicko-fyzikálny časopis*, 14(3), 208–216.
- Černý, J., Pirická, A., & Rosenauerová, B. (1971). On directable automata. *Kybernetika*, 7(4), 289–298.
- Chow, T. S. (1978). Testing software design modeled by finite-state machines. *IEEE Trans. Software Eng.*, 4(3), 178–187.
- Çirisci, B., Emek, M. Y., Sorguç, E., Kaya, K., & Yenigün, H. (2019). Using synchronizing heuristics to construct homing sequences. In Hammoudi, S., Pires, L. F., & Selic, B. (Eds.), *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2019, Prague, Czech Republic, February 20-22, 2019.*, (pp. 362–369). SciTePress.
- Eppstein, D. (1990). Reset sequences for monotonic automata. *SIAM J. Comput.*, 19(3), 500–510.
- Gonenc, G. (1970). A method for the design of fault detection experiments. *IEEE transactions on Computers*, 100(6), 551–558.
- Harary, F. & Palmer, E. (2014). *Graphical Enumeration*. Elsevier Science.
- Harel, D. & Politi, M. (1998). *Modeling Reactive Systems with Statecharts: The Stateate Approach* (1st ed.). New York, NY, USA: McGraw-Hill, Inc.
- Hennine, F. (1964). Fault detecting experiments for sequential circuits. In *1964 Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, (pp. 95–110). IEEE.
- Hibbard, T. N. (1961). Least upper bounds on minimal terminal state experiments for two classes of sequential machines. *J. ACM*, 8(4), 601–612.
- Hierons, R. M. & Ural, H. (2006). Optimizing the length of checking sequences. *IEEE Transactions on Computers*, 55(5), 618–629.
- Kisielewicz, A. & Szykuła, M. (2013). Generating small automata and the černý conjecture. In Konstantinidis, S. (Ed.), *Implementation and Application of Automata*, (pp. 340–348)., Berlin, Heidelberg. Springer Berlin Heidelberg.
- Kohavi, Z. (1978). *Switching and Finite Automata Theory*. New York: McGraw-Hill.
- Krichen, M. (2005). State identification. In Broy et al. (2005), (pp. 35–67).
- Kudłacik, R., Roman, A., & Wagner, H. (2012). Effective synchronizing algorithms. *Expert Systems with Applications*, 39(14), 11746–11757.
- Lee, D. & Yannakakis, M. (1994). Testing finite-state machines: State identification and verification. *IEEE Transactions on computers*, 43(3), 306–320.
- Lee, D. & Yannakakis, M. (1996). Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84(8), 1090–1123.
- Moore, E. F. (1956). Gedanken-experiments on sequential machines. *Automata studies*, 34, 129–153.
- OEIS (2019). The on-line encyclopedia of integer sequences.
- Roman, A. (2009). Synchronizing finite automata with short reset words. *Applied Mathematics and Computation*, 209(1), 125–136.

- Roman, A. & Szykuła, M. (2015). Forward and backward synchronizing algorithms. *Expert Systems with Applications*, 42(24), 9512–9527.
- Sandberg, S. (2005). Homing and synchronizing sequences. In Broy et al. (2005), (pp. 5–33).
- Simao, A. & Petrenko, A. (2010). Checking completeness of tests for finite state machines. *IEEE Transactions on Computers*, 59(8), 1023–1032.
- Trahtman, A. (2004). Some results of implemented algorithms of synchronization. *10th Journees Montoises d'Inform.*
- Ural, H., Wu, X., & Zhang, F. (1997). On minimizing the lengths of checking sequences. *IEEE Transactions on Computers*, 46(1), 93–99.