

Федеральное государственное автономное образовательное учреждение высшего образования

**«Уральский Федеральный Университет  
имени первого Президента России Б.Н.Ельцина»**

Институт естественных наук и математики  
Кафедра алгебры и фундаментальной информатики

На правах рукописи

**Шабана Ханан Магди Дарвиш**

**СИНХРОНИЗАЦИЯ ЧАСТИЧНЫХ  
И НЕДЕТЕРМИНИРОВАННЫХ АВТОМАТОВ:  
ПОДХОД НА ОСНОВЕ SAT-РЕШАТЕЛЕЙ**

Специальность 05.13.17 — Теоретические основы информатики

Диссертация на соискание учёной степени  
кандидата физико-математических наук

Научный руководитель  
доктор физико-математических наук  
профессор Волков Михаил Владимирович

Екатеринбург, 2020

Federal State Autonomous Educational Institution of Higher Education  
**«Ural Federal University**  
**named after the first President of Russia B.N.Yeltsin»**  
Institute of Natural Science and Mathematics  
Chair of Algebra and Theoretical Computer Science

Retaining manuscript rights

**Hanan Magdy Darwish Shabana**

**SYNCHRONIZATION OF PARTIAL  
AND NON-DETERMINISTIC AUTOMATA:  
A SAT-BASED APPROACH**

05.13.17 — Theoretical Foundations of Computer Science

A Thesis Submitted for the Degree  
of Candidate of Physical and Mathematical Sciences

Supervisor:  
Doctor of Physical and Mathematical Sciences  
Professor Volkov Mikhail Vladimirovich

Ekaterinburg, 2020

# Contents

<b>Introduction</b>	<b>6</b>
Relevance of the topic . . . . .	6
Degree of development of the topic . . . . .	15
Goals and objectives of the thesis. SAT-solver method . . . . .	16
Overview of the thesis . . . . .	19
The main achievements of the thesis . . . . .	21
Publications . . . . .	22
Approbation at seminars and conferences . . . . .	23
Scientific novelty . . . . .	24
Degree of correctness of the results . . . . .	24
Theoretical and practical importance . . . . .	25
Research methods . . . . .	25
Length and structure of thesis . . . . .	25
Acknowledgments . . . . .	26
<b>1 Preliminaries</b>	<b>27</b>
1.1 Complexity classes . . . . .	27
1.2 Satisfiability . . . . .	30
1.3 Finite automata . . . . .	32

1.4	Synchronizing DFA . . . . .	35
1.5	NFA Synchronization . . . . .	41
1.6	Complexity of synchronization in finite automata . . . . .	45
1.7	Shortest synchronizing word . . . . .	46
1.7.1	Shortest synchronizing word for DFAs . . . . .	46
1.7.2	Shortest synchronizing word for NFA . . . . .	48
1.8	Careful synchronization . . . . .	49
1.9	Exact synchronization . . . . .	54
1.10	Checking careful synchronization . . . . .	55
1.11	Testing exact synchronization . . . . .	57
<b>2</b>	<b>Synchronization of PFAs</b>	<b>62</b>
2.1	Carefully synchronizing words . . . . .	62
2.2	Exactly synchronizing words . . . . .	69
2.3	Ladder encoding . . . . .	76
<b>3</b>	<b>Experimental study in PFAs synchronization</b>	<b>78</b>
3.1	General settings of our experiments . . . . .	78
3.2	Experiments and implementation . . . . .	83
3.3	Generating random PFAs . . . . .	83
3.4	Experimental results for randomly generated PFAs and their analysis . . . . .	85
3.4.1	Series 1: Probability of synchronization . . . . .	85
3.4.2	Series 2: Average length of the shortest synchronizing word . . . . .	92
3.4.3	Series 3: Influence of the input alphabet size . . . . .	96
3.4.4	Series 4: Influence of density . . . . .	97

3.5	Slowly synchronizing automata and benchmarks . . . . .	100
3.6	A comparison with the partial power automaton method .	106
<b>4</b>	<b>Synchronization problems of NFAs</b>	<b>109</b>
4.1	Modeling NFA computation as SAT: Variables . . . . .	109
4.2	Modeling NFA computation as SAT: Clauses . . . . .	111
4.3	Propositional logic formulas for rules . . . . .	114
4.4	CNF formulas . . . . .	116
4.5	NFA-synchronization problems . . . . .	118
4.5.1	$D_3$ -synchronization . . . . .	118
4.5.2	$D_2$ synchronization . . . . .	122
4.5.3	$D_1$ -synchronization . . . . .	126
4.6	Example of the CNF table for DiW problems . . . . .	128
<b>5</b>	<b>Experiments in NFA synchronization</b>	<b>132</b>
5.1	NFA Generation . . . . .	134
5.2	Uniform Model results . . . . .	137
5.3	Poisson Model results . . . . .	138
5.3.1	$D_3$ results . . . . .	140
5.3.2	$D_2$ results . . . . .	141
5.4	Enhancement of the algorithm . . . . .	142
	<b>Conclusion</b>	<b>146</b>
	<b>Bibliography</b>	<b>148</b>

# Introduction

## Relevance of the topic

Finite automata are mathematical models for a lot of discrete dynamical systems arising in robotics, communication protocols, biology, computer hardware design, artificial intelligence, linguistics, and other areas. These systems are known as finite-state transition systems. Such a system consists of a finite number of states and transitions rules. The state of the system at any moment of time is changed, responding to external input. The transitions rules determine the transition between those states, according to the input.

In the design of the control systems modeled by finite automata, the essential behavior of the finite automaton is the mapping of external sequences into internal states. When for some reasons the automaton fails to take the correct transition (there is a failure in the system), the person responsible of the system decides to direct the system to a known state from which he or she can restore control over the system. He or she can do that if and only if the system is *synchronizing*. Hence, synchronization is a significant concept for a lot of systems as it makes the systems more robust.

Synchronization frequently appears in the following situation. Suppose that a system is composed of sub-systems that work as identical mechanisms but may be in different states. For some reason, we want all of these sub-systems to be at the same state at the same time. This task can be easily accomplished if the given system is synchronizing. In this situation we can say that the synchronization of a system means that all parts of the system are in agreement regarding the present state of the system.

Now we switch from an informal discussion of synchronization to precise definitions. An *automaton* is defined as a triple  $\langle Q, \Sigma, \delta \rangle$  where  $Q$  is the *state set*,  $\Sigma$  is the *input alphabet* and  $\delta$  is the transition function that defines the action of elements of  $\Sigma$  on the elements of  $Q$ . The result of the action of an input letter  $a \in \Sigma$  at a state  $q \in Q$  is denoted as  $\delta(q, a)$  (or  $q.a$  for simplicity). This action is naturally extended to define the action of a word in  $\Sigma^*$  at any state in  $Q$ . (Here  $\Sigma^*$  stands for the set of all words over the alphabet  $\Sigma$ , including the empty word.) We do not need to specify initial nor final states of the automaton as the study of synchronization considers, in a sense, all states being initial and final.

An automaton is *synchronizing* if it has an input word that brings it to a unique state regardless of the state of the automaton before reading this word. Figure 1 shows an example of a synchronizing automaton with four states 0, 1, 2, 3 and two input letters  $a$  and  $b$ . It is easy to verify that the word  $baaa$  transfers the automaton from any state to state 0. Such a word is called a *synchronizing word* or a *reset word*.

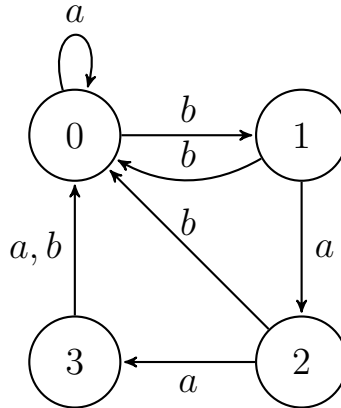


Figure 1: A synchronizing automaton

The concept of synchronizing words for finite automata has received lots of attention during the last years as they appear in a wide range of applications. Here we briefly describe a few examples of such applications.

**Industrial robotics.** Synchronizing words reset the automaton to a unique state regardless of its present state. These words are useful in industrial robotics see [30,61,62]. In such industrial issues, synchronizing automata are widely used to model and design feeders, sorters, and orienters that work with flows of certain objects carried by a conveyor. We borrow an illustrative example from [2]. Consider an automatic line for manufacturing of a device. Suppose a certain part of the device has the shape shown in Figure 2. Such parts arrive to the conveyor in random orientations. These different orientations of the part are illustrated in Figure 3. For assembly these parts need to be oriented in the same way. Assume that each part needs to be in the second (from the left) position from Figure 3.



Thus, one has to design an orienter that senses the position of the incoming part and then rotates it to the prescribed position. Generally speaking, such an orienter is complicated as its mechanism is dependent on the shape of the part. Practical considerations favor methods which require little or no sensing, employ simple tools, and are as robust as possible. The desired orienter is one whose processing is independent of the initial orientation of the part. Now we quote from [2]. “For our particular case, these goals can be achieved as follows. We put parts to be oriented on a conveyer belt which takes them to the assembly point and let the stream of the parts encounter a series of passive obstacles placed along the belt. We need two type of obstacles: tall (T) and short (S). A tall obstacle should be tall enough in order that any part on the belt encounters this obstacle by its rightmost low angle (we assume that the belt is moving from left to right). Being carried by the belt, the part then is forced to turn  $90^\circ$  clockwise as shown in Figure 4. A short obstacle has the same effect whenever the part is in the ‘bump-down’ orientation (the first from the left in Figure 3); otherwise it does not touch the part which therefore passes by without changing the orientation”. Thus, we can construct the orienter as an automaton  $\mathcal{A}$  whose state set consists of the different orientations, whose input alphabet is  $\{T,S\}$ , and whose transition function is defined by the action of each obstacle on different orientations of the part. The automaton  $\mathcal{A}$  is described in Figure 5. It is easy to verify that this automaton is synchronizing and the sequence S–T–T–T–S–T–T–T–S of obstacles is a synchronizing word that yields the desired sensorless orienter.

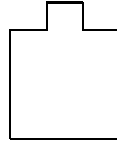


Figure 2: A polygonal part



Figure 3: Four possible orientations; the “correct” orientation is the second one from the left.

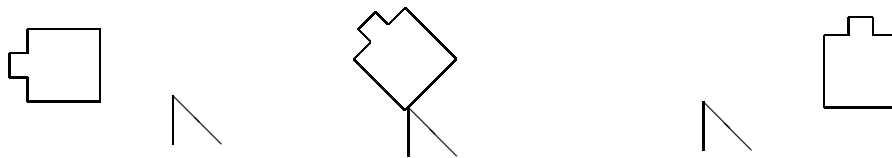


Figure 4: The action of a tall obstacle

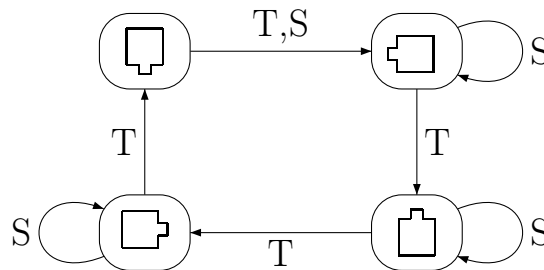


Figure 5: Sensorless orienter  $\mathcal{A}$

**Coding theory.** A word  $u$  over an alphabet  $\Sigma$  is said to be a *prefix* of a word  $w$  over  $\Sigma$  if  $w$  can be written as  $w = uv$  for some  $v \in \Sigma^*$ . A *prefix code*  $X$  over  $\Sigma$  is a set of non-empty words from  $\Sigma^*$  such that no word of  $X$  is a prefix of another word of  $X$ . It is *maximal* if it is not contained in another prefix code over the same alphabet. A maximal prefix code  $X$  over  $\Sigma$  is called *synchronized* if there is a word  $z$  such that for any word  $y \in \Sigma^*$ , the word  $yz$  can be decomposed as a product of words from  $X$ . Such a word  $z$  is called a *synchronizing word* for  $X$ . When coding a stream of data with a synchronized code, we have a guarantee that we can recover after a loss of synchronization between the decoder and the coder caused by channel errors [9]. In the case of such a loss, it suffices to transmit a synchronizing word and the following symbols will be decoded correctly. Moreover, this may happen automatically when a sufficiently long word has been read. For synchronizing codes, the probability that a word  $w \in \Sigma^*$  contains a synchronizing word as a factor tends to 1 as the length of  $w$  increases [13].

For an illustration, we consider the following example from [42]. Let  $\Sigma = \{0, 1\}$  and  $X = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$ . Then  $X$  is a maximal prefix code over  $\Sigma$  and it is easy to check that each of the words  $010, 011110, 01111110, \dots$  is a synchronizing word for  $X$ . For instance, if the code word  $000$  has been sent but, due to a channel error, the word  $100$  has been received, the decoder interprets  $10$  as a code word, and thus, loses synchronization. However, with a high probability this synchronization loss only propagates for a short while; in particular, the decoder definitely re-synchronizes as soon as it encounters one of the segments  $010, 011110, 01111110, \dots$  in the received stream of symbols. A few samples of such streams are shown in

Figure 6, where vertical lines show the partition of each stream into code words and the boldfaced code words indicate the position at which the decoder re-synchronizes.

Sent	000		0010		<b>0111</b>		...								
Received	10		000		10		<b>0111</b>		...						
Sent	000		0111		110		0011		000		10		<b>110</b>		...
Received	10		0011		111		000		110		0010		<b>110</b>		...
Sent	000		000		111		<b>10</b>		...						
Received	10		000		0111		<b>10</b>		...						

Figure 6: Automatic synchronization

There is a strong connection between codes and finite automata, see the monograph [9] for an extensive treatment. Here we limit ourselves to just one example of such a connection. If  $X$  is a finite maximal prefix code over an alphabet  $\Sigma$ , then its decoding can be implemented by a finite automaton  $\mathcal{A}_X$  whose set of states  $Q$  is the set of all proper prefixes of the words in  $X$  (including the empty word  $\varepsilon$ ) and whose transitions are defined as follows: for  $q \in Q$  and  $a \in \Sigma$ ,

$$q.a = \begin{cases} qa & \text{if } qa \text{ is a proper prefix of a word of } X, \\ \varepsilon & \text{if } qa \in X, \end{cases}$$

$X$  is a synchronized code if and only if the automaton  $\mathcal{A}_X$  has a synchronizing word that resets it to the state  $\varepsilon$ .

**Model based testing.** Other problems for which synchronizing automata and synchronizing words are important can be found in model conformance testing [8, 16, 51, 52]. These problems consist of some testings to ensure that a system verifies its prerequisites. When the abstract behavior of an interactive system is implemented by finite automata, there are various methods to derive some test sequences with high fault coverage. These methods construct a test sequence to be applied when the implementation under test (IUT) is in a certain state. Therefore it is required to bring the IUT to this particular state regardless of its initial state which can be accomplished by using a synchronizing sequence for the IUT. Figure 7 shows the general technique of model based testing process.

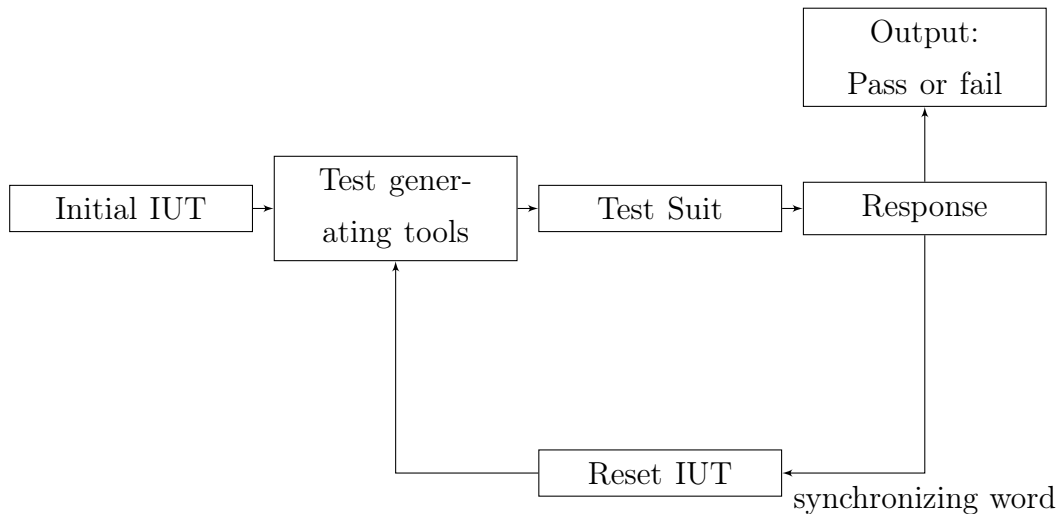


Figure 7: Model based testing process

A popular automata model in the area of model based testing is the *finite state machine* (FSM) model, see, e.g., [45–50, 89, 90]. A FSM is basically a nondeterministic automaton with output, and a common

restriction to FSMs used in model based testing is *observability*: a FSM  $\mathcal{F}$  with input alphabet  $I$  and output alphabet  $O$  is said to be *observable* if for each state  $q$  of  $\mathcal{F}$  and for each input/output pair  $(i, o) \in I \times O$ , at most one action of the input  $i$  at  $q$  produces the output  $o$ . It is easy to see that from the synchronization viewpoint, such an observable FSM is equivalent to a partial automaton with the same state set and the alphabet  $\Sigma = I \times O$ .

We think that this brief survey suffices to support the claim that synchronizing automata serve as simple yet adequate models of error-resistant systems in many applied areas. At the same time, synchronizing automata surprisingly arise in some parts of pure mathematics and theoretical computer science (symbolic dynamics, theory of substitution systems, formal language theory). From both applied and theoretical viewpoints, the key question is to find the optimal, i.e., shortest synchronizing word for a given synchronizing automaton. Under standard assumptions of complexity theory, this optimization question is known to be computationally hard. As it is quite common for hard problems of applied importance, there have been many attempts to develop practical approaches to the question. These approaches have been based on certain heuristics [1, 39–41] and/or popular techniques, including (but not limiting to) binary decision diagrams [68], genetic and evolutionary algorithms [44, 73], satisfiability solvers [77], answer set programming [32], hierarchical classifiers [69], and machine learning [70].

## Degree of development of the topic

So far, most of the published research on synchronization has focused on systems that are modeled as complete deterministic automata (DFAs). We refer to the survey [88] and the chapter [42] of the forthcoming ‘Handbook of Automata Theory’ for a discussion of synchronizing complete deterministic automata as well as their diverse connections and applications. In such systems a full specification of the system is provided; response of the system to any input is uniquely determined by the current state and the incoming input. Recently a great deal of attention has also been given to synchronization of partially specified systems, in which only a subset of the system’s events are available for external observation. These systems are also known as nondeterministic systems. In such systems, the knowledge of the current state and the incoming input is insufficient to uniquely determine the next state. Such nondeterminism arises due to un-modeled system dynamics and/or partial observation. For example, a machine in a manufacturing system may incur a partial undetectable failure while performing a certain task. This can be modeled by having a nondeterministic transition on the task completion event, leading to two successor states depending on whether or not the failure occurred while completing the task. Other kind of nondeterministic systems is partial deterministic system in which there are some events that are not allowed for some states. Complete deterministic automata cannot be used in modeling partially specified systems. For these systems, other classes of automata are used: *nondeterministic automata* (NFAs) and *partial deterministic automata* (PFAs). Nowadays, synchronization of these automata has become an interesting research topic.

## Goals and objectives of the thesis.

### SAT-solver method

In this thesis we **focus on the case of partial and nondeterministic automata**. We investigate synchronization of these automata. In contrast to synchronization of complete deterministic automata, synchronization problems for nondeterministic or even partial deterministic automata are much harder. For these automata, there is more than one version of synchronization. An automaton may be synchronizing with respect to one version but not synchronizing from the view point of another version.

Some of problems that are frequently asked in this area are:

1. *For a version of synchronization  $\mathfrak{B}$ , is a given automaton  $\mathcal{A}$  synchronizing with respect to  $\mathfrak{B}$ ?*
2. *Can we solve Problem 1 in polynomial time as for complete deterministic automata?*
3. *How does the degree of nondeterminism in  $\mathcal{A}$  affect being synchronizing?*
4. *If the automaton  $\mathcal{A}$  already belongs to  $\mathfrak{B}$ , does it have a synchronizing word of a specified length?*
5. *What is the length of the shortest synchronizing word for  $\mathcal{A}$ ?*
6. *For synchronizing nondeterministic or partial deterministic automata with a given number of states, what is the average length of their shortest synchronizing words?*



In the literature, there are two methods to solve the above problems. The basic one is the power automaton method. It is based on the classical construction of power automata due to Rabin and Scott [71]. This method has delivered many important theoretical results but it is not efficient in practice as the power automaton has an exponential size compared to the size of the input automaton. The second method uses the deterministic automata as tools to solve these problems. Since the synchronization of deterministic automata is extensively studied and upper bounds or the exact length of the shortest synchronizing word for various kinds of such automata are given, there were attempts to solve Problems 3-6 by certain reductions to complete deterministic automata; see [35] and [37] for details. Although these attempts have proved their efficiency in complete nondeterministic automata for a certain version of synchronization, they can not be applied to other versions of synchronization.

In accordance to the hardness of these problems, our motivation was to find tools that have proved to be powerful in dealing with computationally hard issues. Such tools are provided in particular, by SAT solvers; these are computer programs designed to solve the Boolean satisfiability problem (SAT).

SAT is a decision problem in propositional logic on a set of variables  $V$  and clauses  $C$ . The question is: is there a Boolean assignment for  $V$  that satisfies all clauses in  $C$ ? Modern SAT-solvers can solve such decision problems with millions of variables and clauses in a few minutes. Due to this advantage, the following approach to computationally hard problems has become quite popular nowadays: one encodes instances of such problems into instances of SAT that are then fed to a SAT solver.

We refer to this approach as the *SAT-solver method*. SAT-solver method has proved to be very efficient for an extremely wide range of problems of both theoretical and practical importance. Its applications are far too numerous to be listed here; we refer the reader to the survey [29] or to the handbook [6] for some examples of successful applications of SAT-solver method in various areas.

Here we mention only three recent papers that deal with two difficult problems related to finite automata. Geldenhuys, van der Merwe, and van Zijl [26] have used the SAT-solver method to attack the minimization problem for NFAs. In the minimization problem, which is known to be PSPACE-complete [38], an NFA  $\mathcal{A}$  with designated initial and final states is given, and one looks for an NFA of minimum size that accepts the same set of words as  $\mathcal{A}$ . Skvortsov and Tipikin [77] have applied the method to find a synchronizing word of minimum length for a given complete deterministic automaton (DFA) with two input symbols, and Güniçen, Erdem, and Yenigün [32] have extended their approach to DFAs with arbitrary input alphabets. The problem of finding a synchronizing word of minimum length is known to be hard for the complexity class  $\text{FP}^{\text{NP}[\log]}$ , the functional analogue of the class of problems solvable by a deterministic polynomial-time Turing machine that has an access to an oracle for an NP-complete problem, with the number of queries being logarithmic in the size of the input [65].

It should be stressed that neither the encoding of NFAs used in [26] nor the encoding of synchronization used in [32, 77] work for our purposes. Therefore, we have had to invent essentially different encodings specific for the problems addressed in the thesis.

## Overview of the thesis

In **Chapter 1** we give an overview of relevant aspects related to synchronization of automata. In order to place our research in a proper perspective, we start by synchronization of fully specified automata (complete deterministic automata). Then we switch to the nondeterministic automata where we give a survey on the synchronization issues and overview results of recent papers published in this area. From this discussion we conclude that Problem 1 above is PSPACE-complete and hence the answer to Problem 2 is NO in general. Consequently all the remaining problems are computationally hard.

**Chapter 2** investigates synchronization of partial deterministic automata (PFAs). For these automata there are two versions of synchronizations called Careful and Exact synchronization. We introduce encodings that model these versions of synchronization as SAT problems. The main results of the chapter are **Theorem 2.1** and **Theorem 2.2** that prove the adequacy of these encodings.

**Chapter 3** presents an experimental study of synchronization of PFAs. We performed a series of experiments to find an approximation of the shortest synchronizing word for a given automaton in each version. It also shows the probability of being synchronizing for each version. The results of another series of experiments is presented. These results show the influence of increasing partiality in the automaton on the synchronization and the length of the shortest synchronizing word for each version. At the end of chapter we give some of benchmarks used to prove the efficiency of our algorithm comparing to the other known algorithms. The experiments have also allowed us to find two new infinite series of slowly synchronizing partial deterministic automata. The results of our

algorithm match the result of brute-force algorithm in [86]; the latter algorithm is used only for one version of synchronization. Along with experimental results and their discussion, the chapter also contains two theoretical results: **Proposition 3.3** and **Proposition 3.4** that find the exact length of the shortest synchronizing word for two series of slowly synchronizing partial deterministic automata.

**Chapter 4** studies the synchronization problems in nondeterministic automata. Synchronization of nondeterministic automata is more complicated than that of partial deterministic automata since the latter automata still have the property that at any moment of time, the automaton can be in one state at most. In nondeterministic automata this is not true in general. There are three different ways of formalizing synchronization for these automata. We present a technique able to simulate these formalizations and to give answers to the above problems. The main results of the chapter are **Theorem 4.2**, **Theorem 4.3** and **Theorem 4.4** that prove the accuracy of this technique.

In complete deterministic automata the known upper bound of the shortest synchronizing word is cubic in the number of states [23,67,81,82] but on average any random DFA has a synchronizing word which length is much less than this bound (see [43] for experimental results and [64] for their partial theoretical explanation). This fact makes Catalano and Jungers in [14] pose the following open problem:

*In nondeterministic automata the upper bound for the length of the shortest synchronizing word is known to be an exponential function in the number of automaton's states but what about the average length of the shortest synchronizing word in different issues of synchronization?*

This problem motivates our experimental research presented in

**Chapter 5.** While the random generation of complete deterministic automata is well understood, the random generation of nondeterministic automata is not that obvious. We used two models for the random generation of NFAs. In each model we show how the average length is affected by the parameters of the model. In each model, a series of experiments has been done. Depending on the result of these experiments and using some standard tools of probability theory we give an approximation to the average length of the shortest synchronizing word in different issues of NFAs synchronization.

## The main achievements of the thesis

1. A systematic approach to the synchronization problems of partial deterministic and nondeterministic automata based SAT-solver.
2. Proving the validity of presented models from the theoretical point of view and using some benchmarks.
3. Two new series of slowly synchronizing partial deterministic automata.
4. Proving complexity of some problem related to synchronizing nondeterministic automata.
5. Extensive experimental studies of synchronization problems for partial deterministic and nondeterministic automata.

## Publications

The main results of the dissertation are published in the following papers:

1. Shabana, H., Volkov, M. V.: Using Sat solvers for synchronization issues in partial deterministic automata. In: Mathematical Optimization Theory and Operations Research, 18th Int. Conf. MOTOR 2019. *Communications in Computer and Information Science (CCIS)*, volume 1090, 103–118. Springer, 2019.
2. Shabana, H.: Exact synchronization in partial deterministic automata. *J. Phys.: Conf. Ser.*, Paper №012047. 1352:1–8, 2019.
3. Shabana, H., Volkov, M. V.: Using Sat solvers for synchronization issues in nondeterministic automata. *Siberian Electronic Math. Reports*, 15:1426–1442, 2018.
4. Shabana, H.:  $D_2$ -synchronization in nondeterministic automata. *Ural Math. J.*, 4(2):99–110, 2018.

The first three of these papers are indexed by Scopus, and the third is indexed also by Web of Science.

In the two papers coauthored with the supervisor, the supervisor suggested the general approach while the author developed, justified and implemented all algorithms, designed and performed all experiments, and analyzed experimental results.

In addition, the following computer programs were officially registered at the Russian Federal Service for Intellectual Property:

5. Шабана Ханан Магди Дарвиш. NFAsync: Программный комплекс для вычисления порога синхронизации недетерминированных конечных автоматов. Свидетельство о государственной регистрации программ для ЭВМ №2018663225 от 24 октября 2018. Дата приоритета 26 июня 2018. Правообладатель УрФУ.
6. Шабана Ханан Магди Дарвиш. Программа OSW для вычисления оптимального синхронизирующего слова для частичного детерминированного автомата. Свидетельство о государственной регистрации программ для ЭВМ №2019663027 от 08 октября 2019. Дата приоритета 25 сентября 2019. Правообладатель УрФУ.

## Approbation at seminars and conferences

The main results of the dissertation were reported at the following conferences and seminars:

1. Seminars of the Department of Algebra and Theoretical Computer Science. Institute of Natural Sciences and Mathematics. Ural Federal University. Yekaterinburg, Russia.
2. International (52-nd) Youth School-Conference of Modern problems in mathematics and its applications, Yekaterinburg, Russia, 2020.
3. International conference of Mathematical Optimization Theory and Operations Research (MOTOR). Obukhovskoe, Russia, 2019

4. International Scientific and Practical Conference on Mathematical Modeling, Programming and Applied Mathematics (MMPAM), Veliky Novgorod, Russia 2019.
5. International (51-st) Youth School-Conference of Modern problems in mathematics and its applications, Yekaterinburg, Russia, 2019.
6. International conference (50-th) Youth School-Conference of Modern problems in mathematics and its applications, Yekaterinburg, Russia, 2018.
7. International Conference and PhD-Master Summer School “Groups and Graphs, Metrics and Manifolds”, Yekaterinburg, Russia, 2017.

## Scientific novelty

All results in Chapters 2–5 of the dissertation are new. Chapter 1 collects known results that are used in the thesis. For a few results in Chapter 1, we were not able to locate their proofs in the literature; in such cases, we have provided our proofs for the sake of completeness.

## Degree of correctness of the results

All theoretical results presented in the thesis are supplied with rigorous mathematical proofs. The adequacy of experimental results is confirmed by the fact that our results match the ones obtained by other researchers who used alternative approaches.



## Theoretical and practical importance

The dissertation is mainly of theoretical importance. The results obtained in it can be used in the theory of finite automata and related areas of theoretical computer science. Computer programs that we developed can serve as prototypes of software products in those information technologies where different types of synchronizing automata are employed.

## Research methods

The thesis utilizes methods from various branches of mathematics and theoretical computer science: automata theory, propositional logic, probability theory, graph theory, and complexity theory.

## Length and structure of thesis

The thesis consists of an introduction, five chapters, a conclusion, and a bibliography of 91 titles; the total number of pages is 160. The text excludes program codes and datasets, which are available under <https://github.com/hananshabana/SynchronizationChecker>.

## Acknowledgments

I would like to thank my supervisor Prof. Dr. Mikhail Volkov for his encouragement, professional guidance, and valuable support during my studies. I wish to thank him for his careful editing that contributed enormously to the production of this thesis.

I would also like to extend my thanks to my colleagues in the Department of Algebra and Theoretical Computer Science for their help.

Last, but not least, I would like to thank my husband for his understanding and love during the years of my studies. His support and encouragement were in the end what made this dissertation possible. My parents and brothers receive my deepest gratitude and love for their dedication and kind support.

This work has been supported by a grant from the Egyptian Government and the Competitiveness Enhancement Program of Ural Federal University, Ekaterinburg, Russia.

# Chapter 1

## Preliminaries

In this chapter we provide a formal introduction to the area of synchronizing finite automata. As automata theory is relevant to computational complexity theory, we start by giving a brief overview on the algorithms and problem complexity. Then we present the definitions of finite automata and the concept of synchronizing words. We recall some classical problems and results related to synchronization. We also provide proofs for a few results whose proofs we did not manage to find in the literature.

### 1.1 Complexity classes

The complexity theory is concerned with the amount of computational resources required to solve computational problems, and to classify problems according to their difficulty. Computational resources is standing for *computational time*, and *computational memory* (space) used in solving the problem.

*Definition 1.1.* **The time complexity**  $f_A^T(n)$  of an algorithm  $A$  for a

problem  $H$  denotes the maximal time that  $A$  needs to solve an instance of  $H$  with an input of size  $n$ .

**The space complexity**  $f_A^S(n)$  denotes the maximal space that  $A$  needs to solve an instance of  $H$  with input of size  $n$ .

It is difficult to determine the exact value of  $f_A^T(n)$  or  $f_A^S(n)$ . Therefore in analysis the complexity of an algorithm the asymptotic value of  $f_A^T(n)$  and  $f_A^S(n)$  are used. Thus, Computational complexity of a problem  $H$  is the asymptotic of  $f_A^T(n)$  and  $f_A^S(n)$  where  $A$  is known to be the best algorithm solve it.

The standard notation used to express the asymptotic complexity of an algorithm is *Big-O* notation. If  $f_A^T(n) = O(g(n))$  this means that there are positive constants  $c$  and  $k$ , such that

$$0 \leq f_A^T(n) \leq cg(n) \quad \forall n \geq k,$$

where  $c, k$  are positive constants and do not depend on  $n$

*Definition 1.2.* A decision problem is a problem that has a Yes/No answer.

In the following we emphasize on some complexity classes of decisions problems, more information about complexity theory and complexity classes may be found in [66]

In classifying the problems according to the computational time we have the classes such as P, NP, NP-hard, coNP, NP-complete,...etc

*Definition 1.3. Complexity class P*

A decision problem  $H$  is in the complexity class P (P-problem) if there is an algorithm  $A$  can solve an instance of it with an input of size  $n$  within  $f_A^T(n) = O(n^k)$ ;  $k$  is a positive integer i.e; p-problem can be solved in

polynomial time.

*Definition 1.4. The complexity class NP*

A decision problem is in NP if its Yes answers can be checked in time that is polynomial in the size of the input. In an NP problem the answer may not be found in polynomial time but the claimed solution can be verified quickly.

*Definition 1.5. Polynomial Reduction*

A problem  $H$  is reducible to a problem  $H'$  if there exists a polynomial-time function  $f$  such that  $x \in H$  if and only if  $f(x) \in H'$ . That reduction is formally defined as  $(H \leq_p H')$

The definition of reduction implies the following lemma

**Lemma 1.1.**    1. If  $H \leq_p H'$  and  $H' \in P$  then  $H \in P$

2. If  $H \leq_p H'$  and  $H \in NP$  then  $H' \in NP$

Using reductions we can compare among problems.

*Definition 1.6. NP-Hard*

A decision problem  $H$  is NP-hard if, for every NP problem  $H'$ ,  $H' \leq_p H$ .

One can say that a problem is NP-hard if it is as hard as the hardest NP problem.

*Definition 1.7. NP-Complete*

A decision problem  $H$  is NP-Complete if,  $H$  is NP and NP-Hard.

The problem is NP-complete if it is NP and can be reduced from any known NP-complete problem.

The previous classes of complexity were focused on the time needed to solve the problem. But there is another resource that is needed for solving

the problems; that is the space required to solve the given problem. In order to classify the problems according to the space needed to solve them, the following classes are arising:

*Definition 1.8. PSPACE*

A problem  $H$  is a PSPACE if it can be solved by an algorithm  $A$  such that  $f_A^S(n) = O(n)$ ; memory complexity of  $A$  is expressed as polynomial function of the input size of  $H$ .

*Definition 1.9.* A problem  $H$  is a PSPACE-hard if there is another problem  $H'$  that is known to be a PSPACE-complete such that  $H'$  can be reduced to  $H$ .

As with NP-hard problem, PSPACE-hard may not be in PSPACE. The problem is a PSPACE-complete if it is PSPACE and PSPACE-hard.

## 1.2 Satisfiability

In this section, we will provide the definitions necessary to understand the satisfiability problem (SAT):

SAT is a decision problem described as a Boolean formula. Any Boolean formula consists of:

1. a set  $V$  of Boolean variables;
2. a set  $O$  of Boolean operators (or connectives) such as  $\neg$  (NOT),  $\wedge$  (AND),  $\vee$  (OR),  $\rightarrow$  (Implication),  $\iff$  (If and only if), etc;

*Example 1.1.*  $\varphi = (x_1 \wedge x_2) \vee (\neg x_3 \iff x_4)$ . The Boolean formula  $\varphi$  has  $V = \{x_1, x_2, x_3, x_4\}$  and  $O = \{\wedge, \vee, \neg, \iff\}$ .

Given a Boolean formula  $\varphi(V, O)$ , let  $\tau : V \rightarrow \{0, 1\}$  be a truth assignment function that assigns a truth value to each variable in  $V$ . A *truth assignment* for the formula  $\varphi$  is a set of values for the variables of  $\varphi$ . A *satisfying assignment* for  $\varphi$  is the truth assignment that makes it true (evaluated 1) and the Boolean formula is called satisfiable if it has a satisfying assignment. Namely, the function  $\tau$  extends to a map  $\varphi \mapsto \{0, 1\}$  (still denoted by  $\tau$ ) via the usual rules of propositional calculus:

$$\begin{aligned}\tau(\neg x) &= 1 - \tau(x), \quad \tau(x \vee y) = \max\{\tau(x), \tau(y)\}, \\ \tau(x \wedge y) &= \min\{\tau(x), \tau(y)\}, \text{ etc.}\end{aligned}$$

An instance of SAT is a pair  $(V, C)$ , where  $V$  is a set of Boolean variables and  $C$  is a collection of clauses over  $V$ . (A *clause* over  $V$  is a disjunction of literals and a *literal* is either a variable in  $V$  or the negation of a variable in  $V$ .) Thus, the a pair  $(V, C)$  is in a *conjunction normal form* (CNF). As the clauses, variables, and literals are also formulas so the clause is satisfiable if at least one of its literals is satisfied. A clause is unsatisfiable if all its literals are not satisfiable. The question is to decide whether or not there is an assignment on  $V$  that makes each clause in  $C$  satisfiable? If the answer is yes,  $(V, C)$  is satisfiable (SAT), if no, the instance is unsatisfiable (UNSAT). Formally SAT is described as the following problem:

SAT:

INPUT: A finite set  $V$  of Boolean variables and a finite set  $C$  of clauses over  $V$ .

OUTPUT: SAT, if there is a truth assignment to all elements of  $V$  satisfies every clause in  $C$ . UNSAT, otherwise.

By Cook's classic theorem (see, e.g., [66, Theorem 8.2]), SAT is NP-complete, and by the very definition of NP-completeness, every problem in NP reduces to SAT.

### 1.3 Finite automata

A *finite automaton* is a 3-tuple  $\mathcal{A} = (Q, \Sigma, \delta)$ , where

1.  $Q$  is a finite non-empty set called the *state set* which elements are referred to as *states*,
2.  $\Sigma$  is a finite non-empty set called the *input alphabet* which elements are referred to as *input symbols* or *input letters*,
3.  $\delta$  is a map, called the *transition function*; *not necessarily total*, that describes the action of elements of  $\Sigma$  at each state in  $Q$ .

The conventional concept of a finite automaton includes distinguishing two non-empty subsets of  $Q$  consisting of *initial* and *final* states. As these play no role in our considerations, the above simplified definition well suffices for the purpose of the thesis.

Finite automata are usually classified into the following three categories according to the nature of their transition function.



*Definition 1.10.* An automaton  $\mathcal{A} = (Q, \Sigma, \delta)$  is called a *Complete Deterministic Finite Automaton* (DFA) if the transition function  $\delta$  is a total map  $Q \times \Sigma \rightarrow Q$ , that is,  $\delta(q, a)$  is defined for every pair  $(q, a) \in Q \times \Sigma$ . In symbols,  $\forall q \in Q, a \in \Sigma; |\delta(q, a)| = 1$ . We interpret  $\delta(q, a)$  as the next state where the DFA would move to if it was at the state  $q$  and read the symbol  $a$ .

*Definition 1.11.* An automaton  $\mathcal{A} = (Q, \Sigma, \delta)$  is called a *Partial Deterministic Finite Automaton* (PFA) if the transition function  $\delta$  is a partial map  $Q \times \Sigma \rightarrow Q$ , that is,  $\delta(q, a)$  is defined for some pairs  $(q, a) \in Q \times \Sigma$  but may be undefined for some other pairs. In symbols,  $\forall q \in Q, a \in \Sigma; |\delta(q, a)| \leq 1$ . We again interpret  $\delta(q, a)$ , provided it is defined, as the next state where the PFA would move to if it was at the state  $q$  and read the symbol  $a$ , and we write  $\delta(q, a) = \emptyset$  to indicate that  $\delta(q, a)$  is undefined.

*Definition 1.12.* An automaton  $\mathcal{A} = (Q, \Sigma, \delta)$  is called a *Nondeterministic Finite Automaton* (NFA) if the transition function  $\delta$  is a map  $Q \times \Sigma \rightarrow \mathcal{P}(Q)$ , where  $\mathcal{P}(Q)$  is the power set of  $Q$ , that is, for every state  $q \in Q$  and for every symbol  $a \in \Sigma$ , the expression  $\delta(q, a)$  is not a single state, but rather a subset of states. If this subset is non-empty, we interpret it as the set of all possible states where the NFA could move to if it was at the state  $q$  and read the symbol  $a$ . If  $\delta(q, a) = \emptyset$ , we say that the action of  $a$  is undefined at  $q$ .

A finite automaton  $\mathcal{A} = (Q, \Sigma, \delta)$  is conveniently represented as a directed graph  $G_{\mathcal{A}} = (Q, E)$ . Thus, the vertices of  $G_{\mathcal{A}}$  are just the states of  $\mathcal{A}$ . The directed edges of  $E$  are labelled by the elements of  $\Sigma$  defining the transitions between states. The graph  $G_{\mathcal{A}}$  is called *the underlying graph* of the automaton  $\mathcal{A}$ . The graph  $G_{\mathcal{A}}$  has an edge labelled by  $a$

from the vertex  $q$  to the vertex  $q'$  if and only if  $q' \in \delta(q, a)$  in  $\mathcal{A}$ , so we have

$$E = \{q \xrightarrow{a} q' \mid q, q' \in Q, a \in \Sigma, q' \in \delta(q, a)\}.$$

Figure 1.1 illustrates the above three categories of finite automata.

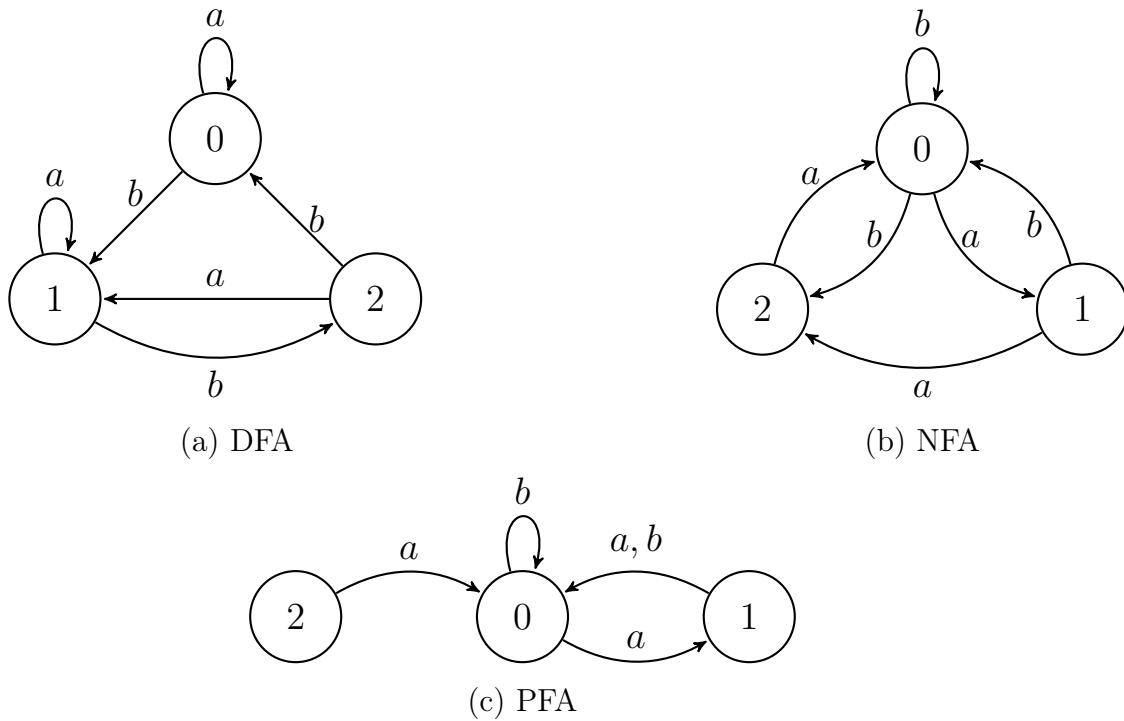


Figure 1.1: Different categories of finite automata

Let  $\Sigma$  be an alphabet, a *word* over  $\Sigma$  is a finite sequence of symbols from  $\Sigma$ . We do not exclude the empty sequence from this definition; that is, we allow the *empty word*  $\varepsilon$ . The set of all words over  $\Sigma$  including  $\varepsilon$  is denoted by  $\Sigma^*$  and is referred to as the *free monoid over  $\Sigma$* . If  $w = a_1 \cdots a_\ell$  with  $a_1, \dots, a_\ell \in \Sigma$  is a non-empty word over  $\Sigma$ , the number  $\ell$  is said to be the *length* of  $w$  and is denoted by  $|w|$ . The length

of the empty word is defined to be 0. The set of all words of a given length  $\ell$  over  $\Sigma$  is denoted by  $\Sigma^\ell$ .

Given an automaton  $\mathcal{A} = (Q, \Sigma, \delta)$ , the transition function extends to a function  $\mathcal{P}(Q) \times \Sigma^* \rightarrow \mathcal{P}(Q)$ , still denoted  $\delta$ , in the following inductive way: for every subset  $R \subseteq Q$  and every word  $w \in \Sigma^*$ , we set

$$\delta(R, w) := \begin{cases} R & \text{if } w = \varepsilon, \\ \bigcup_{q \in \delta(R, v)} \delta(q, a) & \text{if } w = va \text{ with } v \in \Sigma^* \text{ and } a \in \Sigma. \end{cases} \quad (1.1)$$

(Here the set  $\delta(R, v)$  is defined by the induction assumption since  $v$  is shorter than  $w$ .) In the following, we often write  $q.w$  for  $\delta(q, w)$  and  $R.w$  for  $\delta(R, w)$  whenever we deal with a fixed automaton  $\mathcal{A} = (Q, \Sigma, \delta)$ .

If the automaton possesses an input word that takes it from any state to a specific state, this special word is called a *synchronizing word* or a *reset word*. The automaton which has such a word is called *synchronizing automaton*. The formal definitions of a synchronizing word and a synchronizing automaton depend on the class of the investigated automaton. Let us start with DFAs.

## 1.4 Synchronizing DFA

*Definition 1.13.* (Synchronizing DFA) A DFA  $\mathcal{A} = (Q, \Sigma, \delta)$  is *synchronizing* if and only if it has a word  $w \in \Sigma^*$  such that

$$\forall q \in Q, q.w = Q.w \quad (1.2)$$

Each word  $w$  that satisfies the condition (1.2) is called a *synchronizing* or *reset* word for the automaton  $\mathcal{A}$ .

According to Definition 1.13, after the synchronizing word has been applied to the automaton, the current state of the automaton is known. At this point we can say that the automaton is completely under control as it is possible to force it into the desirable mode of operation.

Not all DFAs are synchronizing so the following *SYN-DFA* decision problem is the first problem that arises in this area.

SYN-DFA

INPUT: Given a DFA  $\mathcal{A} = (Q, \Sigma, \delta)$ , is  $\mathcal{A}$  synchronizing?

OUTPUT: Yes, if there is a synchronizing word for it. No, otherwise.

The SYN-DFA can be decided with the help of an auxiliary automaton called *the power automaton*. Given an automaton  $\mathcal{A} = (Q, \Sigma, \delta)$ , the power automaton  $\mathcal{P}(\mathcal{A}) = (S, \Sigma, \delta)$  where  $S$  is the set of all nonempty subsets of  $Q$ ;  $|S| = 2^{|Q|} - 1$ , and  $\delta$  is extended to  $S \times \Sigma$  as in (1.1). SYN-DFA Problem is reduced to the reachability problem in the underlying graph of the automaton  $\mathcal{P}(\mathcal{A})$ . The automaton  $\mathcal{A}$  is synchronizing if and only if the graph  $G_{\mathcal{P}(\mathcal{A})}$  has a path starting at the vertex that represents the set  $Q$  and ending at a singleton vertex. This reduction is shown in Figures 1.2, and 1.3 where Figure 1.3 is the power automaton of the automaton in Figure 1.2. Although this process is conceptually simple, it is inefficient as the size of the power automaton  $\mathcal{P}(\mathcal{A})$  is exponentially larger than that of  $\mathcal{A}$ .

Another technique used to solve SYN-DFA Problem and makes it easier was introduced independently by Černý [15] and Liu [53]. This technique depends on the *pair-wise synchronization* as in the following proposition.

**Proposition 1.1.** *{pair-wise synchronization}* A DFA  $\mathcal{A} = (Q, \Sigma, \delta)$

is synchronizing if and only if any pair of states  $p, q \in Q$  can be synchronized, that is,

$$\forall \{p, q \mid p \neq q\} \subset Q, \exists w \in \Sigma^*, q.w = p.w$$

The pair-wise synchronization condition based on the property that the given automaton is complete. This condition can be easily checked by the construction of another auxiliary automaton called *the pair automaton or the square automaton*.

Given an automaton  $\mathcal{A} = (Q, \Sigma, \delta)$ , its pair automaton is the automaton  $\mathcal{A}^{[2]} = (Q^{[2]}, \Sigma, \delta^{[2]})$ . The set of states  $Q^{[2]}$  is the set of all 2-element and 1-element subsets of  $Q$ ; that is,  $Q^{[2]} := \{\{p, q\} \mid p, q \in Q\}$ , and the transition function  $\delta^{[2]}$  is defined as follows:

$$\delta^{[2]}(\{p, q\}, a) := \{\delta(p, a), \delta(q, a)\}.$$

The automaton  $\mathcal{A}$  has a synchronizing word if and only if all the following  $\binom{|Q|}{2}$  reachability problems have a Yes answer.

Let  $X$  be the set of all 2-element subsets of  $Q$ . For each element  $e$  in  $X$ , is there a path from  $e$  to an element in  $Q$ ?

The pair-wise synchronization condition may be verified by using backtracking; running Breadth First Search (BFS) algorithm in the *inverse automaton*  $i\mathcal{A}^{[2]}$  of  $\mathcal{A}^{[2]}$ . See [17] for a detailed presentation of BFS technique. The inverse automaton  $i\mathcal{N}(Q, \Sigma, \delta^i)$  of the automaton  $\mathcal{N}(Q, \Sigma, \delta)$  is the automaton whose transition function has the form  $\delta^i(q, a) = \{p \mid \delta(p, a) = q\}$ . The automaton is synchronizing if and only if the BFS starting from the set  $Q$  reaches every element of the set  $X$ . For an illustration, look at Figure 1.4, which shows the pair automaton

of the DFA in Figure 1.2.

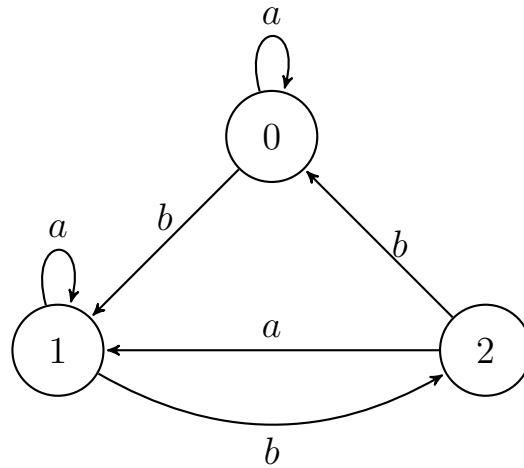


Figure 1.2: A DFA with  $Q := \{0, 1, 2\}$  and  $\Sigma := \{a, b\}$

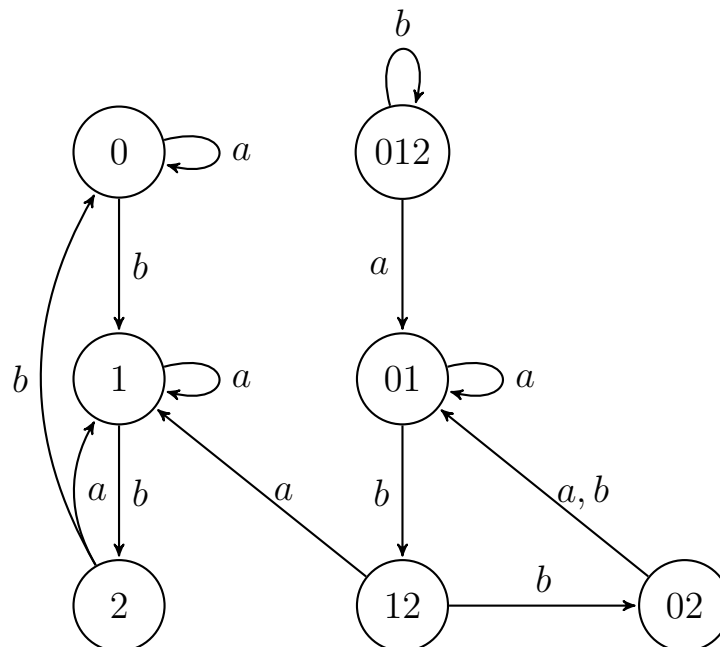


Figure 1.3: Power automaton of the automaton in Figure 1.2

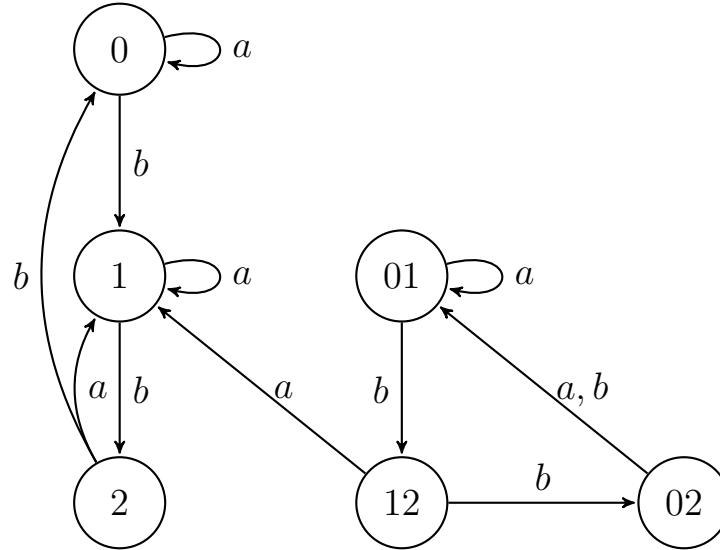


Figure 1.4: Pair automaton of the automaton in Figure 1.2

In the following we mention a significant phenomenon of the strongly connected synchronizing DFAs

*Definition 1.14.* A directed graph  $G = (V, E)$  is called *strongly connected* if

$$\text{for any } s, t \in V, \exists (s \xrightarrow{\text{path}} t) \wedge (t \xrightarrow{\text{path}} s)$$

*Definition 1.15.* A *strongly connected automaton* is an automaton such that its underlying graph is a strongly connected graph.

The following lemma comes from the straightforward application of Definitions 1.14 and 1.15.

**Lemma 1.2.** *Synchronizing strongly connected DFA can be synchronized to any state.*

As soon as verifying that the automaton is synchronizing, the first question is: what is the synchronizing word for it? Since the automaton

is synchronizing, then the pair-wise condition is verified. A synchronizing word for the automaton is constructed by finding the synchronizing word for a one pair then, this word can be appended to the synchronizing word of another pair of states till reaching to the synchronizing word of the last pair. This process is illustrated in Algorithm 1. Since the automaton  $\mathcal{A}^{[2]}$  has  $\frac{|Q|(|Q|+1)}{2}$  states, Algorithm 1, solves this question in  $O(|Q|^2 \cdot |\Sigma|)$  time and uses  $O(|Q|^2)$  working space. This algorithm is considered in [22].

**Input** :  $\mathcal{A} = (Q, \Sigma, \delta)$   
**Output**: a synchronizing word  $w$  for  $\mathcal{A}$   
construct  $\mathcal{A}^{[2]}$ ;  $Q^{[2]} = X \cup Q$   
initialization  
 $P \leftarrow X$   
 $w \leftarrow \varepsilon$   
**while**  $P \neq \emptyset$  **do**  
    | find  $v \in P$  and  $u \in \Sigma^*$  with  $\delta^{[2]}(v, u) \in Q$   
    |  $w \leftarrow wu$   
    |  $P \leftarrow \delta(P, u) \cap X$   
**end**  
**return**  $w$

**Algorithm 1:** Finding a synchronizing word for a DFA.

On each loop, the algorithm appends the word  $u$  of length  $O(|Q|^2)$ , and the number of loops is  $O(|Q|)$ . therefore the synchronizing word  $w$  returned by Algorithm 1 has length  $O(|Q|^3)$ .



## 1.5 NFA Synchronization

In this section we overview different formalizations of synchronization in nondeterministic automata. Section 1.4 introduced the synchronization of DFAs. Synchronization of such automata has a unique formalization as described in Definition 1.13. This is due to the fact that DFAs are fully specified in the sense that for each  $q \in Q, a \in \Sigma$  we have  $|q.a| = 1$ . In contrast, NFAs are not fully specified; the knowledge of the current state and the incoming input is insufficient to uniquely determine the next state. Accordingly, the concept of synchronization for DFAs was extended to NFAs in several non-equivalent ways. The following three nowadays popular versions were suggested and analyzed in [34] in 1999 (although, in an implicit form, some of them appeared in the literature much earlier, see, e.g., [11, 31]). A given NFA may be synchronizing with respect to one version but not for another version.

*Definition 1.16.* Let  $\mathcal{A} = (Q, \Sigma, \delta)$  be an NFA. A word  $w \in \Sigma^*$  is called

1.  $D_1$ -synchronizing word if there is a state  $p \in Q$  such that  $q.w = \{p\}$  for all  $q \in Q$ ;
2.  $D_2$ -synchronizing word if  $q.w = q'.w$  for all  $q, q' \in Q$ ;
3.  $D_3$ -synchronizing word if  $\bigcap_{q \in Q} q.w \neq \emptyset$ .

*Definition 1.17.* An NFA is called  $D_i$ -synchronizing,  $i = 1, 2, 3$ , if it has a  $D_i$ -synchronizing word.

These definitions yield the following properties of  $D_i$ -synchronizing words:

**Lemma 1.3.** 1. A  $D_1$ -synchronizing word or  $D_3$ -synchronizing word is defined at each state.

2. A  $D_2$ -synchronizing word is either defined at each state or undefined at each state.

3. Every  $D_1$ -synchronizing word is both  $D_2$ -synchronizing word and  $D_3$ -synchronizing word; every  $D_2$ -synchronizing word defined at each state is  $D_3$ -synchronizing word.

For brevity, we will use the acronym sw for “synchronizing word” and sws for “synchronizing words”. For an illustration of the three versions for NFAs synchronization, consider the NFA  $\mathcal{A}$  in Figure 1.5. It is easy to see that the word  $abc$  is a  $D_1$ -sw, since  $0.abc = 1.abc = 2.abc$ ; the word  $ab$  is a  $D_2$ -sw since  $0.ab = 1.ab = 2.ab$ , but not  $D_1$ -sw; and the word  $a$  is a  $D_3$ -sw since  $\bigcap_{0 \leq i \leq 2} (i.a) \neq \emptyset$ , but not a  $D_2$ -sw.

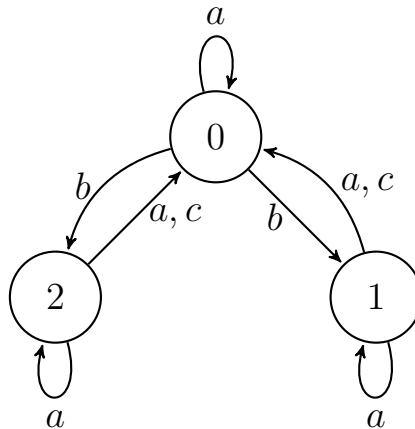


Figure 1.5: A NFA with  $Q = \{0, 1, 2\}$  and  $\Sigma = \{a, b, c\}$

We also mention in passing that  $D_i$ -synchronization gets a very transparent meaning within a standard matrix representation of NFAs. In

this representation, an NFA  $\mathcal{A} = (Q, \Sigma, \delta)$  becomes a collection of  $|\Sigma|$  Boolean  $Q \times Q$ -matrices where to each input symbol  $a \in \Sigma$ , a matrix  $M(a)$  is assigned such that the  $(q, q')$ -entry of  $M(a)$  is 1 if  $q' \in \delta(q, a)$  and 0 otherwise. Then it is not hard to realize that the automaton  $\mathcal{A}$  is  $D_3$ -synchronizing if and only if some product of the matrices  $M(a)$ ,  $a \in \Sigma$ , has a column consisting entirely of 1s.  $\mathcal{A}$  is  $D_2$ -synchronizing if in some product of the matrices  $M(a)$ ,  $a \in \Sigma$ , each column consists either entirely of 1s or entirely of 0s.  $\mathcal{A}$  is  $D_1$ -synchronizing if and only if some product of the matrices  $M(a)$ ,  $a \in \Sigma$ , has an exactly one column consisting entirely of 1s and all other columns with all entries of 0s.

Some information about  $D_i$ -synchronization,  $i = 1, 2, 3$ , can be found in Chapter 8 of Ito's monograph [35]. Recently, some aspects of  $D_3$ -synchronization have been considered in [7, 12, 19, 28, 80]. (The papers [7, 28] use the language of matrices rather than that of automata.) According to Definitions 1.10, 1.12, 1.13, 1.16 and 1.17, the following lemma is easy to verify

**Lemma 1.4.** *For a DFA, the notions of a  $D_1$ -synchronizing word, a  $D_2$ -synchronizing word and a  $D_3$ -synchronizing word coincide with each other and with the notion of a synchronizing word as defined in Section 1.4*

$D_3$ -synchronization is the most general version of synchronization for NFAs amongst those considered in the literature so far. Besides that, it reasonably reflects the basic nature of non-determinism. Indeed, if an NFA  $\mathcal{A} = (Q, \Sigma, \delta)$  is used as an *acceptor*, we designate some states in  $Q$  as initial and final and then say that  $\mathcal{A}$  *accepts* a word  $w \in \Sigma^*$  whenever there exists a path labeled  $w$  that starts at an initial state and terminates at a final state. The definition of a  $D_3$ -synchronizing word

very much resembles this concept: a word  $w \in \Sigma^*$  is  $D_3$ -synchronizing whenever for each  $q \in Q$ , there exists a path labeled  $w$  that starts at  $q$  and terminates at a certain common state, independent of  $q$ . In both cases we do not require that a starting state uniquely determines the path labeled  $w$  nor that every path labeled  $w$  with a given starting state should arrive at a final/common state.

To understand the ‘physical meaning’ of  $D_2$ , imagine a big quantity of identical NFAs which get the same input sequence and work on it in parallel. If the sequence constitutes a  $D_2$ -synchronizing word, then after consuming the input, the NFAs will demonstrate identical (that is, synchronous) behaviour, even though originally they all might have been in different states that were unknown to us.

Another situation that the  $D_2$  synchronization appears is in the game theory, since it shrinks the uncertainty on the current state of the game. Consider a player is in a set of possible states  $S_1$ . If we have a  $D_2$ -synchronizing word that sends all states to a set of states  $S_2$  with  $|S_2| < |S_1|$ , then the player has gained information since the number of possible states given what he knows has decreased.

The equality  $q.w = q'.w$  for all  $q, q' \in Q$  does not imply that the action of  $w$  must be everywhere defined. Hence the word that is undefined everywhere is also a  $D_2$ -synchronizing word. This special word is called *a mortal word* for the given NFA.

In the following we show the relation between various versions of synchronization for NFAs and *Complete Nondeterministic Automata* (CNFAs). Any CNFA is an NFA such that the current set of possible states is always nonempty:

$$\forall q \in Q, a \in \Sigma, q.a \neq \emptyset.$$

*Definition 1.18.* Let  $\mathcal{A} = (Q, \Sigma, \delta)$  be an NFA with  $|Q| = n > 1$ . For  $i \in \{1, 2, 3\}$ ,

1.  $D_i$  is the class of all  $D_i$ -synchronizing automata,
2.  $CD_i$  is the class of all complete  $D_i$ -synchronizing automata,
3.  $D_i(\mathcal{A})$  is the set of  $D_i$ -synchronizing words for  $\mathcal{A}$ ,
4.  $d_i(n) = \max\{D_i(\mathcal{A}) \mid \mathcal{A} \in D_i\}$ ,
5.  $cd_i(n) = \max\{D_i(\mathcal{A}) \mid \mathcal{A} \in CD_i\}$ .

The next lemmas introduce the relation between the three versions of synchronization for NFA and CNFA

**Lemma 1.5.** *For any CNFA  $\mathcal{A}$ ,*

$$D_1(\mathcal{A}) \subseteq D_2(\mathcal{A}) \subseteq D_3(\mathcal{A}).$$

**Lemma 1.6.** *For any NFA  $\mathcal{A}$ ,*

$$D_1(\mathcal{A}) \subseteq D_2(\mathcal{A}) \cap D_3(\mathcal{A}).$$

## 1.6 Complexity of synchronization in finite automata

Checking synchronization of a DFA is fast and easy. In contrast checking synchronization for a given NFA or a PFA is hard and takes more time.

The following result was found in [75, 76] and later rediscovered and strengthened in [58].

**Theorem 1.1.** [58, 75, 76] *The problems of checking whether or not a given NFA is  $D_1$ -,  $D_2$ - or  $D_3$ -synchronizing are PSPACE-complete.*

The PSPACE-completeness result even holds for NFAs with only one ambiguous transition or only one undefined transition and, even more surprisingly, also for PFAs [58, 60].

## 1.7 Shortest synchronizing word

For evident reasons of economy, it is useful to have synchronizing words as short as possible. This motivated a large number of papers devoted to the study of bounds for the length of the shortest synchronizing word of synchronizing automaton. In the following two sections we survey the basic known results of how these bounds depend on the number of states for DFAs and NFAs.

### 1.7.1 Shortest synchronizing word for DFAs

Černý [15] constructed a series of  $n$ -states binary DFAs (binary means that  $|\Sigma| = 2$ ). Each automaton in this series is known as the Černý automaton  $\mathcal{C}_n$ , where  $n > 1$ . The definition of the automaton  $\mathcal{C}_n$  is clear from Figure 1.6. Černý proved that the word  $w = (ab^{n-1})^{n-2}a$  with  $|w| = (n - 1)^2$  is the shortest synchronizing word for  $\mathcal{C}_n$ . This result made him propose the most famous conjecture in the synchronization theory. It is known in literature as the Černý conjecture.

*Conjecture 1.1.* [15] Any synchronizing DFA with  $n$  states has a synchronizing word of length at most  $(n - 1)^2$ .

The conjecture resists researchers' efforts for more than 50 years. The best upper bound achieved so far is cubic in  $n$ ; it is due to Shitov [81] who has slightly improved the bound established by Szykuła [82]. In turn, Szykuła's bound is only slightly better than the upper bound  $\frac{n^3-n}{6}$  established by Pin [67] and Frankl [23] approx. 38 years ago.

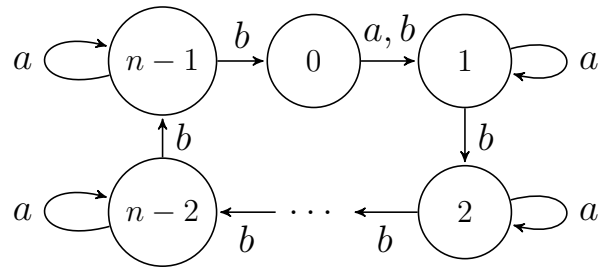


Figure 1.6: Černý automaton  $\mathcal{C}_n$

Another problem that is related to the length of the synchronizing word is the  $\ell$ -Syn-Word problem:

$\ell$ -Syn-Word: Given a synchronizing automaton  $\mathcal{A}$  and a positive integer  $\ell$ , is it true that  $\mathcal{A}$  has a synchronizing word of length  $\ell$ ?

Unfortunately, this problem is computationally hard. Eppstein [22] proved that the  $\ell$ -Syn-Word problem is *NP-complete* by a polynomial reduction from 3-SAT. In [56, 78], it was proved that the problem of deciding whether the length of the shortest synchronizing word is equal to  $\ell$  is both NP-hard and coNP-hard. So we can say that under the standard assumptions of complexity theory even non-deterministic algorithm can not find the length of the shortest synchronizing word in polynomial time.

The exact complexity of such problem has been determined by Gawrychowski [24] and, independently, by Olschewski and Ummels [65]. It

turns out that the appropriate complexity class is **DP** (Difference Polynomial Time) introduced by Papadimitriou [66]; **DP** consists of languages of the form  $L_1 \cap L_2$  where  $L_1$  is a language from **NP** and  $L_2$  is a language in **coNP**. A ‘standard’ **DP**-complete problem is SAT-UNSAT whose instance is a pair of clause systems  $\psi, \chi$ , say, and whose question is whether  $\psi$  is satisfiable and  $\chi$  is unsatisfiable.

### 1.7.2 Shortest synchronizing word for NFA

The problem of finding the shortest  $D_i$ -synchronizing word for a  $D_i$ -synchronizing NFA or even the  $\ell$ -syn word that was described for DFAs is computationally harder than that for a DFA. In DFAs the length of the shortest synchronizing word is bounded by polynomial function. On the contrary, there is no polynomial in  $n$  upper bound on the length of synchronizing words for synchronizing NFA with  $n$  states. Burkhard in [11] introduced  $D_1$ -synchronization of CNFAs, and proved that:

$$n \geq 1, cd_1(n) = 2^n - n - 1.$$

In [25, 37, 57], it was proved that

$$2^n - n \leq d_1(n) \leq \Theta(2^n)$$

$$2^n - n - 1 \leq d_2(n) \leq \Theta(2^n)$$

$$\Omega(3^{n/3}) \leq d_3(n) \leq \Theta(n^2 4^{n/3}).$$

See Section 1.5 for the notions of  $d_1(n)$ ,  $d_2(n)$ , and  $d_3(n)$ .



## 1.8 Careful synchronization

Another version of synchronization for NFAs was introduced in [36, 75] and systematically studied in [55, 57–60].<sup>1</sup> An NFA  $\mathcal{A} = (Q, \Sigma, \delta)$  is called *carefully synchronizing* if there is a word  $w = a_1 \cdots a_\ell$ , with  $a_1, \dots, a_\ell \in \Sigma$ , that satisfies the condition (C), being the conjunction of (C1)–(C3) below:

(C1): the letter  $a_1$  is defined at every state in  $Q$ ;

(C2): the letter  $a_t$  with  $1 < t \leq \ell$  is defined at each state in  $Q.a_1 \cdots a_{t-1}$ ;

(C3):  $|Q.w| = 1$ .

Any  $w$  satisfying (C) is called a *carefully synchronizing word* for  $\mathcal{A}$ . Thus, when a carefully synchronizing word is applied at any state in  $Q$ , no undefined transition occurs during the course of application. In the literature, Careful synchronization is studied for PFAs. Practically careful synchronization of PFAs is relevant in numerous applications. We mention two of these applications:

Carefully synchronizing automata may be used in industrial robotics as DFAs synchronizing automata. See page 7 of the introduction for a detailed discussion and an illustrative example for the application of DFAs in such industry. Now imagine that the objects to be oriented or sorted have a fragile side that could be damaged if hitting an obstacle. In order to prevent any damage, we have to forbid ‘dangerous’ transitions in the automaton modelling the orienter/sorter so that the automaton becomes partial and the obstacle sequences must correspond to carefully

---

<sup>1</sup>It should be mentioned that Rystsov [75] used the term “synchronizing word” for what we call “carefully synchronizing word”, following Martyugin [55, 57–60]

synchronizing words. (Actually, the term ‘careful synchronization’ has been selected with this application in mind.) For an illustration, we consider the following example from Martyugin’s PhD thesis [54]. Suppose the part to be oriented has the shape shown in Figure 1.7. The part has four faces, over two of them there are protrusions, and the corner between them is fragile. Assume that the part hit the conveyor randomly and should be oriented correctly. Suppose that there are four possible orientations of the part shown in Figure 1.8, and for assembly, the part must be oriented with the tabs to the left and up (position 2 from the left in the Figure 1.8). To design the orienter, we will use a conveyor that contains a certain number of obstacles that the parts will have to overcome while moving along the conveyor. We will use two types of obstacles: tall (T) and short(S). If the part, overcoming a short obstacle lies on the face without a protrusion (positions 2 and 3 in the Figure 1.8), it does not turn over, and if it lies on the face with a protrusion (positions 1 and 4 in Figure 1.8), it turns 90 degrees clockwise. Overcoming a tall obstacle, the part is always turned clockwise by 90 degrees. However, if the part lies with the fragile side forward (position 4 in Figure 1.8), then the tall obstacle will hit the part, touch its fragile side and break it. This situation is not acceptable, so a tall obstacle should not meet the part in position 4. Thus, we can construct the orienter as an PFA automaton  $\mathcal{A}$  whose state set consists of the different orientations, whose input alphabet is  $\{T, S\}$ , and whose transition function  $\delta$  is defined by the action of each obstacle on different orientations of the part. This automaton is described in Figure 1.9 and It is easy to verify that this automaton is carefully synchronizing and the sequence S–S–T–S–T–S–S of obstacles is a carefully synchronizing word.

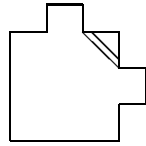


Figure 1.7: Incoming part

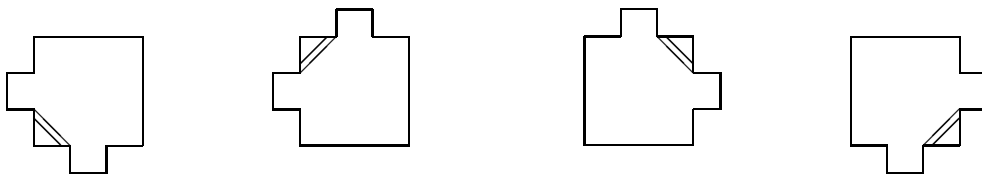


Figure 1.8: Possible orientations of the incoming part

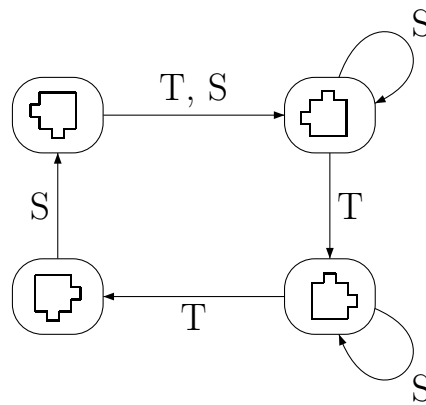


Figure 1.9: Carefully synchronizing orienter

Second example relates to so-called synchronized codes. See page 10 of the introduction for a detailed discussion for the application of synchronizing automata in coding theory. Recall that a *prefix code* over a finite alphabet  $\Sigma$  is a set  $X$  of words in  $\Sigma^*$  such that no word of  $X$  is

a prefix of another word of  $X$ . Decoding of a finite prefix code  $X$  over  $\Sigma$  can be implemented by a finite deterministic automaton  $\mathcal{A}_X$  whose state  $Q$  is the set of all proper prefixes of the words in  $X$  (including the empty word  $\varepsilon$ ) and whose transitions are defined as follows: for  $q \in Q$  and  $a \in \Sigma$ ,

$$q.a := \begin{cases} qa & \text{if } qa \text{ is a proper prefix of a word of } X, \\ \varepsilon & \text{if } qa \in X, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

In general,  $\mathcal{A}_X$  is a PFA (it is complete if and only if the code  $X$  is not contained in another prefix code over  $\Sigma$ ). It can be shown that if  $\mathcal{A}_X$  is carefully synchronizing, the code  $X$  has a very useful property: whenever a loss of synchronization between the decoder and the coder occurs (because of a channel error), it suffices to transmit a carefully synchronizing word  $w$  of  $\mathcal{A}_X$  such that  $w$  sends all states in  $Q$  to the state  $\varepsilon$  to ensure that the following symbols will be decoded correctly. The synchronizing code  $\{011, 001, 10, 110, 111\}$  and its automaton are illustrated in Figures 1.10 and 1.11. The word 11011110" is a carefully synchronizing word.

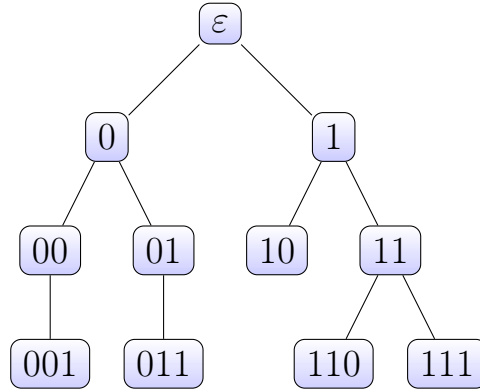


Figure 1.10: The tree of the code  $\{011, 001, 10, 110, 111\}$

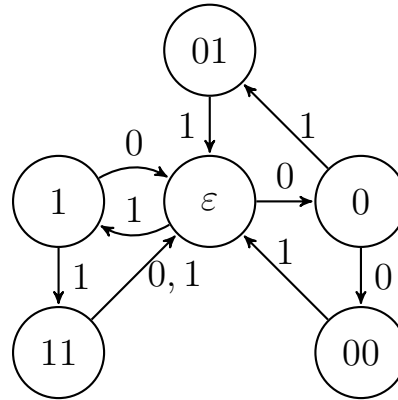


Figure 1.11: The automaton of the code in Figure 1.10

On the theoretical side, every carefully synchronizing word is clearly  $D_1$ -synchronizing but the converse is not true in general; moreover, a  $D_1$ -synchronizing NFA may admit no carefully synchronizing word. Thus, if we denote by  $D_i$ ,  $i = 1, 3$ , the class of all  $D_i$ -synchronizing NFAs, by  $D_2^*$  the class of all  $D_2$ -synchronizing NFAs with  $D_2$ -synchronizing word defined at each state, and by  $C$  the class of all carefully synchronizing NFAs, we have the following strict inclusions:

$$C \subset D_1 \subset D_2^* \subset D_3.$$

It is easy to see that each of the conditions  $(C)$ ,  $D_1$ ,  $D_3$  leads to the same notion when restricted to PFAs. As for  $D_2$ -synchronization, if a word  $w$  is  $D_2$ -synchronizing for a PFA  $\mathcal{A}$ , then  $w$  carefully synchronizes  $\mathcal{A}$  whenever  $w$  is defined at each state. Otherwise  $w$  is nowhere defined by Lemma 1.3b, and such mortal words are nothing but usual synchronizing words for the DFA obtained from  $\mathcal{A}$  by adding a new sink state and making all transitions undefined in  $\mathcal{A}$  lead to this sink state. Synchronization of DFAs with a sink state is relatively well understood (see [74]), and therefore, we may conclude that  $D_2$ -synchronization also reduces to careful synchronization in the realm of PFAs. On the other hand, there exists a simple transformation that converts every NFA  $\mathcal{A} = (Q, \Sigma)$  into a PFA  $\mathcal{B} = (Q, \Sigma')$  such that  $\mathcal{A}$  is  $D_3$ -synchronizing if and only if so is  $\mathcal{B}$  and the minimum lengths of  $D_3$ -synchronizing words for  $\mathcal{A}$  and  $\mathcal{B}$  are equal; see [35, Lemma 8.3.8] and [37, Lemma 3]. These observations demonstrate that from the viewpoint of optimal synchronization, studying carefully synchronizing words for PFAs may provide both lower and upper bounds applicable to arbitrary NFAs and all aforementioned kinds of synchronization.

## 1.9 Exact synchronization

Let  $\mathcal{A} = (Q, \Sigma, \delta)$  be an NFA. A word  $w \in \Sigma^*$  is called *exactly synchronizing* for  $\mathcal{A}$  if there exists a state  $p \in Q$  such that for each state  $q \in Q$ , either  $w$  maps  $q$  to  $p$  or  $w$  is undefined at  $q$ , and there is at least one state at which  $w$  is defined. That is, a word is called exactly

synchronizing if it maps the whole set of states of the automaton to a set of size exactly one. Thus, a word  $w$  is exactly synchronizing for the automaton  $\mathcal{A}$  if it satisfies the condition (C3) from conditions described in the definition of careful synchronization:  $|Q.w| = 1$ . Clearly, a carefully synchronizing word is exactly synchronizing but the converse needs not be true. A NFA is said to be *exactly synchronizing* if it possesses an exactly synchronizing word.

Similarly to careful synchronization, exact synchronization has interesting connections and numerous applications. In particular, exact synchronization is relevant in biologically inspired computing where exactly synchronizing words appear under the name ‘constants’ in the study of so-called splicing systems, see [10]. Another cause of interest in exact synchronization is provided by so-called  $\epsilon$ -machines, important models in the theory of stationary information sources, see [84, 85].

## 1.10 Checking careful synchronization

It was proved by Martyugin in [58, 60] that the problem of checking the careful synchronization for a given PFAs is PSPACE-complete. This problem remain PSPACE-complete also for strongly connected PFAs. This result was mentioned in [87] without proof. To the sake of completeness we provide a proof here.

**Theorem 1.2.** *Checking careful synchronization for a given strongly connected PFA is PSPACE-complete.*

*Proof.* Given a strongly connected DFA  $\mathcal{A} = (Q, \{a, b\}, \delta)$ . Take  $S := \{q_1, \dots, q_k\} \subseteq Q$ . Construct a PFA  $\mathcal{C} = (P, \{a, b, c, d\}, \gamma)$  such that

$P := Q \cup \{Y, Z\}$  and  $\gamma$  is defined as follows.

For any  $e \in \{a, b\}$ ,

$$\gamma(q, e) := \begin{cases} \delta(q, e) & \text{if } q \in Q \\ Z & \text{if } q = Z \\ \text{undefined} & \text{if } q = Y \end{cases}$$

$$\gamma(q, c) := \begin{cases} q_1 & \text{if } q \notin S \\ q_k & \text{if } q = Y \\ q & \text{otherwise.} \end{cases}$$

$$\gamma(q, d) := \begin{cases} Y & \text{if } q \in \{Y, Z\} \\ Z & \text{if } q = q_k \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The graphical representation of the automaton  $\mathcal{C}$  is shown in Figure 1.12.

The automaton  $\mathcal{C}$  is strongly connected PFA. Moreover,  $\mathcal{C}$  is carefully synchronizing if and only if the set  $S$  is synchronizing set in  $\mathcal{A}$ . Let  $w \in \{a, b\}^*$  be a synchronizing word for the set  $S$ . Hence the word  $w' = cwdd$  is a carefully synchronizing word for  $\mathcal{C}$ . It is defined at all states and sends all of them to  $Y$ . If  $\mathcal{C}$  is carefully synchronizing, any carefully synchronizing word for  $\mathcal{C}$  must start with the letter  $c$  as it the only one that is defined at all states in  $\mathcal{C}$ . The application of the letter  $c$  results in  $S \cup Z$ . The letter  $c$  can not be applied at this stage as it works as an identity function on  $S \cup Z$ . The letter  $d$  is not defined at all



elements in  $S$ . Thus we have to apply only a sequence of the letters  $a$  and  $b$ . Let  $w \in \{a, b\}^*$  be that sequence. The word  $w$  must send  $S$  to  $q_k$  ( the only state from  $S$  at which  $d$  is defined ). This proves that the word  $w \in \{a, b\}^*$  is a synchronizing word for  $S$ .  $\square$

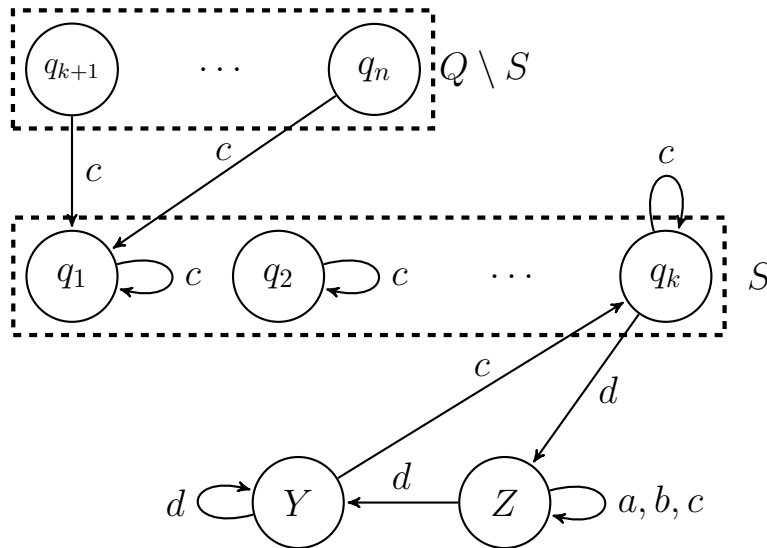


Figure 1.12: The automaton  $\mathcal{C}$

## 1.11 Testing exact synchronization

For strongly connected PFAs, exact synchronization behaves similarly to synchronization of DFAs. In particular, checking whether a given strongly connected PFA is exactly synchronizing can be done in polynomial time, and for strongly connected exactly synchronizing PFAs with  $n$  states, there exists a cubic in  $n$  upper bound on the minimum length of exactly synchronizing words. Both these facts are straightforward consequences of the following observation by Travers and Crutchfield [84]:

**Proposition 1.2.** [84] *A strongly connected PFA  $\mathcal{A} = (Q, \Sigma, \delta)$  is exactly synchronizing if and only if for every pair  $(q, q') \in Q \times Q$  there exists a word  $w$  such that either  $q.w = q'.w$  or  $w$  is defined at one of the states  $q$  and  $q'$  and undefined at the other.*

Unfortunately, sufficiency in Proposition 1.2 does not extend to general PFAs. In the absence of strong connectivity, properties of exact synchronization became rather similar to those of careful synchronization. The following facts were established by Berlinkov [4, Corollary 1]:

**Proposition 1.3.** [4] *Testing a given PFA with 2 input letters for exact synchronization is PSPACE-complete. There is a series of  $n$ -state PFAs with 2 input letters with the minimum length of exactly synchronizing words of magnitude  $2^{\Omega(n)}$ .*

In the following results we complement those in [84].

**Theorem 1.3.** *Given a strongly connected PFA  $\mathcal{B}$  and a positive integer  $\ell$ , checking whether or not  $\mathcal{B}$  has an exactly synchronizing word of length  $\ell$  is NP-complete.*

*Proof.* Given a strongly connected DFA  $\mathcal{A} = (Q, \{a, b\}, \delta)$  such that  $Q := \{q_0, \dots, q_{n-1}\}$ . We construct a PFA  $\mathcal{B} = (P, \{a, b, c, d\}, \gamma)$  for which  $P := Q \cup \{p_0, \dots, p_{n-1}\}$  and  $\gamma$  is defined as follows.

$$\text{For any } e \in \{a, b\}, \gamma(q, e) := \begin{cases} \delta(q, e) & \text{if } q \in Q; \\ p_{(i+1) \bmod n} & \text{if } q = p_i. \end{cases}$$

$$\gamma(q, c) := \begin{cases} q_i & \text{if } q = p_i; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

$$\gamma(q, d) := \begin{cases} p_{n-1} & \text{if } q = q_{n-1}; \\ p_i & \text{if } q = p_i; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The automaton  $\mathcal{B}$  is strongly connected. If the automaton  $\mathcal{A}$  is synchronizing then  $\mathcal{B}$  is exactly synchronizing. Let  $w \in \{a, b\}^*$  be a synchronizing word for  $\mathcal{A}$ . The word  $v = cw$  is an exactly synchronizing word for  $\mathcal{B}$ . Any exactly synchronizing word  $v$  for  $\mathcal{B}$  must contain the factor  $w$  that is a synchronizing word for  $\mathcal{A}$ . If the length of  $w$  is  $\ell$  then the length of  $v$  is at least  $\ell + 1$ . The problem of determining if a DFA has a synchronizing word of length  $\ell$  is known as NP-complete then the same problem for the PFA is NP-complete. The length of  $v$  is still polynomial and bounded by  $O(n^3)$   $\square$

An automaton  $\mathcal{A} = (Q, \Sigma)$  is said to be *0-automaton* if it has a 0-state; a state  $q \in Q$  such that  $q.a = q$  for all  $a \in \Sigma$ .

**Theorem 1.4.** *Checking exact synchronization for a given 0-PFA can be done in polynomial time.*

*Proof.* Given a 0-PFA  $\mathcal{A} = (Q, \Sigma, \delta)$  where  $Y \in Q$  is a 0-state. We can convert it to a DFA  $\mathcal{B} = (Q, \Sigma, \gamma)$  as follows.

For any  $a \in \Sigma$ ,

$$\gamma(q, a) := \begin{cases} \delta(q, a) & \text{if } |\delta(q, a)| > 0; \\ Y & \text{if } |\delta(q, a)| = 0; \\ Y & \text{if } q = Y. \end{cases}$$

The automaton  $\mathcal{A}$  is exactly synchronizing if and only if the automaton  $\mathcal{B}$  is synchronizing. Moreover, the synchronizing word for  $\mathcal{A}$  is an exactly synchronizing word for  $\mathcal{B}$ . Hence the theorem is proved.  $\square$

The proof of the next result is an easy adaptation of the well-known reduction by Eppstein [22] and is therefore omitted.

**Theorem 1.5.** *Checking whether or not a given 0-PFA has an exactly synchronizing word of a given length  $\ell$  is NP-complete.*

**Corollary 1.1.** *Checking whether or not a given 0-NFA has an exactly synchronizing word is PSPACE-complete.*

*Proof.* Starting with 0-NFA, we construct 0-CNFA as illustrated in Theorem 1.4. However, here the original NFA is exact synchronizing if and only if the new generated 0-CNFA has a  $D_1$ -synchronizing word which is known as PSPACE-complete.  $\square$

**Theorem 1.6.** *A strongly connected PFA  $\mathcal{A} = (Q, \Sigma, \delta)$  with exactly one undefined transition is exactly synchronizing by a word of length at most  $n(n-1)/2$  where  $n = |Q|$ .*

*Proof.* Let  $q \in Q$  and  $a \in \Sigma$  be such that  $q.a$  is undefined. Then  $|Q.a| < n$ . If  $|Q.a| = 1$ , we are done as the letter  $a$  becomes an exactly synchronizing word of length 1. If  $|Q.a| = k > 1$ , we take a shortest word  $w_1$  such that  $Q.aw_1$  contains  $q$ . Such a word  $w_1$  exists because  $A$  is strongly connected, and it is easy to see that the length of  $w_1$  is at most  $n - k$ . Now we apply the letter  $a$  again. Then  $|Q.aw_1a| < k$ . If  $|Q.aw_1a| = 1$ , we are done as  $aw_1a$  becomes an exactly synchronizing word of length at most  $1 + (n - k) + 1$ ; if  $|Q.aw_1a| = m > 1$ , we take a shortest word  $w_2$  such that  $Q.aw_1aw_2$  contains  $q$ ; the length of  $w_2$  is at most  $n - m$ . Repeating the process, we eventually get an exactly synchronizing word of the form  $aw_1aw_2 \dots aw_s a$  where  $s < n - 1$  and one can calculate that the length of this word does not exceed  $1 + 1 +$

$$\begin{aligned} 1 + 2 + 1 + 3 + \dots + 1 + (n - 2) + 1 &= (n - 2)(n - 1)/2 + (n - 1) \\ &= (n - 1)[(n - 2)/2 + 1] = n(n - 1)/2. \quad \square \end{aligned}$$

The bound established in Theorem 1.6 is tight. The corresponding series of strongly connected PFAs with exactly one undefined transition can be obtained by removing 0 in Rystsov's series of synchronizing automata with 0 from [74].

From the above discussion of different versions related to NFAs synchronization. We may conclude that the problems of determining whether or not a given NFA or even PFA is synchronizing (with respect to any version of synchronization) and of finding its shortest synchronizing word turn out to be quite difficult. It is known that the first problem is PSPACE-complete and that the minimum length of synchronizing word for synchronizing NFA can be exponential as a function of the number of states. Thus, one has to use some tools that have proved to be efficient for dealing with computationally hard problems. In the following chapters, we employ a satisfiability solver as such a tool.

## Chapter 2

# Synchronization of PFAs

In this chapter we consider two problems of PFAs synchronization. We call them as CSW and ESW for carefully and exactly synchronizing words respectively.

### 2.1 Carefully synchronizing words

CSW: the existence of a carefully synchronizing word of a given length

INPUT: a PFA  $\mathcal{A}$  and a positive integer  $\ell$  (given in unary);

OUTPUT: YES if  $\mathcal{A}$  has a carefully synchronizing word of length  $\ell$ ; NO otherwise.

*Remark 2.1.* We have to assume that the integer  $\ell$  is given in unary because with  $\ell$  given in binary, a polynomial time reduction from CSW to SAT is hardly possible. Indeed, it easily follows from [58] that the version of CSW in which the integer parameter is given in binary is PSPACE-hard, and the existence of a polynomial reduction from a PSPACE-hard problem to SAT would imply that the polynomial hierarchy collapses at level 1. In contrast, the version of CSW with the unary integer parameter is easily seen to belong to NP: given an instance  $(\mathcal{A} = (Q, \Sigma, \delta), \ell)$  of CSW in this setting, guessing a word  $w \in \Sigma^*$  of length  $\ell$  is legitimate. Then one just checks whether or not  $w$  is carefully synchronizing for  $\mathcal{A}$ , and time spent for this check is clearly polynomial in the size of  $(\mathcal{A}, \ell)$ .

Given an arbitrary instance  $(\mathcal{A}, \ell)$  of CSW, we construct an instance  $(V, C)$  of SAT such that the answer to  $(\mathcal{A}, \ell)$  is YES if and only if so is the answer to  $(V, C)$ . As we deal with a fixed automaton  $\mathcal{A}$  we may simplify the notation for  $\mathcal{A}$ , writing  $\mathcal{A} = (Q, \Sigma)$  rather than  $\mathcal{A} = (Q, \Sigma, \delta)$ . In the following presentation of our encoding, precise definitions and statements are interwoven with less formal comments explaining the ‘physical’ meaning of variables and clauses.

So, take a PFA  $\mathcal{A} = (Q, \Sigma)$  and an integer  $\ell > 0$ . Denote the sizes of  $Q$  and  $\Sigma$  by  $n$  and  $m$  respectively, and fix some numbering of these sets so that  $Q = \{q_1, \dots, q_n\}$  and  $\Sigma = \{a_1, \dots, a_m\}$ .

We start with introducing the variables used in the instance  $(V, C)$  of SAT that encodes  $(\mathcal{A}, \ell)$ .

The set  $V$  consists of two sorts of variables:  $m\ell$  *letter variables*  $x_{i,t}$  with  $1 \leq i \leq m$ ,  $1 \leq t \leq \ell$ , and  $n(\ell + 1)$  *state variables*  $y_{j,t}$  with  $1 \leq j \leq n$ ,  $0 \leq t \leq \ell$ . We use the letter variables to encode the letters of a hypothetical carefully synchronizing word  $w$  of length  $\ell$ : namely, we

want the value of the variable  $x_{i,t}$  to be 1 if and only if the  $t$ -th letter of  $w$  is  $a_i$ . The intended meaning of the state variables is as follows: we want the value of the variable  $y_{j,t}$  to be 1 whenever the state  $q_j$  belongs to the image of  $Q$  under the action of the prefix of  $w$  of length  $t$ , in which situation we say that  $q_j$  is *active after  $t$  steps*. We see that the total number of variables in  $V$  is  $m\ell + n(\ell + 1) = (m + n)\ell + n$ .

Now we turn to constructing the set of clauses  $C$ . It consists of four groups. The group  $I$  of *initial clauses* contains  $n$  one-literal clauses as in (2.1) and expresses the fact that all states are active after 0 steps.

$$y_{j,0}, \quad 1 \leq j \leq n. \quad (2.1)$$

For each  $t = 1, \dots, \ell$ , the group  $L$  of *letter clauses* includes the clauses

$$x_{1,t} \vee \dots \vee x_{m,t}, \quad \neg x_{r,t} \vee \neg x_{s,t}, \quad \text{where } 1 \leq r < s \leq m. \quad (2.2)$$

Clearly, the clauses (2.2) express the fact that the  $t$ -th position of our hypothetical carefully synchronizing word  $w$  is occupied by exactly one letter in  $\Sigma$ . Altogether,  $L$  contains  $\ell \left( \frac{m(m-1)}{2} + 1 \right)$  clauses.

For each  $t = 1, \dots, \ell$  and each triple  $(q_j, a_i, q_k)$  in the transition relation of  $\mathcal{A}$ , the group  $T$  of *transition clauses* includes the clause

$$\neg y_{j,t-1} \vee \neg x_{i,t} \vee y_{k,t}. \quad (2.3)$$

Invoking the basic laws of propositional logic, one sees that the clause (2.3) is equivalent to the implication  $y_{j,t-1} \wedge x_{i,t} \rightarrow y_{k,t}$ , that is, (2.3) expresses the fact that if the state  $q_j$  has been active after  $t - 1$  steps and  $a_i$  is the  $t$ -th letter of  $w$ , then the state  $q_k = q_j.a_i$  becomes active



after  $t$  steps. Further, for each  $t = 1, \dots, \ell$  and each pair  $(q_j, a_i)$  such that  $a_i$  is undefined at  $q_j$  in  $\mathcal{A}$ , we add to  $T$  the clause

$$\neg y_{j,t-1} \vee \neg x_{i,t}. \quad (2.4)$$

The clause is equivalent to the implication  $y_{j,t-1} \rightarrow \neg x_{i,t}$ , and thus, it expresses the requirement that the letter  $a_i$  should not occur in the  $t$ -th position of  $w$  if  $q_j$  has been active after  $t - 1$  steps. Obviously, this corresponds to the conditions (C1) (for  $t = 0$ ) and (C2) (for  $t > 0$ ) in the definition of careful synchronization. For each  $t = 1, \dots, \ell$  and each pair  $(q_j, a_i) \in Q \times \Sigma$ , exactly one of the clauses (2.3) or (2.4) occurs in  $T$ , whence  $T$  consists of  $\ell mn$  clauses.

The final group  $S$  of *synchronization clauses* includes the clauses

$$\neg y_{r,\ell} \vee \neg y_{s,\ell}, \quad \text{where } 1 \leq r < s \leq n. \quad (2.5)$$

The clauses (2.5) express the requirement that at most one state remains active when the action of the word  $w$  is completed, which corresponds to the condition (C3) from the definition of careful synchronization. The group  $S$  contains  $\frac{n(n-1)}{2}$  clauses.

Summing up, the number of clauses in  $C := I \cup L \cup T \cup S$  is

$$n + \ell \left( \frac{m(m-1)}{2} + 1 \right) + \ell mn + \frac{n(n-1)}{2} = \ell \left( \frac{m(m-1)}{2} + mn + 1 \right) + \frac{n(n+1)}{2}.$$

**Theorem 2.1.** *A PFA  $\mathcal{A}$  has a carefully synchronizing word of length  $\ell$  if and only if the instance  $(V, C)$  of SAT constructed above is satisfiable. Moreover, the carefully synchronizing words of length  $\ell$  for  $\mathcal{A}$  are in a 1-1 correspondence with the restrictions of satisfying assignments of  $(V, C)$  to the letter variables.*

*Proof.* Suppose that  $\mathcal{A}$  has a carefully synchronizing word of length  $\ell$ . We fix such a word  $w$  and denote by  $w_t$  its prefix of length  $t = 1, \dots, \ell$ . Define a truth assignment  $\varphi: V \rightarrow \{0, 1\}$  as follows: for  $1 \leq i \leq m$ ,  $0 \leq j \leq n$ ,  $1 \leq t \leq \ell$ , let

$$\varphi(x_{i,t}) := \begin{cases} 1 & \text{if the } t\text{-th letter of } w \text{ is } a_i, \\ 0 & \text{otherwise;} \end{cases} \quad (2.6)$$

$$\varphi(y_{j,0}) := 1; \quad (2.7)$$

$$\varphi(y_{j,t}) := \begin{cases} 1 & \text{if the state } q_j \text{ lies in } Q.w_t, \\ 0 & \text{otherwise.} \end{cases} \quad (2.8)$$

In view of (2.6) and (2.7),  $\varphi$  satisfies all clauses in  $L$  and respectively  $I$ . As  $w_\ell = w$  and  $|Q.w| = 1$ , we see that (2.8) ensures that  $\varphi$  satisfies all clauses in  $S$ . It remains to analyze the clauses in  $T$ . For each fixed  $t = 1, \dots, \ell$ , these clauses are in a 1-1 correspondence with the pairs in  $Q \times \Sigma$ . We fix such a pair  $(q_j, a_i)$ , denote the clause corresponding to  $(q_j, a_i)$  by  $c$  and consider three cases.

**Case 1:** *the letter  $a_i$  is not the  $t$ -th letter of  $w$ .* In this case  $\varphi(x_{i,t}) = 0$  by (2.6), and hence,  $\varphi(c) = 1$  since the literal  $\neg x_{i,t}$  occurs in  $c$ , independently of  $c$  having the form (2.3) or (2.4).

**Case 2:** *the letter  $a_i$  is the  $t$ -th letter of  $w$  but it is undefined at  $q_j$ .* In this case the clause  $c$  must be of the form (2.4). Observe that  $t > 1$  in this case since the first letter of the carefully synchronizing word  $w$  must be defined at each state in  $Q$ . Moreover, the state  $q_j$  cannot belong to the set  $Q.w_{t-1}$  because  $a_i$  must be defined at each state in this state. Hence  $\varphi(y_{j,t-1}) = 0$  by (2.8), and  $\varphi(c) = 1$  since the literal  $\neq y_{j,t-1}$  occurs in  $c$ .

**Case 3:** the letter  $a_i$  is the  $t$ -th letter of  $w$  and it is defined at  $q_j$ . In this case the clause  $c$  must be of the form (2.3), in which the literal  $y_{k,t}$  corresponds to the state  $q_k = q_j \cdot a_i$ . If the state  $q_j$  does not belong to the set  $Q \cdot w_{t-1}$ , then as in the previous case, we have  $\varphi(y_{j,t-1}) = 0$  and  $\varphi(c) = 1$ . If  $q_j$  belongs to  $Q \cdot w_{t-1}$ , then the state  $q_k$  belongs to the set  $(Q \cdot w_{t-1}) \cdot a_i = Q \cdot w_t$ , whence  $\varphi(y_{k,t}) = 1$  by (2.8). We conclude that  $\varphi(c) = 1$  since the literal  $y_{k,t}$  occurs in  $c$ .

Conversely, suppose that  $\varphi: V \rightarrow \{0, 1\}$  is a satisfying assignment for  $(V, C)$ . Since  $\varphi$  satisfies the clauses in  $L$ , for each  $t = 1, \dots, \ell$ , there exists a unique  $i \in \{1, \dots, m\}$  such that  $\varphi(x_{i,t}) = 1$ . This defines a map  $\chi: \{1, \dots, \ell\} \rightarrow \{1, \dots, m\}$ . Let  $w := a_{\chi(1)} \cdots a_{\chi(\ell)}$ . We aim to show that  $w$  is a carefully synchronizing word for  $\mathcal{A}$ , i.e., to verify that  $w$  fulfils the conditions (C1)–(C3) from the definition of a carefully synchronizing word. For this, we first prove two auxiliary claims. Recall that a state is said to be active after  $t$  steps if it lies in  $Q \cdot w_t$ , where, as above,  $w_t$  is the length  $t$  prefix of the word  $w$ . (By the length 0 prefix we understand the empty word  $\varepsilon$ .)

**Claim 1.** For each  $t = 0, 1, \dots, \ell$ , there are states active after  $t$  steps.

**Claim 2.** If a state  $q_k$  is active after  $t$  steps, then  $\varphi(y_{k,t}) = 1$ .

We prove both claims simultaneously by induction on  $t$ . The induction basis  $t = 0$  is guaranteed by the fact that all states are active after 0 steps and  $\varphi$  satisfies the clauses in  $I$ . Now suppose that  $t > 0$  and there are states active after  $t - 1$  steps. Let  $q_r$  be such a state. Then  $\varphi(y_{r,t-1}) = 1$  by the induction assumption. Let  $i := \chi(t)$ , that is,  $a_i$  is the  $t$ -th letter of the word  $w$ . Then  $\varphi(x_{i,t}) = 1$ , whence  $\varphi$  cannot satisfy the clause of the form (2.4) with  $j = r$ . Hence this clause cannot

appear in  $T$  as  $\varphi$  satisfies the clauses in  $T$ . This means that the letter  $a_i$  is defined at  $q_r$  in  $\mathcal{A}$ , and the state  $q_s := q_r.a_i$  is active after  $t$  steps. Claim 1 is proved.

Now let  $q_k$  be an arbitrary state that is active after  $t > 0$  steps. Since  $a_i$  is the  $t$ -th letter of  $w$ , we have  $Q.w_t = (Q.w_{t-1}).a_i$ , whence  $q_k = q_j.a_i$  for some  $q_j \in Q.w_{t-1}$ . Therefore the clause (2.3) occurs in  $T$ , and thus, it is satisfied by  $\varphi$ . Since  $q_j$  is active after  $t - 1$  steps,  $\varphi(y_{j,t-1}) = 1$  by the induction assumption; besides that,  $\varphi(x_{i,t}) = 1$ . We conclude that in order to satisfy (2.3), the assignment  $\varphi$  must fulfil  $\varphi(y_{k,t}) = 1$ . This completes the proof of Claim 2.

We turn to prove that the word  $w$  fulfils (C1) and (C2). This amounts to verifying that for each  $t = 1, \dots, \ell$ , the  $t$ -th letter of the word  $w$  is defined at every state  $q_j$  that is active after  $t - 1$  steps. Let, as above,  $a_i$  stand for the  $t$ -th letter of  $w$ . If  $a_i$  were undefined at  $q_j$ , then by the definition of the set  $T$  of transition clauses, this set would include the corresponding clause (2.4). However,  $\varphi(x_{i,t}) = 1$  by the construction of  $w$  and  $\varphi(y_{j,t-1}) = 1$  by Claim 2. Hence  $\varphi$  does not satisfy this clause while the clauses from  $T$  are satisfied by  $\varphi$ , a contradiction.

Finally, consider (C3). By Claim 1, some state is active after  $\ell$  steps. On the other hand, the assignment  $\varphi$  satisfies the clauses in  $S$ , which means that  $\varphi(y_{j,\ell}) = 1$  for at most one index  $j \in \{1, \dots, n\}$ . By Claim 2 this implies that at most one state is active after  $\ell$  steps. We conclude that exactly one state is active after  $\ell$  steps, that is,  $|Q.w| = 1$ .  $\square$

## 2.2 Exactly synchronizing words

In this section, we deal with the second version of PFAs synchronization, that is, exact synchronization. We consider the following problem.

ESW: the existence of an exactly synchronizing word of a given length  
 INPUT: a PFA  $\mathcal{A}$  and a positive integer  $\ell$  (given in unary);  
 OUTPUT: YES if  $\mathcal{A}$  has an exactly synchronizing word of length  $\ell$ ; NO otherwise.

Here we use the same set of variables as in Section 2.1 but the set of clauses is essentially different.

The set  $C$  of clauses in the encoding of  $(\mathcal{A}, \ell)$  as an instance of ESW consists of four groups  $I, L, T, S$ , where the groups  $I$  and  $L$  play the role of initial clauses and letter clauses. The main job of the groups  $I$  and  $L$  is encoding the automaton at its initial position and define the letters of a hypothetical exactly synchronizing word. Hence their construction will be the same as in Clauses (2.1) and (2.2) respectively. In contrast, the groups  $T$  and  $S$  are very sensitive to which version of synchronization is considered. In the following we discuss construction of these groups.

The group  $T$  consists of clauses that encode the transitions of  $\mathcal{A}$  under the hypothetical exactly synchronizing word. For a state  $q_j \in Q$ , let  $P_i(q_j)$  stand for the set of all preimages of  $q_j$  under the action of the letter  $a_i$ , that is,

$$P_i(q_j) := \{p \in Q \mid p.a_i = q_j\}.$$

Consider for every  $t = 1, \dots, \ell$  and every  $j = 1, \dots, n$ , the following formula:

$$y_{j,t} \iff \bigvee_{i=1}^m \left( x_{i,t} \wedge \bigvee_{q_k \in P_i(q_j)} y_{k,t-1} \right). \quad (2.9)$$

Recall that  $m$  stands for the number of letters in the alphabet  $\Sigma$ . Observe that the equivalence (2.9) just expresses, in the language of propositional logic, the fact that the state  $q_j$  is active after  $t$  steps if and only if some preimage of  $q_j$  under the action of the  $t$ -th letter of  $w$  is active after  $t-1$  steps. A direct conversion of (2.9) into a conjunctive normal form leads to quite a bulky system of clauses. Instead, we use the following lemma.

**Lemma 2.1.** *Fix numbers  $t \in \{1, \dots, \ell\}$  and  $j \in \{1, \dots, n\}$  and take any truth assignment  $\varphi: V \rightarrow \{0, 1\}$  such that  $\varphi(x_{i,t}) = 1$  for exactly one value of  $i \in \{1, \dots, m\}$ . Then  $\varphi$  satisfies the equivalence (2.9) if and only if  $\varphi$  satisfies the following system of clauses:*

$$\neg y_{j,t} \vee \neg x_{i,t} \vee \bigvee_{q_k \in P_i(q_j)} y_{k,t-1} \quad \text{for each } i \in \{1, \dots, m\}, \quad (2.10)$$

$$y_{j,t} \vee \neg x_{i,t} \vee \neg y_{k,t-1} \quad \text{for each } i \in \{1, \dots, m\} \text{ and each } q_k \in P_i(q_j). \quad (2.11)$$

*Proof.* Let  $i_0$  be such that  $\varphi(x_{i_0,t}) = 1$ . Then  $\varphi(x_{i,t}) = 0$  for all  $i \neq i_0$  whence the right-hand side of the equivalence (2.9) gets the same value under  $\varphi$  as the expression  $\bigvee_{q_k \in P_{i_0}(q_j)} y_{k,t-1}$ . Thus, if (2.9) is satisfied by  $\varphi$ , so are the implications

$$y_{j,t} \rightarrow \bigvee_{q_k \in P_{i_0}(q_j)} y_{k,t-1}, \quad (2.12)$$

$$\bigvee_{q_k \in P_{i_0}(q_j)} y_{k,t-1} \rightarrow y_{j,t}. \quad (2.13)$$

The implication (2.12) is equivalent to the clause  $\neg y_{j,t} \vee \bigvee_{q_k \in P_{i_0}(q_j)} y_{k,t-1}$ , and we see that  $\varphi$  satisfies the clause (2.10) with  $i = i_0$ . The implication (2.13) is equivalent to the system of clauses  $y_{j,t} \vee \neg y_{k,t-1}$  where  $k$  runs over the indices of states in  $P_{i_0}(q_j)$ . Hence, if  $\varphi$  satisfies (2.13), we see that  $\varphi$  satisfies all clauses (2.11) with  $i = i_0$ . Besides that,  $\varphi$  satisfies all clauses (2.10) and (2.11) with  $i \neq i_0$  because each of these clauses includes  $\neg x_{i,t}$  as a literal and  $\varphi(x_{i,t}) = 0$  for all  $i \neq i_0$ .

Thus, we have shown that if  $\varphi$  satisfies the equivalence (2.9), then  $\varphi$  also satisfies all clauses (2.10) and (2.11). All our arguments are reversible, and therefore, the converse claim holds as well.  $\square$

We collect clauses of the form (2.10) and (2.11) for all  $t = 1, \dots, \ell$  and  $j = 1, \dots, n$  in the group  $T$  of transition clauses. There are  $\ell mn$  clauses of the form (2.10); as for clauses of the form (2.11), its number for each triple  $(i, j, t)$  depends on the cardinality of the set  $P_i(q_j)$ , which clearly does not exceed  $n$ . Hence the number of clauses of the form (2.11) is at most  $\ell mn^2$  whence  $T$  contains at most  $\ell mn(n+1)$  clauses in total.

It may be worth explaining how the clauses of the form (4.5) and (2.11) are understood in the case when one of the sets  $P_i(q_j)$  happens to be empty. In (2.10) the disjunction over the empty set is omitted so that if  $P_i(q_j) = \emptyset$ , then the clause (2.10) reduces to  $\neg y_{j,t} \vee \neg x_{i,t}$ . The latter clause simply means that if the  $t$ -th letter of  $w$  is  $a_i$  and the state  $q_j$  has no preimage under  $a_i$ , then  $q_j$  cannot be active after  $t$  steps. As for (2.11), the clauses of this sort disappear for all  $i$  such that  $P_i(q_j)$  is empty.

The final group  $S$  of *synchronization clauses* describes the situation at the end of the synchronization process when the action of the word  $w$  is completed. It consists of the following clauses:

$$y_{1,\ell} \vee \cdots \vee y_{n,\ell}, \tag{2.14}$$

$$\neg y_{r,\ell} \vee \neg y_{s,\ell}, \quad \text{where } 1 \leq r < s \leq n. \tag{2.15}$$

Clearly, the clauses in (2.14) and (2.15) express the fact that exactly one state remains active after  $\ell$  steps, which corresponds to the requirement  $|Q.w| = 1$ . The group  $S$  contains  $\frac{n(n-1)}{2} + 1 = \frac{n(n+1)}{2}$  clauses.

We let  $C := I \cup L \cup T \cup S$ . The number of clauses in the set  $C$  is no greater than  $\ell m \left( \frac{(m+1)}{2} + n(n+1) \right) + \frac{n(n+3)}{2}$ . Now we can state and prove the main theorem of this section.

**Theorem 2.2.** *A PFA  $\mathcal{A}$  has an exactly synchronizing word of length  $\ell$  if and only if the instance  $(V, C)$  of SAT constructed above is satisfiable, and the construction of this instance can be done in polynomial time. Moreover, the exactly synchronizing words of length  $\ell$  for  $\mathcal{A}$  are in a one-to-one correspondence with the satisfying assignments of  $(V, C)$ .*

As the reader sees, Theorem 2.2 looks similar to Theorem 2.1 that dealt with carefully synchronizing words. The arguments in the proof of Theorem 2.1 partly repeat the ones from the proof of Theorem 2.2. However, for the reader's convenience we have decided to present the proof of Theorem 2.2 in full details, without referring to analogous arguments in the proof of Theorem 2.1.

*Proof.* We keep the notation and terminology introduced in the course of the construction of our encoding  $(\mathcal{A}, \ell) \mapsto (V, C)$ . In particular,  $\mathcal{A} = (Q, \Sigma)$  with  $Q = \{q_1, \dots, q_n\}$  and  $\Sigma = \{a_1, \dots, a_m\}$ . The fact that the instance  $(V, C)$  can be constructed in polynomial of  $n$ ,  $m$ , and  $\ell$  time follows from the estimates of  $|V|$  and  $|C|$  established above.



Now suppose that  $\mathcal{A}$  has an exactly synchronizing word of length  $\ell$  and fix such a word  $w$ . We define a truth assignment  $\varphi: V \rightarrow \{0, 1\}$  as follows: for the letter variables  $x_{i,t}$  with  $1 \leq i \leq m$ ,  $1 \leq t \leq \ell$ ,

$$\varphi(x_{i,t}) = \begin{cases} 1 & \text{if the } t\text{-th letter of } w \text{ is } a_i, \\ 0 & \text{otherwise;} \end{cases}$$

for the state variables  $y_{j,t}$  with  $1 \leq j \leq n$ ,  $0 \leq t \leq \ell$ ,

$$\varphi(y_{j,t}) = \begin{cases} 1 & \text{if the state } q_j \text{ is active after } t \text{ steps,} \\ 0 & \text{otherwise.} \end{cases}$$

Clearly,  $\varphi$  satisfies all clauses in  $L$  and  $I$ . Since  $w$  is an exactly synchronizing word, exactly one state remains active after  $\ell$  steps, whence  $\varphi$  satisfies all clauses in  $S$ . By the construction,  $\varphi$  satisfies the formulas (2.9) for  $t = 1, \dots, \ell$  and  $j = 1, \dots, n$  as these formulas describe the propagation of active states. Since  $\varphi$  is such that  $\varphi(x_{i,t}) = 1$  for exactly one value of  $i \in \{1, \dots, m\}$ , Lemma 2.1 ensures that  $\varphi$  also satisfies all clauses in  $T$ . We see that the SAT-instance  $(V, C)$  is satisfied by  $\varphi$ .

Conversely, suppose that  $(V, C)$  is satisfiable and fix a satisfying assignment  $\varphi$  for  $(V, C)$ . Since  $\varphi$  satisfies all clauses in  $L$ , for each  $t = 1, \dots, \ell$ , there exists exactly one index  $i(t) \in \{1, \dots, m\}$  such that  $\varphi(x_{i(t),t}) = 1$ . Define a word  $w := a_{i(1)} \cdots a_{i(\ell)}$  and let  $w_t$  be the prefix of  $w$  of length  $t$  for  $t = 0, 1, \dots, \ell$ . (Here  $w_0$  is the empty word  $\varepsilon$ .) We aim to prove that for each  $t = 0, 1, \dots, \ell$ ,

$$Q.w_t = \{q_j \mid \varphi(y_{j,t}) = 1\}. \tag{2.16}$$

We induct on  $t$ . The claim (2.16) holds for  $t = 0$  since  $\varphi$  satisfies all clauses in  $I$ . Now suppose that  $t > 0$ . Lemma 2.1 applies to  $\varphi$  whence the condition that  $\varphi$  satisfies all clauses in  $I$  implies that  $\varphi$  also satisfies the equivalences (2.9) for  $t = 1, \dots, \ell$  and  $j = 1, \dots, n$ . By the induction assumption, we have that a state  $q_k$  is active after  $t - 1$  steps if and only if  $\varphi(y_{k,t-1}) = 1$ . As mentioned, the equivalence (2.9) translates into the language of propositional logic that  $q_j$  is active after  $t$  steps if and only if some preimage of  $q_j$  under the action of the letter  $a_{i(t)}$  is active after  $t - 1$  steps; on the other hand, since  $\varphi$  satisfies (2.9), we have  $\varphi(y_{j,t}) = 1$  if and only if  $\varphi(y_{k,t-1}) = 1$  for some  $k \in P_{i(t)}(q_j)$ . Combining these two facts, we get (2.16).

Since  $\varphi$  satisfies all clauses in  $S$ , the equality  $\varphi(y_{j,\ell}) = 1$  holds for exactly one value of  $j \in \{1, \dots, n\}$ . By (2.16), this means that  $|Q.w_\ell| = 1$ , that is,  $w = w_\ell$  is an exactly synchronizing word for  $\mathcal{A}$ .

We see that any exactly synchronizing word of length  $\ell$  for  $\mathcal{A}$  determines a satisfying assignment for  $(V, C)$  and vice versa. Moreover, from the above proof it is clear that if we start with an exactly synchronizing word  $w$  of length  $\ell$ , construct from  $w$  a satisfying assignment  $\varphi$  for  $(V, C)$ , and then build an exactly synchronizing word from  $\varphi$ , we get back the word  $w$ . Thus, the correspondence between the exactly synchronizing words of length  $\ell$  for  $\mathcal{A}$  and the satisfying assignments of  $(V, C)$  is indeed one-to-one.  $\square$

*Remark 2.2.* Encoding of ESW uses the same set of variables as the above encoding for CSW but the set of clauses is essentially different. One may think that since the definition of an exactly synchronizing word differs from the definition of a carefully synchronizing word by the absence of the conditions (C1) and (C2), one could get an encoding for ESW

by just omitting the clauses (2.4) that control these conditions in the encoding for CSW, and vice versa, one could encode CSW by appending the clauses (2.4) to the encoding for ESW. However, it is easy to exhibit counterexamples to show that such a naive transformation of our CSW encoding into an encoding for ESW fails. In the converse direction, the transformation produces a valid encoding for CSW but this encoding has many more clauses than the CSW encoding suggested here.

For an illustration of the presented encodings, we consider the automaton in Figure 2.1. Table 2.1 shows the encodings of  $(\mathcal{E}_p, 1)$  as an instance of CSW and ESW respectively.

<b>Clauses</b>	<b>CSW</b> $(\mathcal{E}_p, 1)$	<b>ESW</b> $(\mathcal{E}_p, 1)$
I	$y_{0,0}$ $y_{1,0}$	$y_{0,0}$ $y_{1,0}$
T	$\neg y_{0,0} \vee \neg x_1 \vee y_{1,1}$ $\neg y_{1,0} \vee \neg x_1$ $\neg y_{0,0} \vee x_1 \vee y_{0,1}$ $\neg y_{1,0} \vee x_1 \vee y_{0,1}$	$\neg y_{1,1} \vee y_{0,0}$ $\neg y_{1,1} \vee x_1$ $\neg y_{0,1} \vee y_{0,0} \vee y_{1,0}$ $\neg y_{0,1} \vee \neg x_1$ $\neg y_{0,0} \vee \neg x_1 \vee y_{1,1}$ $\neg y_{1,0} \vee \neg x_1$ $\neg y_{0,0} \vee x_1 \vee y_{0,1}$ $\neg y_{1,0} \vee x_1 \vee y_{0,1}$
S	$\neg y_{1,1} \vee \neg y_{0,1}$	$y_{1,1} \vee y_{0,1}$ $\neg y_{1,1} \vee \neg y_{0,1}$

Table 2.1: The SAT encoding of the CSW and ESW instances  $(\mathcal{E}_p, 1)$

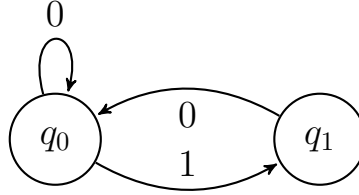


Figure 2.1: The PFA  $\mathcal{E}_p$

## 2.3 Ladder encoding

Observe that all clauses in (2.5) and (2.15) and a majority of clauses in (2.2) are typical “at-most-one” constraints. There are various way to express such constraints by fewer clauses. In our implementation, we have used the so-called ladder encoding suggested in [27], see also [6, Chapter 2]. We demonstrate how the ladder encoding works on the set  $S$ . We introduce  $n - 1$  additional variables  $f_1, f_2, \dots, f_{n-1}$  and substitute the clauses (2.5) for CSW and (2.15) for ESW by two new groups of clauses: the *ladder validity* clauses

$$\neg f_{j+1} \vee f_j \tag{2.17}$$

for  $j = 1, 2, \dots, n - 2$ , and the *channelling* clauses that correspond to the equivalence  $y_{j,\ell} \iff f_{j-1} \wedge \neg f_j$ :

$$\neg f_{j-1} \vee f_j \vee y_{j,\ell}, \quad \neg y_{j,\ell} \vee f_{j-1}, \quad \neg y_{j,\ell} \vee \neg f_j \tag{2.18}$$

for  $j = 1, 2, \dots, n$ , where the clauses containing  $f_0$  or  $f_n$  are simplified as if  $f_0 = 1$  or  $f_n = 0$ . Altogether, we get  $4n - 4$  clauses in (2.17) and (2.18) instead of  $\frac{n(n-1)}{2}$  clauses in (2.5) on the price of adding  $n - 1$  extra variables. The same trick allows us to decrease the number of clauses in

the set  $L$ , but this is less important because the parameter  $m$  (the size of the input alphabet) is usually small. Our experiments have shown that using ladder encoding significantly reduces time needed to solve CSW and ESW instances, especially for automata with large number of states.

*Remark 2.3.* In a majority of our experiments, we deal with PFAs that have only two input letters. As above, we call such PFAs binary. To encode the CSW/ESW instance  $(\mathcal{A}, \ell)$ , where  $\mathcal{A}$  is a binary PFA, we can use only  $\ell$  letter variables  $x_1, \dots, x_\ell$  to encode the letters of a hypothetical carefully/exactly synchronizing word  $w$  of length  $\ell$  since there is an obvious 1 - 1 correspondence between the truth assignments on the set  $\{x_1, \dots, x_\ell\}$  and the words of length  $\ell$  over any fixed 2-letter alphabet. In more formal terms, we can modify the encoding of CSW and ESW, substituting  $x_t$  for  $x_{1,t}$  and  $\neg x_t$  for  $x_{2,t}$  for all  $t = 1, 2, \dots, \ell$  in all clauses in which  $x_{1,t}$  or  $x_{2,t}$  occur. Observe that the letter clauses (2.2) become tautologies after this substitution, and hence, they can be safely omitted. Thus, for a binary PFA  $\mathcal{A}$  with  $n$  states, we may encode the CSW instance  $(\mathcal{A}, \ell)$  into a SAT instance with  $\ell(n + 2) + n - 2$  variables and only  $2\ell n + 5n - 4$  clauses if we use both the modification just described and the ladder encoding.

## Chapter 3

# Experimental study in PFAs synchronization

It was discussed in Chapter 1 that the upper bound for the length of the shortest exactly or carefully synchronizing word may be an exponential function in the number of states. But what about the average of such length? What are the parameters that may affect this length? In this chapter we present an experimental study from which we can partly answer these questions.

### 3.1 General settings of our experiments

The general framework of our experiments with random automata consists of the following basic steps.

1. A positive integer  $n$  (the number of states) is fixed.
2. A random PFA  $\mathcal{A}$  with  $n$  states is generated.

3. The pair  $(\mathcal{A}, 1)$  is encoded into a SAT instance  $(V', C')$  as described in Section 2.1 (if we study careful synchronization) or in Section 2.2 (if we study exact synchronization).
4. The instance  $(V', C')$  is scaled to the instance  $(V, C)$  that encodes the pair  $(\mathcal{A}, \ell)$ , see Remark 3.1 below.
5. The MiniSat 2.2.0 is invoked to solve the SAT instance  $(V, C)$ .

We refer to [20] for a description of the underlying ideas of the SAT solver MiniSat and to [21] for a discussion and the source code of the solver.

*Remark 3.1.* An important feature of our encodings is that as soon as we have constructed the “primary” SAT instance  $(V', C')$  that encodes the CSW/ESW instance  $(\mathcal{A}, 1)$ , we are in a position to scale  $(V', C')$  to the SAT instance encoding the CSW/ESW instance  $(\mathcal{A}, \ell)$  for any value of  $\ell$ . In order to explain this feature, recall that MiniSAT accepts its input in the following text format (so-called simplified DIMACS CNF format). Every line beginning with `c` is a comment. The first non-comment line is of the form:

```
p cnf NUMBER_OF_VARIABLES NUMBER_OF_CLAUSES
```

Variables are represented by integers from 1 to `NUMBER_OF_VARIABLES`. The first non-comment line is followed by `NUMBER_OF_CLAUSES` non-comment lines each of which defines a clause. Every such line starts with a space-separated list of different non-zero integers corresponding to the literals of the clause: a positive integer corresponds to a literal which is a variable, and a negative integer corresponds to a literal which is the negation of a variable; the line ends in a space and the number 0.

For simplicity, we describe the scaling procedure for binary PFAs only and we assume that the ladder encoding not been used. (Both the generalization to PFAs over larger alphabets and the modification needed to accommodate additional variables involved in the ladder encoding are fairly straightforward.) Given a binary PFA  $\mathcal{A}$  with  $n$  states, we write the SAT instance  $(V', C')$ , which corresponds to  $(\mathcal{A}, 1)$ , in DIMACS CNF format, representing the variables  $x_1, y_{j,0}, y_{j,1}, j = 1, \dots, n$ , by the numbers, respectively,  $n + 1, j, j + n + 1$ . For an illustration, see Table 3.1 that shows the SAT encoding of CSW for the PFA  $\mathcal{E}_4$  from Figure 3.1.

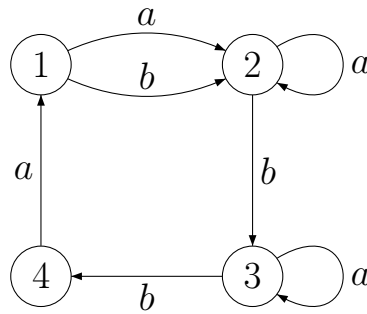


Figure 3.1: The automaton  $\mathcal{E}_4$



Table 3.1: The SAT encoding of the CSW instance ( $\mathcal{E}_4, 1$ )

Clauses	DIMACS CNF lines
	p cnf 9 18
$I'$ $\left\{ \begin{array}{l} y_{1,0} \\ y_{2,0} \\ y_{3,0} \\ y_{4,0} \end{array} \right.$	$\begin{array}{l} 1 \ 0 \\ 2 \ 0 \\ 3 \ 0 \\ 4 \ 0 \end{array}$
$T'$ $\left\{ \begin{array}{l} \neg y_{1,0} \vee \neg x_1 \vee y_{2,1} \\ \neg y_{2,0} \vee \neg x_1 \vee y_{2,1} \\ \neg y_{3,0} \vee \neg x_1 \vee y_{3,1} \\ \neg y_{4,0} \vee \neg x_1 \vee y_{1,1} \\ \neg y_{1,0} \vee x_1 \vee y_{2,1} \\ \neg y_{2,0} \vee x_1 \vee y_{3,1} \\ \neg y_{3,0} \vee x_1 \vee y_{4,1} \\ \neg y_{4,0} \vee x_1 \end{array} \right.$	$\begin{array}{l} -1 \ -5 \ 7 \ 0 \\ -2 \ -5 \ 7 \ 0 \\ -3 \ -5 \ 8 \ 0 \\ -4 \ -5 \ 6 \ 0 \\ -1 \ 5 \ 7 \ 0 \\ -2 \ 5 \ 8 \ 0 \\ -3 \ 5 \ 9 \ 0 \\ -4 \ 5 \ 0 \end{array}$
$S'$ $\left\{ \begin{array}{l} \neg y_{1,1} \vee \neg y_{2,1} \\ \neg y_{1,1} \vee \neg y_{3,1} \\ \neg y_{1,1} \vee \neg y_{4,1} \\ \neg y_{2,1} \vee \neg y_{3,1} \\ \neg y_{2,1} \vee \neg y_{4,1} \\ \neg y_{3,1} \vee \neg y_{4,1} \end{array} \right.$	$\begin{array}{l} -6 \ -7 \ 0 \\ -6 \ -8 \ 0 \\ -6 \ -9 \ 0 \\ -7 \ -8 \ 0 \\ -7 \ -9 \ 0 \\ -8 \ -9 \ 0 \end{array}$

Now, in order to scale  $(V', C')$  to the SAT instance  $(V, C)$  that encodes the pair  $(\mathcal{A}, \ell)$  for some given  $\ell > 1$  one has to transform the DIMACS CNF representation of  $C' = I' \cup T' \cup S'$  as described in Figure 3.2 and in the following steps:

1. In the first non-comment line, replace the numbers  $2n + 1$  and  $2n + \frac{n(n+1)}{2}$  by respectively  $(\ell + 1)n + \ell$  and  $2\ell n + \frac{n(n+1)}{2}$ .
2. Apply the following three commands
  - 2.1. **k**: Keep the lines corresponding to the clauses in  $C'_0$  and  $C'_1$ .
  - 2.2. **rt**: For each  $t = 2, \dots, \ell$ , add all the lines obtained from the lines that correspond to the clauses in  $T'$  by keeping the sign of every non-zero integer and adding  $(t - 1)(n + 1)$  to its absolute value.
  - 2.3. **rs**: In each line corresponding to a clause in  $S'$ , substitute every nonzero integer  $\pm k$  by the integer  $\pm(k + (\ell - 1)(n + 1))$ .

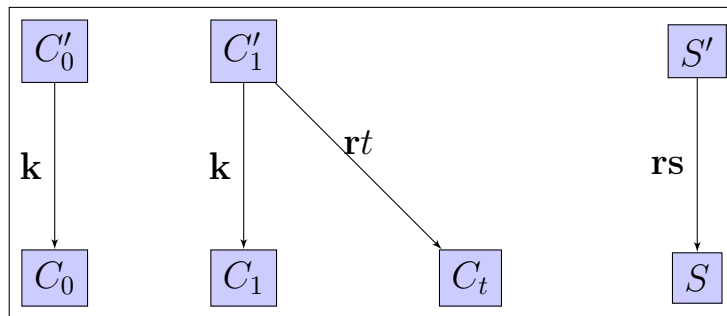


Figure 3.2: Scaling procedure

## 3.2 Experiments and implementation

We have performed four series of experiments with random PFAs.

Series 1: studying the probability of being synchronizing for each version of synchronization for randomly generated binary PFAs with one undefined transition.

Series 2: finding an approximation for the average length of shortest carefully synchronizing words and exactly synchronizing words for randomly generated binary PFAs with one undefined transition.

Series 3: studying the influence of the input alphabet size on the length of the shortest synchronizing word.

Series 4: studying the influence of the *density* (the number of defined transitions) on the length of the shortest synchronizing word.

All our algorithms were implemented in C++ and compiled with GCC 4.9.2. In our experiments we used a personal computer with an Intel(R) Core(TM) i5-2520M processor with 2.5 GHz CPU and 4GB of RAM. Our code and datasets are available under <https://github.com/hananshabana/SynchronizationChecker>.

## 3.3 Generating random PFAs

In experiments from Series 1 and 2 we worked with binary PFAs  $\mathcal{A} = (Q, \Sigma, \delta)$  with  $n \leq 100$  states and only one undefined transition. Then one letter must be everywhere defined; we denoted it by  $a$  and selected the action of  $a$  uniformly at random from all  $n^n$  maps  $Q \rightarrow Q$ .

To ensure that there is a unique undefined transition with  $b$ , we chose uniformly at random a state  $q_b \in Q$  and then selected the action of  $b$  uniformly at random from all  $n^{n-1}$  maps  $Q \setminus \{q_b\} \rightarrow Q$ .

In experiments from Series 3, we again considered PFAs with an everywhere defined letter and defined its action as above. Then, for each of the remaining letters, we first chose the number  $k$  of states at which the letter should be defined;  $k$  was chosen uniformly at random from the set  $\{1, 2, \dots, n-1\}$ . Then we selected uniformly at random  $k$  different states from  $Q$ , and for each of these states we chose uniformly at random a state in  $Q$  as its image under the action of the letter.

In experiments from Series 4, we considered binary PFAs  $\mathcal{A}$  with  $n \leq 100$  states. Let  $\rho_{\mathcal{A}}$  stand for the density of  $\mathcal{A}$ . In experiments with careful synchronization, possible values of  $\rho_{\mathcal{A}}$  were chosen between  $n+1$  and  $2n-1$  since one of the letters must be everywhere defined. For the other letter, we set  $k := \rho_{\mathcal{A}} - n$  and then proceeded as in the preceding paragraph. In experiments with exact synchronization, the value of  $\rho_{\mathcal{A}}$  can be any number between 1 and  $2n-1$ . However, it is easy to realize that a PFA with density 1, that is, a PFA with a unique defined transition, is always exactly synchronizing by a word of length 1 (namely, by the letter which action at some state is defined). Thus, the case of density 1 is not interesting at all, and we chose possible values of  $\rho_{\mathcal{A}}$  from numbers between 2 and  $2n-1$ . Then we chose uniformly at random a non-negative number  $k \leq \rho_{\mathcal{A}}$  and applied the procedure described in the preceding paragraph first to  $k$  and then to  $\rho_{\mathcal{A}} - k$ .

## 3.4 Experimental results for randomly generated PFAs and their analysis

### 3.4.1 Series 1: Probability of synchronization

This series of our experiments aims to compare the probability of being exactly or carefully synchronizing for the same sample of random automata. Figure 3.3 shows the probability of being synchronizing in each of these two versions of synchronization for the class of binary PFAs  $(Q, \{a, b\})$  such that the letter  $a$  is everywhere defined and the letter  $b$  is undefined at exactly one state in  $Q$ . For brevity, we refer to automata from this class as *almost complete* PFAs. Observe that the problem of deciding whether or not a given PFA is carefully synchronizing remains PSPACE-complete even if restricted to this rather special case [58, Theorem 3]. For each fixed  $n$ , we generated up to 1000 random almost complete PFAs.

Let  $P_E(n)$  stand for the probability of a random almost complete PFA with  $n$  states to be exactly synchronizing and let  $P_C(n)$  be the probability that the same random PFA is carefully synchronizing. We have  $P_E(n) > P_C(n)$  since, as mentioned, every carefully synchronizing PFA is exactly synchronizing. The data in Figure 3.3 show that the gap between  $P_E(n)$  and  $P_C(n)$  decreases as  $n$  grows but remains non-negligible even for  $n$  close to 100. We also see that  $P_E(n)$  quickly tends to 1 as the state number grows. Recall the same effect was experimentally observed for DFAs and was then theoretically justified by Berlinkov [5] and Nicaud [63, 64]: the probability  $P_S(n)$  that a random binary DFA with  $n$  states is synchronizing tends to 1 as  $n$  tends to infinity. Moreover,

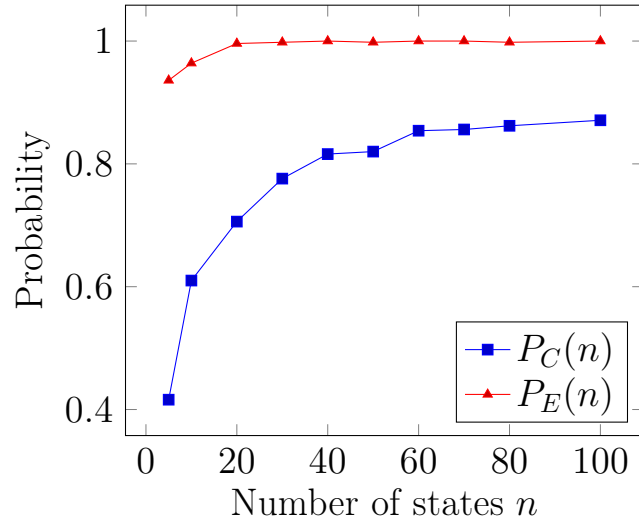


Figure 3.3: Probability of being synchronizing for two versions of synchronization

it is shown in [5] that  $1 - P_S(n) = O(\frac{1}{n})$ . It is not difficult to extend the latter result to random almost complete PFAs. Since we have not found such an extension in the literature, we have included it here, without any originality claim.

The extension is based on the following easy observation.

**Lemma 3.1.** *Let  $\mathcal{A}$  be a synchronizing DFA. Then every PFA obtained from  $\mathcal{A}$  by removing a single transition is exactly synchronizing.*

*Proof.* Let  $\mathcal{A} = (Q, \Sigma, \delta)$ . Fix an arbitrary pair  $(p, b) \in Q \times \Sigma$  and consider the PFA  $\mathcal{B} = (Q, \Sigma, \zeta)$ , where  $\zeta$  coincides with  $\delta$  on the set  $Q \times \Sigma \setminus \{(p, b)\}$  and  $\zeta(p, b)$  is undefined. Let  $w \in \Sigma^*$  be a synchronizing word for  $\mathcal{A}$  so that  $|\delta(Q, w)| = 1$ . Clearly, if  $w$  is defined in  $\mathcal{B}$  at some  $q \in Q$ , then  $\zeta(q, w) = \delta(q, w)$  whence  $|\zeta(Q, w)| = 1$  and  $w$  is an exactly synchronizing word for  $\mathcal{B}$ . Thus, assume that  $w$  is nowhere

defined in  $\mathcal{B}$ . Let  $v$  be the longest prefix of  $w$  which is defined in  $\mathcal{B}$  at some state and let  $x$  be the letter that follows  $v$  in  $w$ . By the choice of  $v$ , the set  $P := \zeta(Q, v)$  is not empty but  $\zeta(P, x)$  is empty. Thus, all transitions of the form  $\zeta(q, x)$  with  $q \in P$  must be undefined. However, by the definition of  $\zeta$ , the only undefined transition in  $\mathcal{B}$  is  $\zeta(p, b)$  whence  $x = b$  and  $P = \{p\}$ . In particular,  $|\zeta(Q, v)| = 1$  and  $v$  is an exactly synchronizing word for  $\mathcal{B}$ .  $\square$

Given an almost complete PFA  $\mathcal{B} = (Q, \{a, b\}, \zeta)$ , its *completion* is any DFA obtained by defining the only undefined transition of  $\mathcal{A}$ . Let  $n := |Q|$ . If the letter  $b$  is undefined at a certain state  $p \in Q$ , then we can choose any state in  $Q$  as the image of  $p$  under  $b$  in the completion, whence  $\mathcal{B}$  has  $n$  different completions. Conversely, any given DFA  $\mathcal{A} = (Q, \{a, b\}, \delta)$  serves as a completion for  $n$  different almost complete PFAs obtained from  $\mathcal{A}$  by ‘forgetting’ the value of  $\delta(p, b)$ , where  $p$  runs over  $Q$ .

Now consider the set  $\mathbf{P}$  of all pairs  $(\mathcal{B}, \mathcal{A})$  such that  $\mathcal{A}$  is a completion of  $\mathcal{B}$  and  $\mathcal{B}$  is not exactly synchronizing. Denoting by  $N$  the number of almost complete PFAs with the state set  $Q$  that are not exactly synchronizing, we have  $|\mathbf{P}| = Nn$ . Lemma 3.1 implies that no DFA  $\mathcal{A}$  such that there is  $\mathcal{B}$  with  $(\mathcal{B}, \mathcal{A}) \in \mathbf{P}$  can be synchronizing. Any non-synchronizing DFA may occur in at most  $n$  pairs from  $\mathbf{P}$  whence the number  $M$  of DFAs with the state set  $Q$  that are not synchronizing satisfies  $M \geq \frac{|\mathbf{P}|}{n} = \frac{Nn}{n} = N$ . Observe that the total number  $n^{2n}$  of binary DFAs is the same as the total number of almost complete PFAs: to construct an almost complete PFA with  $n$  states, we have  $n^n$  choices for the action of the everywhere defined letter,  $n$  choices for a state at which the other letter is undefined, and  $n^{n-1}$  choices for the action of

the latter letter at the remaining  $n - 1$  states. Therefore, we conclude that

$$1 - P_E(n) = \frac{N}{n^{2n}} \leq \frac{M}{n^{2n}} = 1 - P_S(n) = O\left(\frac{1}{n}\right).$$

Hence,  $1 - P_E(n) = O\left(\frac{1}{n}\right)$ .

For complete deterministic binary automata, Berlinkov [5] has shown that the bound  $1 - P_S(n) = O\left(\frac{1}{n}\right)$  is tight, that is,  $1 - P_S(n) = \Theta\left(\frac{1}{n}\right)$ . We have proved the same result for almost complete PFAs. For this we describe a construction that yields “sufficiently many” almost complete PFAs with  $n$  states and 2 letters  $a$  and  $b$  that are not exactly synchronizing. The construction is as follows. First we choose a state  $q_0$  at which  $b$  is undefined. There are  $n$  choices for  $q_0$ . Then we define the action of  $a$  at  $q_0$  in an arbitrary way. This gives  $n$  choices. After that, there are  $n - 1$  choices for the state  $q_1$  which is fixed by both  $a$  and  $b$ . Finally, there are  $(n - 2)^{2(n-2)}$  choices for the actions of  $a$  and  $b$  at the remaining  $n - 2$  states. Altogether, the construction gives  $n^2(n - 1)(n - 2)^{2(n-2)}$  almost complete automata which are not exactly synchronizing. Now when we calculate the fraction  $n^2(n - 1)(n - 2)^{2(n-2)}/n^{2n}$ , we get

$$\frac{n^2(n - 1)(n - 2)^{2(n-2)}}{n^{2n}} = \left(1 - \frac{1}{n}\right) \left(1 - \frac{2}{n}\right)^{2n} \left(1 - \frac{2}{n}\right)^{-4} \frac{1}{n}.$$

As  $n$  tends to the infinity, the first and the third factors tend to 1, and the second factor tends to  $e^{-4}$ . Thus, the fraction is asymptotically equivalent to  $\frac{e^{-4}}{n}$ . Hence  $1 - P_E(n) = \Omega\left(\frac{1}{n}\right)$ .

Summarizing, we have

**Proposition 3.1.**  $1 - P_E(n) = \Theta\left(\frac{1}{n}\right)$ .

Back to Figure 3.3, we see that the probability  $P_C(n)$  also grows



with  $n$  but it not clear if it tends to 1 as  $n$  tends to infinity. To the best of our knowledge, no theoretical results published so far predict the asymptotic behavior of the function  $P_C(n)$  nor, more generally, the asymptotic behavior of the probability of being carefully synchronizing for any class of random PFAs. Here, as a result on analysis of the outcome of our experiments, we are able to show that even if  $P_C(n)$  does approach 1 as  $n \rightarrow \infty$ , it does it at much slower rate than  $P_E(n)$ ; see the discussion at the end of the subsection.

First, let us discuss how we proceeded to determine if a PFA  $\mathcal{A}$  from our sample was carefully/exactly synchronizing. According to the general scheme described in Section 3.1, we encoded  $(\mathcal{A}, 1)$  as a SAT instance, wrote the instance in DIMACS CNF format, and scaled it to the instances encoding  $(\mathcal{A}, \ell)$  with  $\ell = 2, 4, 8, \dots$  until we reached an instance on which the SAT solver returned YES. Of course, sometimes it happened that we did not reach such an instance which indicated that either  $\mathcal{A}$  was not carefully/exactly synchronizing or the minimum length of carefully/exactly synchronizing words for  $\mathcal{A}$  was too big so that MiniSat 2.2.0 could not handle the resulting SAT instance. In such cases, we had to use some additional ideas to distinguish between non-synchronizing and “too slowly” synchronizing automata.

For exact synchronization, an additional analysis was needed only for small values of  $n$  ( $n \leq 20$ ) and for a few exceptional PFAs with  $n > 30$ . We analyzed these cases using a brute force algorithm known as the successor tree method. See the recent paper by Türker [86] for a description of the method and its modern implementation<sup>1</sup>.

The situation for careful synchronization was more involved. The

---

<sup>1</sup>Notice that Türker [86] uses the term “reset sequence” for what we call “exactly synchronizing word”.

only known brute force algorithm for careful synchronization is the partial power automaton method, which we will discuss (and compare with our approach) in Section 3.6. It turned out that this method could hardly handle PFAs with more than 20 states. Therefore, we devised a simple theoretical condition under which a binary PFA is not carefully synchronizing and checked PFAs against this condition, prior to having started the procedures from Section 3.1.

Let  $q$  be a state and  $a$  letter of a PFA. We say that  $q$  is *a-cyclic* if  $q = q.a^k$  for some positive integer  $k$ .

**Lemma 3.2.** *Let a PFA  $\mathcal{A} = (Q, \{a, b\})$  be such that the letter  $a$  is everywhere defined and has at least two  $a$ -cyclic states. If the letter  $b$  is undefined at some  $a$ -cyclic state,  $\mathcal{A}$  is not carefully synchronizing.*

*Proof.* Arguing by contradiction, suppose that  $w \in \{a, b\}^*$  is a carefully synchronizing word for  $\mathcal{A}$ . Then  $w$  starts with  $a$  because of the condition (C1). Further,  $w$  cannot be a power of  $a$  because of the condition (C3) as  $a$  has at least two  $a$ -cyclic states and each  $a$ -cyclic state belongs to the image of an arbitrary power of  $a$ . Thus, the letter  $b$  occurs in  $w$  whence  $w$  has a prefix of the form  $a^s b$  for some positive integer  $s$ . As mentioned, each  $a$ -cyclic state belongs to  $Q.a^s$ , and we get a contradiction with the condition (C2) as  $b$  is undefined at some  $a$ -cyclic state.  $\square$

Clearly, given a binary PFA, it is easy to verify if the PFA satisfies the premises of Lemma 3.2. It is Lemma 3.2 that we used to filter out almost complete PFAs that were not carefully synchronizing before having run the SAT-solver method. We stress that Lemma 3.2 is only a sufficient condition for an almost complete PFA to be not carefully synchronizing. However, it was well suited for our purposes because it turned out to

be applicable frequently enough. Indeed, the statistical properties of random maps are well studied; in particular, if the random variable  $\xi$  represents the number of cyclic points of a map chosen uniformly at random from all  $n^n$  maps on an  $n$ -element set, the following expression for the probability of the event  $\xi = j$ , where  $j \in \{1, 2, \dots, n\}$ , is known (see [33]):

$$P(\xi = j) = \frac{(n-1)!j}{(n-j)!n^j}. \quad (3.1)$$

For the premises of Lemma 3.2 to hold for an almost complete PFA  $\mathcal{A} = (Q, \{a, b\}, \delta)$ , the map  $Q \rightarrow Q$  induced by the letter  $a$  must have at least two cyclic points (=  $a$ -cyclic states), and the only state at which the letter  $b$  is undefined must be  $a$ -cyclic. Denoting  $|Q|$  by  $n$ , we derive from (3.1) the following expression for the probability that Lemma 3.2 applies to  $\mathcal{A}$ :

$$\sum_{j=2}^{n-1} \frac{j}{n} P(\xi = j) = \sum_{j=2}^{n-1} \frac{(n-1)!j^2}{(n-j)!n^{j+1}}. \quad (3.2)$$

Observe that the expression (3.2) differs in just one summand, namely, in  $\frac{1}{n}P(\xi = 1) = \frac{1}{n^2}$ , from

$$n^{-1}E[\xi] = \sum_{j=1}^n \frac{j}{n} P(\xi = j).$$

Evaluating the expression (3.2) at  $n = 100$ , one gets 0.121989414. (For these numerical computations, we used an elegant method suggested by Zubkov [91].) Thus, more than 12% of randomly chosen almost complete PFAs with 100 states satisfy the premises of Lemma 3.2. On the other hand, the SAT-solver approach in our experiments succeeded for

more than 87% of almost complete PFAs with 100 states. It is what we meant above when having said that Lemma 3.2 was well sufficient to confirm the absence of careful synchronization for an overwhelming majority of almost complete PFAs which are not carefully synchronizing, and thus, to avoid the SAT-solver having to work in vain.

Back to the aforementioned question of the asymptotic behavior of the function  $P_C(n)$ , we notice that even though Lemma 3.2 does not exclude  $P_C(n)$  tending to 1, it allows us to show that even if  $P_C(n)$  tends to 1 as  $n \rightarrow \infty$ , the convergence rate should be relatively slow. Indeed, it is known (see [33]) that the expectation  $E[\xi]$  is asymptotically equivalent to  $\sqrt{\frac{\pi n}{2}}$ . As observed, the probability (3.2) that Lemma 3.2 applies to a random almost complete PFAs with  $n$  states differs from  $n^{-1}E[\xi] \sim \sqrt{\frac{\pi}{2n}}$  by  $\frac{1}{n^2}$ , which is asymptotically negligible in comparison with  $\sqrt{\frac{\pi}{2n}}$ . By Lemma 3.2, we have that the difference  $1 - P_C(n)$ , that is, the probability that an almost complete PFAs with  $n$  states is not carefully synchronizing is asymptotically greater than or equivalent to  $\sqrt{\frac{\pi}{2n}}$ . Thus,  $1 - P_C(n) = \Omega(\frac{1}{\sqrt{n}})$ , while we have demonstrated above that  $1 - P_E(n) = O(\frac{1}{n})$ .

### 3.4.2 Series 2: Average length of the shortest synchronizing word

We have applied the encoding constructed in Sections 2.1 and 2.2 to solve with the help of a SAT solver CSW and ESW instances respectively. We worked with almost complete PFAs.

For CSW, we worked with such PFAs that were found to be carefully synchronizing in the course of the experiment detailed in Subsection. 3.4.1. For such a PFA  $\mathcal{A}$ , we were left at the end of the experiment with a number  $\ell$ , the least power of 2 for which MiniSat returns YES on the SAT instance that encodes the CSW instance  $(\mathcal{A}, \ell)$ . In order to find a carefully synchronizing word or exactly synchronizing word of minimum length for  $\mathcal{A}$ , we performed standard binary search, having started with  $\ell_{\max} := \ell$  and  $\ell_{\min} := \frac{\ell}{2}$ . That is, we

- 1) let  $\ell := \frac{\ell_{\min} + \ell_{\max}}{2}$ ;
- 2) run MiniSat on the SAT instance that encodes the CSW instance  $(\mathcal{A}, \ell)$ ;
- 3) let  $\ell_{\max} := \ell$  if the answer returned by Minisat was YES, and let  $\ell_{\min} := \ell$  if the answer was NO;
- 4) check if  $\ell_{\max} - \ell_{\min} = 1$ : YES means that  $\ell_{\max}$  is the minimum length of carefully synchronizing words for  $\mathcal{A}$ ; NO means that we have to return to Step 1).

In the following we define the length of the shortest synchronizing word as the *reset threshold* ( $RT$ ). By  $RCT$  we define the length of the shortest carefully synchronizing word and by  $RET$  we define the length of the shortest exactly synchronizing word. Using experimental data found this way, we calculated the average for  $RCT$  of carefully synchronizing almost complete PFAs with  $n$  states and define this average by  $A_C(n)$ . Then we used the least squares method to find a function that best

reflects how  $A_C(n)$  depends on  $n$ . It turned out that our results are reasonably well approximated by the following expression:

$$A_C(n) \approx 3.92 + 0.49n - 0.005n^2 + 0.000024n^3. \quad (3.3)$$

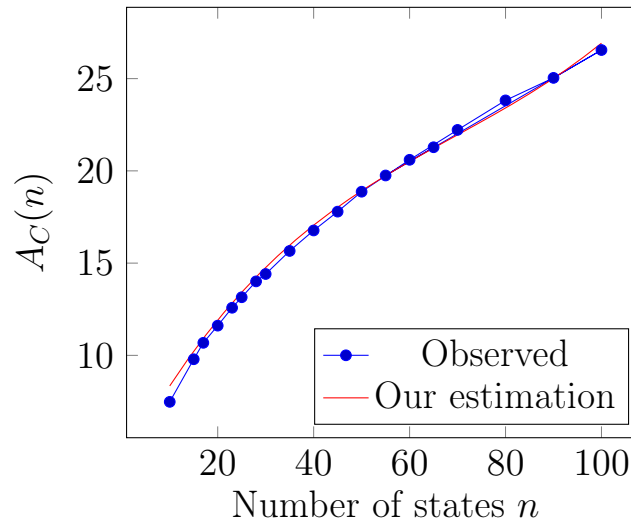


Figure 3.4: Approximation of  $A_C(n)$

The relation between the approximation (3.3) and our experimental data is shown in Figure 3.4, while Figure 3.5 shows the relation between the relative standard deviation (r.s.d) of our datasets and the number of states. We see that the relative standard deviation gradually decreases as the number of states grows.

We have followed the same strategy to find an approximation of the average length  $A_E(n)$  of the shortest exactly synchronizing word for almost complete PFAs. The results yield that  $A_E(n)$  turns out to be smaller than  $A_C(n)$ . However, both values seem to follow the same pattern. These observations are illustrated in Figure 3.6 where  $A(n)$  stands

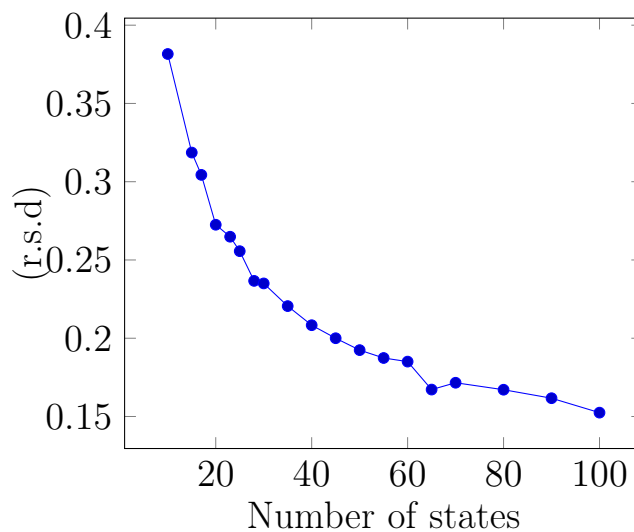


Figure 3.5: Relative standard deviation of datasets

for the average length of shortest synchronizing word for randomly generated almost complete PFAs.

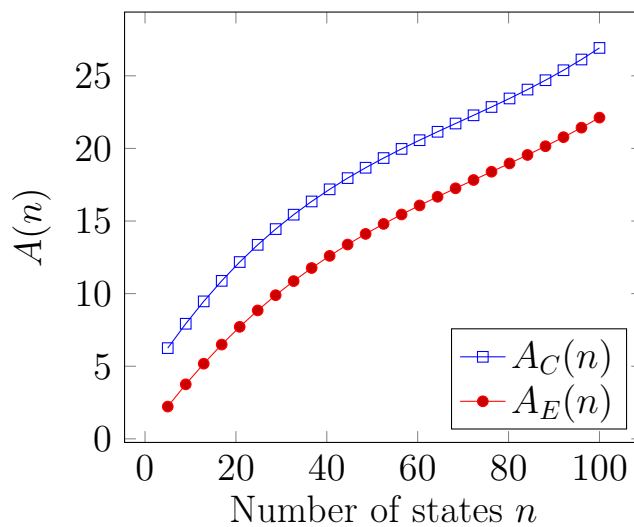


Figure 3.6: Comparison between  $A_C(n)$  and  $A_E(n)$

### 3.4.3 Series 3: Influence of the input alphabet size

This series of experiments aimed to see how the length of the shortest carefully synchronizing word is affected by the number of input letters. We experimented with samples of carefully synchronizing PFAs with varying state and input alphabet sizes but approximately the same *relative density*, that is, the same ratio between the density and the number of states. We generated random PFAs as described in Section 3.3 and applied Lemma 3.2 for filtering out PFAs that were not carefully synchronizing. Then we used binary search as in Subsection 3.4.2 to determine the minimum length of carefully synchronizing words.

Figure 3.7 may serve as an illustration for typical results found in this series of experiments. It shows the average lengths of shortest carefully synchronizing words for carefully synchronizing PFAs with 2 or 3 input letters and relative density close to 2. More precisely, we considered PFAs with  $n$  states and the density  $\rho = 2n - 1$ . (Thus, in the case of 2 input letters, we dealt with almost complete PFAs so that we were in a position to partly re-use the data computed in experiments in Subsection 3.4.2.) We see that the corresponding graphs have similar regular shape and that PFAs with a larger input alphabet synchronize faster. These conclusions held also when other values of relative density were fixed. The observed phenomena are intuitively plausible as having more letters gives more degrees of freedom for careful synchronization and it is to expect that carefully synchronizing words become shorter. However, we have got no rigorous theoretical explanations for these phenomena so far.



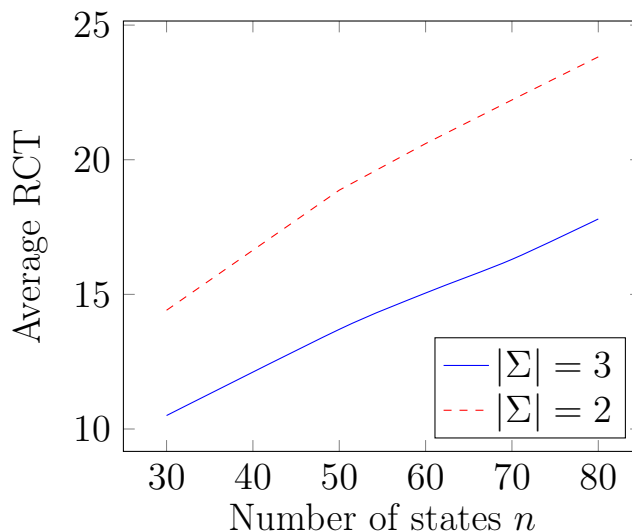


Figure 3.7: The cardinality of the input alphabet versus the average RCT when  $\rho = 2n - 1$

### 3.4.4 Series 4: Influence of density

In this series, we fixed two parameters  $n$  and  $\rho \leq 2n - 1$ . For pairs  $(n, \rho)$  such that  $\rho \geq n + 1$ , we generated a sample of random binary PFAs with  $n$  states, density  $\rho$ , and an everywhere defined letter as described in Section 3.3. Then we computed the average length of shortest carefully synchronizing words for PFAs in this sample, having used the same procedure as above, that is, the pre-selection based on Lemma 3.2 followed by binary search as described in Subsection 3.4.2. Similarly, for pairs  $(n, \rho)$  with  $\rho \geq 2$ , we prepared a sample of random binary PFAs with  $n$  states and density  $\rho$ , and then we computed the average length of shortest exactly synchronizing words for these PFAs. Dealing with shortest exactly synchronizing words was slightly more involved. The complication was due to the fact that, in the absence of an everywhere defined

letter, a PFA having an exactly synchronizing word of some length, may have no exactly synchronizing word of any larger length. In fact, such situations occur quite often for PFAs of low density. Due to this subtlety, binary search could not be used, and therefore, we were forced to check, for each PFA  $\mathcal{A}$  in our sample, the SAT instances that encoded the ESW instances  $(\mathcal{A}, 1)$ ,  $(\mathcal{A}, 2)$ ,  $(\mathcal{A}, 3)$ , etc.

Our experiments showed that the average length of the shortest exactly synchronizing word increased as the density increased. This strongly contrasts the case of careful synchronization where the results were opposite: the more the density was, the less was the average length of shortest carefully synchronizing word. Figures 3.8 and 3.9 illustrate these observations.

When an automaton becomes complete, its carefully and exactly synchronizing words become nothing but classical synchronizing words of the complete case. Therefore, it is natural to expect that, when  $\rho$  approaches  $2n$ , the average lengths of both carefully and exactly synchronizing words for synchronizing binary PFAs with  $n$  states tend to the average length of synchronizing words for synchronizing binary CFAs with  $n$  states. The latter length has been evaluated by Kisielewicz, Kowalski, and Szykuła in [43] as a result of a series of massive experiments. Namely, the average length of synchronizing word for synchronizing binary CFAs with  $n$  states is approximately equal to  $2.5\sqrt{n-5}$ . If one looks at the graphs in Figures 3.8 and 3.9, one may observe that they match the expectation above. Indeed, the expression  $2.5\sqrt{n-5}$  gives 12.5 for  $n = 30$  and approximately 21.65 for  $n = 80$ . Extrapolating the graphs in Figures 3.8 and 3.9 to the right, one gets very close values for the ordinates that would correspond to  $\rho = 60$  and respectively  $\rho = 160$ .

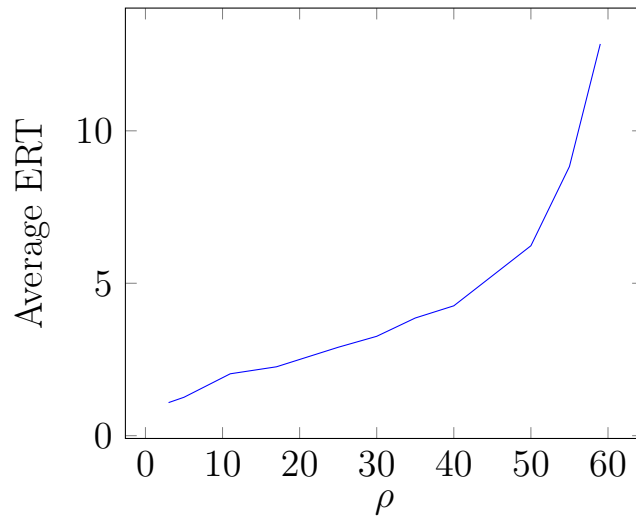


Figure 3.8: Influence of density on the average RET for 30 states

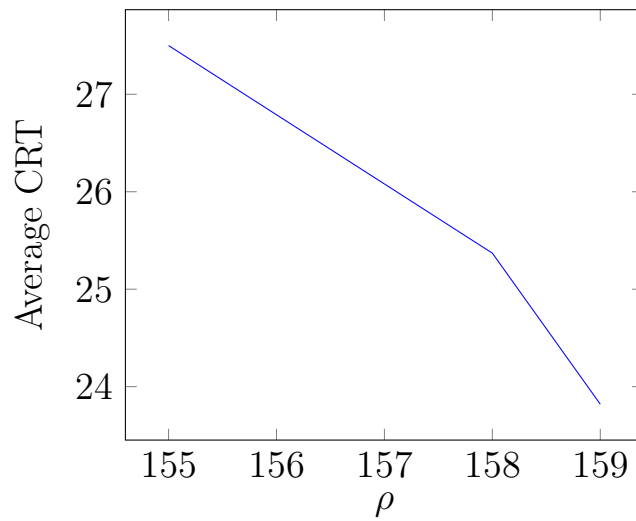


Figure 3.9: Influence of density on the average RCT for 80 states

The same behaviour was observed in our experiments with PFAs of other sizes.

### 3.5 Slowly synchronizing automata and benchmarks

Besides experimenting with randomly generated PFAs, we have tested our approach on certain provably “slowly synchronizing” automata, that is, the ones with the minimum length of carefully synchronizing words close to of the state number squared.

We restrict ourselves to almost complete PFAs in the sense of Subsection 3.4.1; recall that these are binary PFAs with only one undefined transitions. De Bondt, Don, and Zantema [18, Theorem 17] have proved that for any sufficiently large  $n$  divisible by 10, there exists an almost complete PFA with  $n$  states whose shortest carefully synchronizing word length is  $\Omega(2^{\frac{n}{5}})$ . This remarkable result has been obtained by a series of non-trivial constructions, built one of the top of others, so that it is very difficult to estimate the constant behind the  $\Omega$ -notation, to say nothing of exhibiting any such PFA in an explicit form. Therefore we could not test our method on these PFAs.

Fortunately, the same paper [18] provides also the following explicit series of slowly synchronizing almost complete PFAs. For each  $n \geq 3$ , let  $\mathcal{P}_n$  stand for the PFA with the state set  $\{1, 2, \dots, n\}$ , on which the input letters  $a$  and  $b$  act as follows:

$$q.a := \begin{cases} q + 1 & \text{if } q = 1, 2, \\ q & \text{if } 3 \leq q \leq n; \end{cases} \quad q.b := \begin{cases} \text{undefined} & \text{if } q = 1, \\ q + 1 & \text{if } 2 \leq q \leq n - 1, \\ 1 & \text{if } q = n. \end{cases}$$

The automaton  $\mathcal{P}_n$  with  $n \geq 4$  is shown in Figure 3.10.

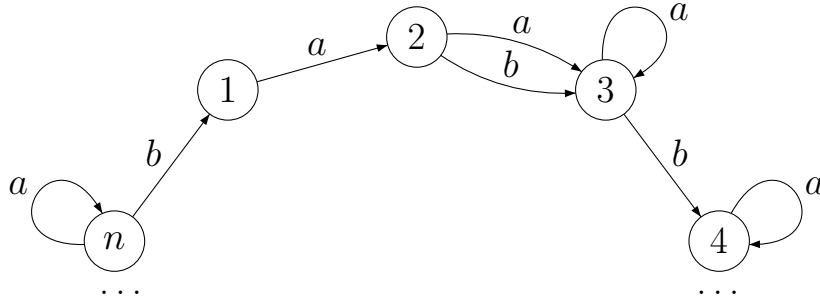


Figure 3.10: The automaton  $\mathcal{P}_n$

Recall that the classic sequence  $\text{fib}(m)$  of the Fibonacci numbers is defined by the recurrence  $\text{fib}(m) = \text{fib}(m - 1) + \text{fib}(m - 2)$  for  $m \geq 2$ , together with the initial condition  $\text{fib}(0) = 0$ ,  $\text{fib}(1) = 1$ . The following result is stated in [18] without proof:

**Proposition 3.2.** *For  $n \geq 3$ , let  $m$  be a unique integer that satisfies the double inequality  $\text{fib}(m - 1) < n - 2 \leq \text{fib}(m)$ . The shortest carefully synchronizing word for the automaton  $\mathcal{P}_n$  has length*

$$n^2 + mn - 5n - \text{fib}(m + 1) - 2m + 8.$$

We applied our algorithm to automata  $\mathcal{P}_n$  with  $n = 4, 5, \dots, 12$ , and for each of them, our result matched the value predicted in Proposition 3.2. The time consumed ranged from 0.301 sec for  $n = 4$  to 14164 sec for  $n = 12$ . Observe that in the latter case the shortest carefully synchronizing word has length 141 so that the “honest” binary search started with  $(\mathcal{P}_{12}, 1)$  required 16 calls of MiniSat, namely, for the encodings of  $(\mathcal{P}_{12}, \ell)$  with  $\ell = 1, 2, 4, 8, 16, 32, 64, 128, 256, 192, 160, 144, 136, 140, 142, 141$ . (Of course, if one just wants to confirm (or to disprove) a theoretical prediction  $\ell$  for the minimum length of carefully syn-

chronizing words for a given PFA  $\mathcal{A}$ , two calls of a SAT solver suffice—on the encodings of the CSW instances  $(\mathcal{A}, \ell)$  and  $(\mathcal{A}, \ell - 1)$ .)

In our experiments, we kept track of PFAs with the minimum length of carefully synchronizing words close to the square of the number of states. Whenever we encountered such examples, we made an attempt to generalize them in order to get infinite series of provably “slowly synchronizing” PFAs. We present here two of the series we found this way.

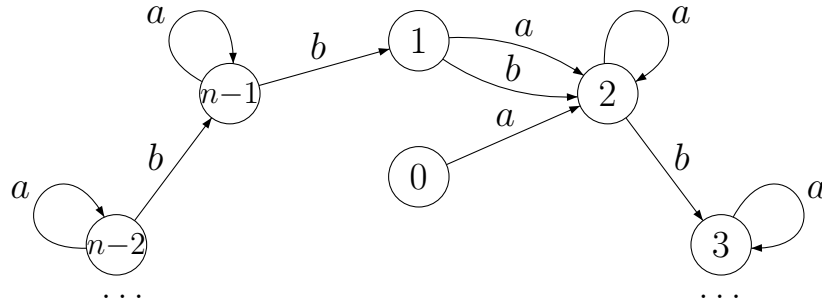


Figure 3.11: The automaton  $\mathcal{H}'_n$

For  $n > 4$ , let  $\mathcal{H}'_n$  be the PFA with the state set  $\{0, 1, \dots, n - 1\}$  on which the input letters  $a$  and  $b$  act as follows:

$$q.a := \begin{cases} 2 & \text{if } q \leq 2, \\ q & \text{if } q \geq 3; \end{cases} \quad q.b := \begin{cases} \text{undefined} & \text{if } q = 0, \\ q + 1 & \text{if } 0 < q < n - 1, \\ 1 & \text{if } q = n - 1. \end{cases}$$

The automaton  $\mathcal{H}'_n$  is shown in Figure 3.11. The reader acquainted with the theory of complete synchronizing automata immediately recognizes that the subautomaton induced by the action of  $a$  and  $b$  on the set  $\{1, \dots, n - 1\}$  is exactly the  $(n - 1)$ -state automaton  $\mathcal{C}_{n-1}$  from

the famous series discovered by Černý [15], see Figure 1.6. Clearly, if a PFA  $\mathcal{A}$  has a subautomaton  $\mathcal{B}$ , then every carefully synchronizing word for  $\mathcal{A}$  (if exists) also serves as a carefully synchronizing word for  $\mathcal{B}$ . Hence, every carefully synchronizing word for  $\mathcal{H}'_n$  (if exists) must be a synchronizing word for the complete subautomaton  $\mathcal{C}_{n-1}$ . It follows from [15, Lemma 1], see also [3, Theorem 3] for an easy alternative proof, that the shortest synchronizing word for  $\mathcal{C}_{n-1}$  is the word  $w := (ab^{n-2})^{n-3}a$  of length  $(n-2)^2$  which brings every state of the subautomaton to the state 2. Hence no carefully synchronizing word for  $\mathcal{H}'_n$  can be shorter than  $w$ . On the other hand, one can readily compute that  $0.w = 2$  as well, whence  $w$  is a carefully synchronizing word for the whole automaton  $\mathcal{H}'_n$ . We have thus established

**Proposition 3.3.** *The automaton  $\mathcal{H}'_n$  is carefully synchronizing and the minimum length of carefully synchronizing words for  $\mathcal{H}'_n$  is equal to  $(n-2)^2$ .*

For  $n > 4$ , let  $\mathcal{H}''_n$  be the PFA with the state set  $\{0, 1, \dots, n-1\}$  on which the input letters  $a$  and  $b$  act as follows:

$$q.a := \begin{cases} q+1 & \text{if } q \leq n-2, \\ 1 & \text{if } q = n-1; \end{cases} \quad q.b := \begin{cases} \text{undefined} & \text{if } q = 0, \\ q+1 \pmod{n} & \text{if } q \geq 1. \end{cases}$$

The automaton  $\mathcal{H}''_n$  is shown in Figure 3.12. We observe that the automata  $\mathcal{H}''_n$  are closely related to the so-called Wielandt automata  $\mathcal{W}_n$  which plays a role in the theory of complete synchronizing automata; see [3, Theorem 2]. Namely,  $\mathcal{W}_n$  is just  $\mathcal{H}''_n$  with the transition  $0 \xrightarrow{b} 1$  added.

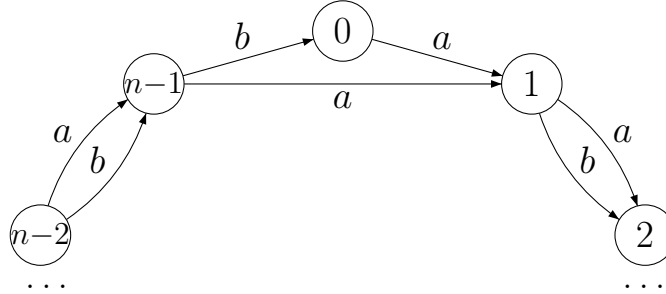


Figure 3.12: The automaton  $\mathcal{H}_n''$

**Proposition 3.4.** *The automaton  $\mathcal{H}_n''$  is carefully synchronizing and the minimum length of carefully synchronizing words for  $\mathcal{H}_n''$  is equal to  $n^2 - 3n + 3$ .*

*Proof.* We use a suitable adaptation of the argument developed in [3] for studying slowly synchronizing DFAs.

Suppose that  $\mathcal{H}_n''$  is carefully synchronizing and let  $w$  be its carefully synchronizing word of minimum length. Then  $w$  must bring the automaton to the state 1; otherwise, removing from  $w$  its last letter would yield a shorter carefully synchronizing word. Since the letter  $a$  is everywhere defined, for every positive integer  $i$ , the word  $a^i w$  also brings  $\mathcal{H}_n''$  to the state 1. In particular,  $1.a^i w = 1$ , that is,  $a^i w$  labels a cycle in the underlying digraph of  $\mathcal{H}_n''$ . Therefore, for every  $\ell \geq |w|$ , there is a cycle of length  $\ell$  in  $\mathcal{H}_n''$ . The underlying digraph of  $\mathcal{H}_n''$  has simple cycles only of two lengths:  $n$  and  $n - 1$ . Each cycle of the digraph must consist of simple cycles of these two lengths, whence each number  $\ell \geq |w|$  must be expressible as a non-negative integer combination of  $n$  and  $n - 1$ . Here we invoke the following well-known and elementary result from number theory:



**Lemma 3.3** ([72, Theorem 2.1.1]). *If  $k_1, k_2$  are relatively prime positive integers, then  $k_1k_2 - k_1 - k_2$  is the largest integer that is not expressible as a non-negative integer combination of  $k_1$  and  $k_2$ .*

Lemma 3.3 implies that  $|w| > n(n-1) - n - (n-1) = n^2 - 3n + 1$ . Suppose that  $|w| = n^2 - 3n + 2$ . Since  $0.w = 1$ , there should be a path of this length the state 0 to the state 1. The only letter defined at 0 is the letter  $a$ , whence  $w = av$  for some  $v$ . Since  $0.a = 1$ , we have  $1.v = 1$  so that the word  $v$  labels a cycle in  $\mathcal{H}_n''$ . However, the length of  $v$  is  $n^2 - 3n + 1 = n(n-1) - n - (n-1)$  and Lemma 3.3. no cycles of this length may exist in the digraph of  $\mathcal{H}_n''$ , a contradiction. Hence,  $|w| \geq n^2 - 3n + 3$ . On the other hand, it can be readily verified that the word  $(aba^{n-2})^{n-3}aba$  of length  $n(n-3) + 3 = n^2 - 3n + 3$  carefully synchronizes the automaton  $\mathcal{H}_n''$ . Hence  $\mathcal{H}_n''$  is carefully synchronizing, and  $n^2 - 3n + 3$  is the minimum length of its carefully synchronizing words. □

From the viewpoint of our studies, the series  $\mathcal{H}_n'$  and  $\mathcal{H}_n''$  are of interest as they exhibit two extremes with respect to amenability of careful synchronization to the SAT-solver approach. The series  $\mathcal{H}_n'$  is turned to be a hard nut to crack for our algorithm: the maximum  $n$  for which the algorithm was able to find a carefully synchronizing word of minimum length is 13, and computing this word (of length 121) took almost 4 hours. In contrast, automata in the series  $\mathcal{H}_n''$  turn out to be quite amenable: for instance, our algorithm found a carefully synchronizing word of length 343 for  $\mathcal{H}_{20}''$  in 13.38 sec. At present, we have no explanation for what causes such a strong contrast: is this an intrinsic structure of the PFAs under consideration, or the nature of the algorithm build in MiniSat, or just a peculiarity of our implementation?

### 3.6 A comparison with the partial power automaton method

We made a comparison between our approach and the only method for computing carefully synchronizing words of minimum length that we had found in the literature, namely, the method based on partial power automata; see [60, p. 295]. Given a PFA  $\mathcal{A} = (Q, \Sigma, \delta)$ , its *partial power automaton*  $\mathcal{P}(\mathcal{A})$  has the non-empty subsets of  $Q$  as the states, the same input alphabet  $\Sigma$ , and the transition function defined as follows: for each  $a \in \Sigma$  and each  $P \subseteq Q$ ,

$$P.a := \begin{cases} \{q.a \mid q \in P\} & \text{provided } q.a \text{ is defined for all } q \in P, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Recall that this definition differs from the one we introduced in (1.1) in Section 1.4 where  $P.a$  was defined just as  $P.a := \{q.a \mid q \in P\}$ . Under that definition, a letter  $a$  was **undefined** at  $P$  if and only if  $a$  was **undefined** at every  $q \in P$ . Here, in contrast,  $a$  is **defined** at  $P$  if and only if  $a$  is **defined** at every  $q \in P$ . For an illustration, Figures 3.13 and 3.14 show a PFA and its partial power automaton.

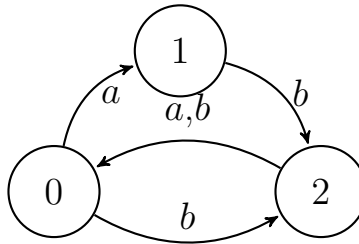


Figure 3.13: A carefully synchronizing PFA

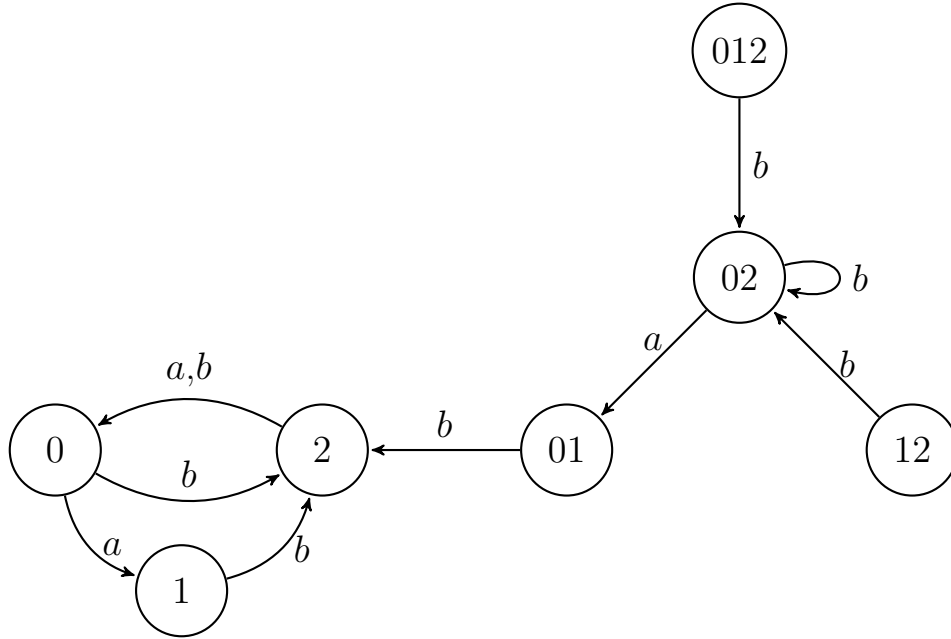


Figure 3.14: Partial power automaton of the PFA in Figure 3.13

It is easy to see that  $w \in \Sigma^*$  is a carefully synchronizing word of minimum length for  $\mathcal{A}$  if and only if  $w$  labels a minimum length path in  $\mathcal{P}(\mathcal{A})$  starting at  $Q$  and ending at a singleton. Such a path can be found by breadth-first search in the underlying digraph of  $\mathcal{P}(\mathcal{A})$ .

We implemented the partial power automaton method and ran it on our samples of random PFAs. Figure 3.15 presents the results of the comparison. In this experiment we had to restrict to PFAs with at most 16 states since beyond this size of states, our implementation of the method based on partial power automata could not complete the computation due to memory restrictions (recall that we used rather modest computational resources). However, we think that the exhibited data suffice to demonstrate that the SAT-solver approach performs by far better.

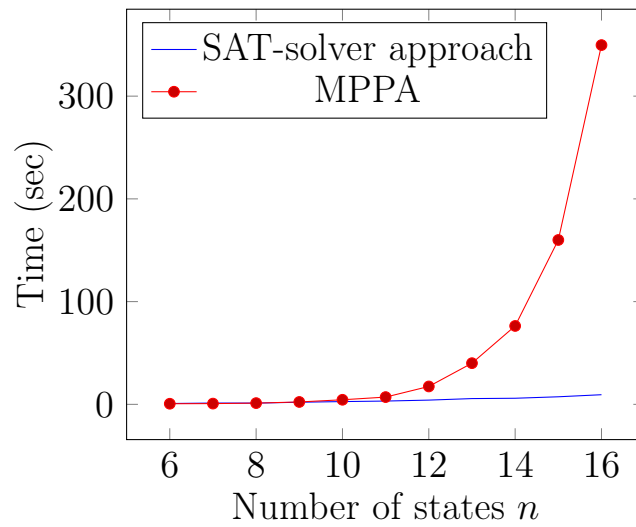


Figure 3.15: Comparison between the partial power automaton method and the SAT-solver approach

## Chapter 4

# Synchronization problems of NFAs

In this chapter we are concerned with modeling  $D_i$ -synchronization problems of a general NFA as SAT problems. Each one of these problems requires its own encoding. However, the three encodings share the same core: a part of clauses that encode the computation of a given NFA. We first present this common part and then proceed with parts that are specific for each version of synchronization.

### 4.1 Modeling NFA computation as SAT: Variables

Given an NFA  $\mathcal{A}$  and an integer  $\ell$ , we will construct, in polynomial time with respect to the size of  $(\mathcal{A}, \ell)$ , an instance  $(V, C)$  of SAT that simulates the computation of a word of length  $\ell$  in  $\mathcal{A}$ . Following the

style adopted in Chapter 2, in the presentation of our modeling, precise definitions and statements are interwoven with less formal comments explaining the “physical” meaning of variables and clauses we introduce and with estimations of their numbers.

Take a NFA  $\mathcal{A} = (Q, \Sigma, \delta)$  and an integer  $\ell > 0$ . Denote the size of  $Q$  by  $n$  and fix some numbering of the states in  $Q$  so that  $Q = \{q_1, \dots, q_n\}$ . Let  $\Sigma = \{0, 1\}$ , we restrict our selves to the case of binary NFAs.

We start with introducing the variables used in the instance  $(V, C)$  of SAT that encodes  $(\mathcal{A}, \ell)$ .

The set  $V$  consists of two sorts of variables:

1. *Letter variables*
2. *Token variables*

The letter variables are  $x_1, \dots, x_\ell$ . They are just placeholders for the input symbols 0 and 1. There is an obvious 1-1 correspondence between the truth assignments on the set  $X = \{x_1, \dots, x_\ell\}$  and the words in  $\Sigma^\ell$ : given a truth assignment  $\tau: X \rightarrow \{0, 1\}$ , the corresponding word is  $\tau(x_1) \cdots \tau(x_\ell)$ , and, conversely, given a word  $a_1 \cdots a_\ell$  with  $a_1, \dots, a_\ell \in \{0, 1\}$ , the corresponding truth assignment is  $x_t \mapsto a_t$  for each  $t = 1, \dots, \ell$ .

The role of the letters variables is exactly the same as for PFAs. In contrast, we need some new sort of variables in order to reflect the non-determinism. When an input word is applied at specified state of an NFA  $\mathcal{A}$  there may be several states to which  $\mathcal{A}$  may transfer. Given an input word  $w$  and an NFA  $\mathcal{A} = (Q, \Sigma, \delta)$ , the significant parameter in verifying whether or not  $w$  is a  $D_i$ -synchronizing word for  $\mathcal{A}$  is the relation between all the sets  $\delta(q, w)$  for each  $q \in Q$ . In contrast for PFAs

the significant parameter of synchronization is the set of active states at the end of the application of the word  $w$ . Hence the state variables used in encoding of PFAs may be not sufficient in  $D_i$ -synchronization. We introduce new sort of variables called *token variables*.

The token variables are  $y_{ij}^t$  where  $i, j \in \{1, \dots, n\}$  and  $t = 0, 1, \dots, \ell$ . To explain the role of these variables, we use a solitaire-like game  $\Gamma$  on the underlying directed graph representing the NFA  $\mathcal{A}$ .

In the initial position of  $\Gamma$ , each state  $q_i \in Q$  holds exactly one token denoted  $\mathbf{i}$ .

In the course of the game, tokens migrate and may multiply or disappear according to certain rules that will be specified a bit later, when we describe the clauses in  $C$ .

For the moment, it is sufficient to say that the rules are designed to ensure that the variable  $y_{ij}^t$  gets value 1 in a satisfying truth assignment for  $C$  if and only if after  $t$  rounds of the game, one of the tokens held by the state  $q_j$  is  $\mathbf{i}$ .

## 4.2 Modeling NFA computation as SAT: Clauses

The computation of the input word with length  $\ell$  in  $\mathcal{A}$  will be read from the set of clauses  $C$ . It is the disjoint union of  $\ell + 1$  sets:

1. The set  $C_0$  of *Initial clauses*
2. The sets  $C_t$ ,  $t = 1, \dots, \ell$ , of *Transition clauses*
3. The set  $S$  of *Synchronization clauses*.

The clauses in  $C_0$  describes the automaton at the initial position (no word has been applied). Hence they encode the initial position of our game  $\Gamma$ . As mentioned, in this position, each state  $q_i \in Q$  holds the token  $\mathbf{i}$  and nothing else. In order to reflect this setting, we let  $C_0$  consist of the clauses:

$$y_{11}^0, \dots, y_{nn}^0 \tag{4.1}$$

along with all clauses of the form

$$\neg y_{ij}^0; \quad i \neq j \tag{4.2}$$

Altogether the set of clauses in (4.1) with the set of clauses in (4.2), the initial clauses  $C_0$  contains  $n^2$  one-literal clauses. Recall that in encoding for PFAs, the set of initial clauses consisted of just  $n$  clauses.

The transition clauses are the clauses that encode the rules of  $\Gamma$ . In the course of the game, tokens migrate and may multiply or disappear according to the previous position of the game and the action of the player. So the transition clauses set is the set of clauses  $C_t$ ,  $t = 1, \dots, \ell$ . The token status is determined according the following rules:

1. At each move an input symbol  $a \in \Sigma$  is chosen.
2. For each state  $q \in Q$  such that  $q.a \neq \emptyset$ , all tokens that were held by  $q$  slide along the edges labeled  $a$  to all states in the set  $q.a$ .
3. If  $|q.a| > 1$ , then every token held by  $q$  multiplies to  $|q.a|$  identical tokens, one for each state in  $q.a$ .
4. If  $q.a = \emptyset$ , then all tokens that were held by  $q$  disappear.

From these rules we have a guarantee that, after the move, the token



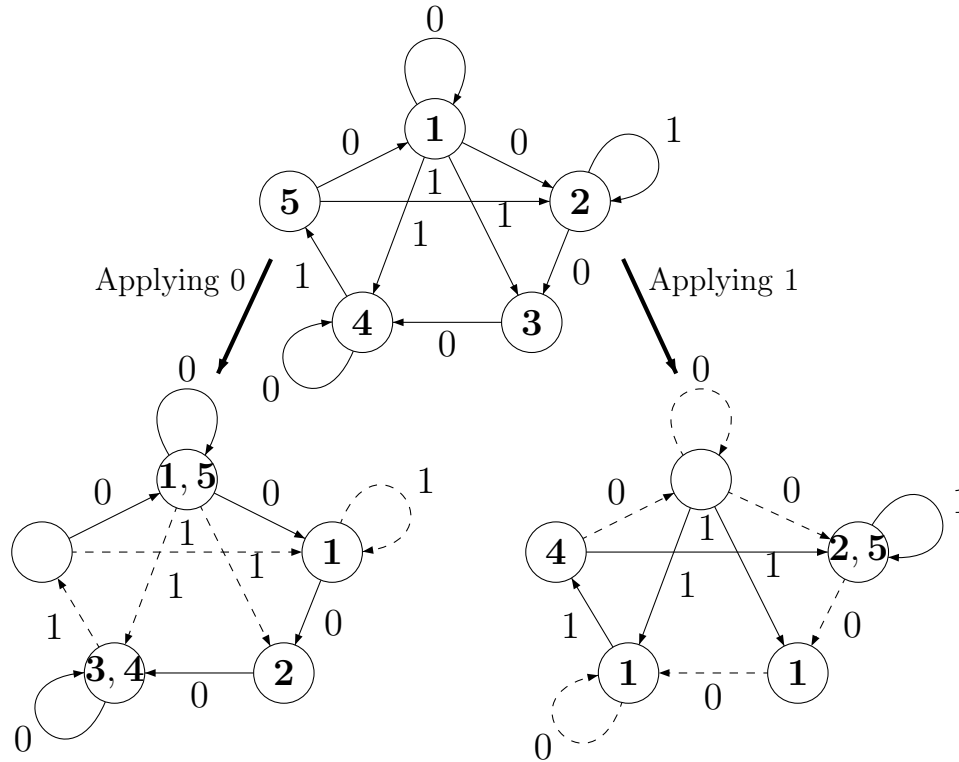


Figure 4.1: Redistribution of tokens after the first move

$\mathbf{i}$  occurs at a state  $p \in Q$  if and only if  $p \in q.a$  for some state  $q$  that had held  $\mathbf{i}$  just prior to the move.

For an illustration, Figure 4.1 demonstrates the initial distribution of tokens on a 5-state NFA with the input alphabet  $\{0, 1\}$  (top), along with the outcomes of the first move, depending on whether 0 or 1 has been chosen for the move (bottom left and bottom right, respectively).

Perhaps, it makes sense to add a matrix interpretation of the game  $\Gamma$  as the token variables get quite a clear meaning under this interpretation. The initial position of  $\Gamma$  can be thought of as the identity Boolean  $Q \times Q$ -matrix. At each move, an input symbol  $a \in \Sigma$  is chosen and the

matrix of the current position is right multiplied by the matrix  $M(a)$  see Subsection 1.5 for matrix representation of NFAs. Then for each fixed  $t$ , the values of the variables  $y_{ij}^t$  are exactly the entries of the matrix corresponding to the position of  $\Gamma$  after  $t$  moves. For instance, the matrices that correspond to two possible positions of the game played on the 5-state NFA in Fig. 4.1 are respectively

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

From that encoding, the following observation is immediate.

**Lemma 4.1.** *Suppose that in the game  $\Gamma$  played on  $\mathcal{A} = (Q, \Sigma, \delta)$ , the sequence of chosen symbols forms a word  $w \in \Sigma^*$ . Then for each  $i = 1, \dots, n$ , the set of states holding the token  $\mathbf{i}$  at the end of the game is  $q_i.w$ .*

### 4.3 Propositional logic formulas for rules

This section expresses the rules of  $\Gamma$  by formulas of propositional logic.

Recall from Section 2.2, that for a state  $q \in Q$ ,  $P_0(q)$  and  $P_1(q)$  stand for the sets of all preimages of  $q$  under the actions of the input symbols 0 and respectively 1, that is, if  $a$  is either of the two symbols,

$$P_a(q) = \{p \in Q \mid q \in p.a\} \tag{4.3}$$

Consider for every  $t = 1, \dots, \ell$  and all  $i, j = 1, \dots, n$ , the following formulas:

$$\Psi_{ij}^t : y_{ij}^t \iff \left( x_t \wedge \bigvee_{q_k \in P_1(q_j)} y_{ik}^{t-1} \right) \vee \left( \neg x_t \wedge \bigvee_{q_h \in P_0(q_j)} y_{ih}^{t-1} \right) \quad (4.4)$$

Observe that the equivalence  $\Psi_{ij}^t$  just translates in the language of propositional logic our propagation rule for the tokens that says that the token  $\mathbf{i}$  occurs at the state  $q_j$  after  $t$  moves if and only if one of the following alternatives takes place:

- the  $t$ -th move was done with the input symbol 1 and one of the preimages of  $q_j$  under the actions of 1 was holding  $\mathbf{i}$  after  $t - 1$  moves, or
- the  $t$ -th move was done with the input symbol 0 and one of the preimages of  $q_j$  under the actions of 0 was holding  $\mathbf{i}$  after  $t - 1$  moves.

**Lemma 4.2.** *For every  $t = 0, 1, \dots, \ell$ , every truth assignment  $\tau$  on the set  $X$  of letter variables has a unique extension  $\bar{\tau}$  to the token variables  $y_{ij}^s$  that makes the clauses in  $C_0$  and the formulas  $\Psi_{ij}^s$  hold true ( $i, j = 1, \dots, n$ ,  $s = 1, \dots, t$ ). The token variable  $y_{ij}^s$  gets value 1 under  $\bar{\tau}$  if and only if after the moves  $\tau(x_1), \dots, \tau(x_s)$  of the game  $\Gamma$ , one of the tokens held by the state  $q_j$  is  $\mathbf{i}$ .*

*Proof.* We induct on  $t$ . The induction basis  $t = 0$  is clear: we have to satisfy the clauses in  $C_0$  and the only way to satisfy a one-literal clause is to assign value 1 to its only literal. Hence, independently of  $\varphi$ , we have to set for all  $i, j = 1, \dots, n$ ,

$$\bar{\tau}(y_{ij}^0) = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Observe that then, in the accordance with the initial setting of the game  $\Gamma$ , the variable  $y_{ij}^0$  gets value 1 exactly when the token held by the state  $q_j$  is  $\mathbf{i}$ .

Now suppose that  $t > 0$  and there exists a unique way to define  $\bar{\varphi}(y_{ij}^s)$  for all  $i, j = 1, \dots, n$ ,  $s = 0, \dots, t - 1$ , such that the clauses in  $C_0$  and the formulas  $\Psi_{ij}^s$  with  $i, j = 1, \dots, n$  and  $s = 1, \dots, t - 1$  hold true. If the variable  $x_t$  is assigned the value  $\varphi(x_t)$ , the value of the right hand side of each equivalence  $\Psi_{ij}^t$  is uniquely defined, and to make this equivalence hold true, we must assign the value to the left hand side, that is, the variable  $y_{ij}^t$ . This gives a unique way to extend  $\bar{\tau}$  to the variables  $y_{ij}^t$ , where  $i, j = 1, \dots, n$ . As observed prior to the formulation of the lemma, the equivalences  $\Psi_{ij}^t$  express the rule of  $\Gamma$ . Therefore the token  $\mathbf{i}$  will migrate to the state  $q_j$  after the move  $\tau(x_t)$  if and only if the variable  $y_{ij}^t$  gets value 1 under this extension.  $\square$

## 4.4 CNF formulas

For each  $t = 1, \dots, \ell$ , we define the set  $C_t$  as the set of all clauses of a suitable CNF (conjunctive normal form) equivalent to  $\bigwedge_{1 \leq i, j \leq n} \Psi_{ij}^t$ . In our basic encoding, the set  $C_t$  consists of the following clauses:

$$\neg y_{ij}^t \vee x_t \vee \bigvee_{q_h \in P_0(q_j)} y_{ih}^{t-1}, \quad \neg y_{ij}^t \vee \neg x_t \vee \bigvee_{q_k \in P_1(q_j)} y_{ik}^{t-1}, \quad (4.5)$$

$$y_{ij}^t \vee \neg x_t \vee \neg y_{ik}^{t-1} \quad \text{for each } q_k \in P_1(q_j), \quad (4.6)$$

$$y_{ij}^t \vee x_t \vee \neg y_{ih}^{t-1} \quad \text{for each } q_h \in P_0(q_j). \quad (4.7)$$

The verification of the equivalence between  $\bigwedge_{1 \leq i, j \leq n} \Psi_{ij}^t$  and the conjunction of the clauses in (4.5)–(4.7) is routine, but we include it here for the sake of completeness.

For each  $t = 1, \dots, \ell$ , and  $i, j = 1, \dots, n$  and any truth assignment such that  $x_t = 1$ , the equivalence  $\Psi_{ij}^t$  in (4.4) simplifies to

$$\Psi_{ij}^t : \quad y_{ij}^t \iff \left( \bigvee_{q_k \in P_1(q_j)} y_{ik}^{t-1} \right). \quad (4.8)$$

And the set of clauses from (4.5)–(4.7) will have the forms

$$\neg y_{ij}^t \vee \bigvee_{q_k \in P_1(q_j)} y_{ik}^{t-1}, \quad (4.9)$$

$$y_{ij}^t \vee \neg y_{ik}^{t-1} \quad \text{for each } q_k \in P_1(q_j). \quad (4.10)$$

Let  $x_t = 0$ . Then the equivalence  $\Psi_{ij}^t$  in (4.4) simplifies to

$$\Psi_{ij}^t : \quad y_{ij}^t \iff \left( \bigvee_{q_h \in P_0(q_j)} y_{ih}^{t-1} \right). \quad (4.11)$$

And the set of clauses from (4.5)–(4.7) will have the forms

$$\neg y_{ij}^t \vee \bigvee_{q_h \in P_1(q_j)} y_{ih}^{t-1}, \quad (4.12)$$

$$y_{ij}^t \vee \neg y_{ih}^{t-1} \text{ for each } q_h \in P_1(q_j). \quad (4.13)$$

It may be worth explaining how the clauses of the form (4.5)–(4.7) are understood in the case when one of the sets  $P_0(q_j)$  or  $P_1(q_j)$  or both of these sets happen to be empty. In (4.5) the disjunctions over the empty sets are omitted so that if, say,  $P_0(q_j) = \emptyset$ , then the first clause in (4.5) reduces to  $\neg y_{ij}^t \vee x_t$ . As for (4.6) or (4.7), these clauses disappear whenever  $P_1(q_j)$  or, respectively  $P_0(q_j)$  are empty.

In the next sections we formally present the three different problems. All of them use the encoding described in Sections 4.1–4.4 and an extra part called *synchronization clauses*. Such part is a critical part for each problem as it defines the required conditions for the problem.

## 4.5 NFA-synchronization problems

The synchronization of nondeterministic automata has several issues. Each one has its rules and formalization that the given automaton must satisfy in order to be a synchronizing with respect this issue. In this section we will show how model works in each problem of NFA synchronization.

### 4.5.1 $D_3$ -synchronization

This subsection presents a model from which we can test if the given NFA has a  $D_3$ -synchronizing word of a specified length or not. This problem is formally described in the following D3W problem

D3W: the existence of a  $D_3$ -synchronizing word of a given length  
 INPUT: A NFA  $\mathcal{A}$  with two input symbols and a positive integer  $\ell$ .  
 OUTPUT: YES if  $\mathcal{A}$  has a  $D_3$ -synchronizing word of length  $\ell$ ; NO otherwise.

Recall that the automaton  $\mathcal{A} = (Q, \Sigma, \delta)$  is a  $D_3$ -synchronizing if it has a word  $w \in \Sigma^*$  such that

$$\bigcap_{q \in Q} q.w \neq \emptyset$$

In previous section, the computation of the word  $w$  of length  $\ell$  in the automaton  $\mathcal{A} = (Q, \Sigma = \{0, 1\}, \delta)$  is encoded to the set of clauses  $C_0$  and  $C_t; 1 \leq t \leq \ell$ . In order to make the complete reduction from the D3W problem to an instance  $(V, C)$  of SAT, we need yet another set of clauses added to  $C_0$  and  $C_t$  that simulates the condition of  $D_3$ -synchronization. This set of clauses is called the set of synchronization clauses and is denoted by  $S$ .

By the definition of  $D_3$ -synchronization, the answer to the instance  $(\mathcal{A}, \ell)$  is YES if and only if there exists a word  $w \in \Sigma^\ell$  such that

$$\bigcap_{q \in Q} q.w \neq \emptyset.$$

Lemma 4.1 readily implies that a word  $w = a_1 \cdots a_\ell$  is  $D_3$ -synchronizing for  $\mathcal{A}$  if and only if after the moves  $a_1, \dots, a_\ell$  in the game  $\Gamma$  on  $\mathcal{A}$ , some state  $q_j$  holds all tokens  $\mathbf{1}, \dots, \mathbf{n}$ . This is equivalent to saying that the formula

$$\bigvee_{j=1}^n \bigwedge_{i=1}^n y_{ij}^\ell \tag{4.14}$$

holds true under the extension, specified in Lemma 4.2, of the truth assignment on  $X$  defined by  $w$ . A little difficulty is that a direct conversion of the formula (4.14) into a CNF produces  $2^n$  clauses. To overcome this difficulty, we use a standard trick for which we need new variables. We call such variables *synchronization variables*.

The synchronization variables are  $z_1, \dots, z_n$ . They play the role of indicators showing which states may occur at the end of applying the input word  $w$ . The clauses of  $C$  will be chosen so that the variable  $z_j$  gets value 1 in a satisfying assignment for  $C$  if and only the state  $q_j$  belongs to the set  $\bigcap_{q \in Q} q.w$ , where  $w$  is the word defined by the restriction of the assignment to  $X$ . Let  $S$  consist of the following  $n^2 + 1$  clauses:

$$\bigvee_{j=1}^n z_j \tag{4.15}$$

$$\neg z_j \vee y_{ij}^\ell \text{ for all } i, j = 1, \dots, n. \tag{4.16}$$

It is easy to see that the set  $S$  and the formula (4.14) are equisatisfiable; moreover, if  $Y = \{y_{ij}^\ell \mid i, j = 1, \dots, n\}$  and  $Z = \{z_1, \dots, z_n\}$ , then every truth assignment on  $Y$  that satisfies (4.14) can be extended to a truth assignment on  $Y \cup Z$  that satisfies  $S$ , and, conversely, for every truth assignment on  $Y \cup Z$  that satisfies  $S$ , its restriction to  $Y$  satisfies (4.14).

**Theorem 4.1.** *There is a polynomial reduction from the D3W problem to an instance  $(V, C)$  of SAT*

*Proof.* Starting with D3W problem, the reduced SAT has the set of



variables  $V$  that is the collection of three sets of variables; letter variables, the token variables, and synchronization variables. Summing up the cardinalities of these sets, the total number of variables in  $V$  is  $\ell + n^2(\ell + 1) + n$ .

Now let us estimate the number of clauses in the set  $C = S \cup \bigcup_{t=0}^{\ell} C_t$ . Let  $m$  stand for the number of all *transitions in*  $\mathcal{A}$ , that is, triples  $(q, a, q') \in Q \times \Sigma \times Q$  with  $q' \in \delta(q, a)$ . Clearly,  $m \leq 2n^2$ . For each fixed  $i$ , the number  $\sum_{j=1}^n (|P_1(q_j)| + |P_0(q_j)|)$  of clauses of the forms (4.6) and (4.7) is equal to  $m$ , whence the total number of such “short” clauses is  $mn$ . As for “long” clauses in (4.5), there are at most two such clauses for each fixed pair  $(i, j)$ , whence their total number does not exceed  $2n^2$ . Altogether,  $|C_t| \leq n(m + 2n) \leq 2n^2(n + 1)$  for each  $t = 1, \dots, \ell$ .  $|S| = n^2 + 1$ , and  $|C_0| = n^2$ . Hence  $C$  consists of at most  $n(m + 2n)\ell + 2n^2 + 1$  clauses. Thus, constructing  $(V, C)$  from  $\mathcal{A}$  takes time polynomial in  $n$  and  $\ell$ . □

Summarizing the above discussion, we arrive at the main result of the subsection.

**Theorem 4.2.** *An NFA  $\mathcal{A}$  has a  $D_3$ -synchronizing word of length  $\ell$  if and only if the instance  $(V, C)$  of SAT constructed above is satisfiable.*

Moreover, the above reduction from D3W to SAT yields the following fact:

**Corollary 4.1.** *There is a 1-1 correspondence between the  $D_3$ -synchronizing words of length  $\ell$  for  $\mathcal{A}$  and the restrictions of satisfying assignments of  $(V, C)$  to the letter variables.*

### 4.5.2 $D_2$ synchronization

At this time we are concerned with another problem of NFA synchronization, that is,  $D_2$ -synchronization. Recall that a word  $w \in \Sigma^*$  is said to be  $D_2$ -synchronizing for  $\mathcal{A} = (Q, \Sigma, \delta)$  if  $q.w = q'.w$  for all  $q, q' \in Q$ . The equality  $q.w = q'.w$  ensures that if a  $D_2$ -synchronizing word is undefined at some state, the word must be nowhere defined. Thus, a  $D_2$ -synchronizing word is either nowhere or everywhere defined. Hence the synchronization clauses have different formulas depending on the criteria that the word  $w$  must satisfy. Indeed, there is a basic criterion:

$$\forall q \in Q \exists \mathcal{R} \subset \mathcal{P}(Q), q.w = \mathcal{R} \quad (4.17)$$

And some other criteria that depend on the practical digital system of that NFA. that is: does the needed  $D_2$ -synchronizing word have to be defined everywhere or nowhere defined? The answer to this question determines the cardinality of the set  $\mathcal{R}$ , and hence we can classify the  $D_2$ -synchronization into three subclasses;  $D_2^r$ ;  $r \in \{1, 2, 3\}$ .

*Definition 4.1.* The word  $w \in \Sigma^*$  will be a  $D_2^r$ -synchronizing word for an NFA  $\mathcal{A} = (Q, \Sigma, \delta)$  if it satisfies the condition ( $D_2^r$ ) from the list below:

$$(D_2^1): \exists \mathcal{R} \subset \mathcal{P}(Q) \forall q \in Q, q.w = \mathcal{R} \text{ and } |\mathcal{R}| \geq 0,$$

$$(D_2^2): \exists \mathcal{R} \subset \mathcal{P}(Q) \forall q \in Q, q.w = \mathcal{R} \text{ and } |\mathcal{R}| > 0,$$

$$(D_2^3): \exists \mathcal{R} \subset \mathcal{P}(Q) \forall q \in Q, q.w = \mathcal{R} \text{ and } |\mathcal{R}| = 0.$$

In  $D_2^1$  and  $D_2^3$  the requirement  $q.w \neq \emptyset$  for all  $q \in Q$  is neglected. In these two versions, every word that is nowhere defined becomes a  $D_2$ -synchronizing word.

*Definition 4.2.* A given NFA  $\mathcal{A} = (Q, \Sigma, \delta)$  is  $D_2^r$ -synchronizing if it has a  $D_2^r$ -synchronizing word;  $r \in \{1, 2, 3\}$ .

Hence we describe the main problem of this subsection by  $D2^rW$  problem.

$D2^rW$ : the existence of a  $D_2^r$ -synchronizing word of a given length  
 INPUT: A NFA  $\mathcal{A}$  with 2 input symbols, a positive integer  $\ell$ , and a positive integer  $r \in \{1, 2, 3\}$ .  
 OUTPUT: YES if  $\mathcal{A}$  has a  $D_2^r$ -synchronizing word of length  $\ell$ ; NO otherwise.

In the following, we formulate the set of synchronizing clauses  $S_r$  for  $D_2^r$ -synchronizing words for every  $r = 1, 2, 3$ .

$$S_1 : \neg y_{ij}^\ell \vee y_{i+1 \pmod n, j}^\ell, \quad i, j = 0, \dots, n-1; \quad (4.18)$$

$$S_2 : \neg y_{ij}^\ell \vee y_{i+1 \pmod n, j}^\ell, \quad i, j = 0, \dots, n-1; \quad (4.19)$$

$$\bigvee_{0 \leq j \leq n-1} y_{0j}^\ell;$$

$$S_3 : \bigwedge_{0 \leq i, j \leq n-1} \neg y_{ij}^\ell. \quad (4.20)$$

Here,  $S_1$  means that all tokens are held by the same set of states or no tokens are held by any state  $q \in Q$ .

$S_2$  means that there is at least one state  $q \in Q$  holds all tokens. We call this version as *proper  $D_2$ -synchronization*

$S_3$  means that there are no tokens held by any state  $q \in Q$ . This set simulates the mortal word.

Summarizing the above discussion, we arrive at the main theorem of this subsection.

**Theorem 4.3.** *An NFA  $\mathcal{A}$  has a  $D_2^r$ -synchronizing word,  $r \in \{1, 2, 3\}$ , of length  $\ell$  if and only if the instance  $(V, C)$  of SAT constructed above is satisfiable, and the construction takes time polynomial in the size of  $\mathcal{A}$  and the value of  $\ell$ . Moreover, a word  $w = a_1 \cdots a_\ell$  with  $a_1, \dots, a_\ell \in \{0, 1\}$  is  $D_2^r$ -synchronizing for  $\mathcal{A}$  if and only if the map  $x_t \mapsto a_t$ ,  $t = 1, \dots, \ell$ , extends to a satisfying assignment for  $(V, C)$ .*

*Proof.* Let the instance SAT  $(V, C)$  be satisfiable. This means that every clause is true. Then each clause in  $S_2$  is true. The set of clauses

$$\neg y_{ij}^\ell \vee y_{i+1 \pmod n j}^\ell, \quad i, j = 0, \dots, n-1.$$

are equivalent to the cycle of implications

$$y_{0j}^\ell \rightarrow y_{1j}^\ell, \quad y_{1j}^\ell \rightarrow y_{2j}^\ell, \dots, \quad y_{n-1j}^\ell \rightarrow y_{0j}^\ell$$

Case 1: If the variable  $y_{ij}^\ell$  is true then so are all variables  $y_{kj}^\ell$  with  $k \neq i$ ,  $0 \leq k \leq n-1$ . This means that the state  $q_j$  must hold all  $n$  tokens  $\mathbf{0}, \mathbf{1}, \dots, \mathbf{n}-\mathbf{1}$ . So we can say that the application of the word  $w$  sends all the states of the automaton to the state  $j$ .

Case 2: Let all variables  $y_{ij}^\ell$ ,  $0 \leq i, j \leq n-1$ , be false. From the meaning of the token variables  $y_{ij}^\ell$ , we can extract that, at the end of the application of the word  $w$  there are no token held by any state in  $Q$ . Hence, the application of the word  $w$  sends all the states of the automaton to an undefined state. that is equivalent to the word  $w$  is nowhere defined

Then the SAT problem with a set of synchronization clauses  $S_1$  is satisfiable in either case 1 or case 2. These two situations imply that the word  $w$  is a  $D_2^1$ -synchronizing word for the given automaton.

In the case of  $D_2^2$ , we have the set of clauses  $S_2$ . If the SAT instance is satisfiable, then the clause

$$\bigvee_{0 \leq j \leq n-1} y_{0j}^\ell.$$

must be true. This clause is true if at least one of its variables  $y_{0j}^\ell$  is true. With the cycle of implications for the variables  $y_{ij}^\ell$ , this means that there is at least one state that hold all  $n$  tokens, and hence the automaton is a  $D_2^2$ -synchronizing.

For  $D_2^3$ , the synchronization clauses  $S_3$  are true if and only if all variables  $y_{ij}^\ell$  are false. Thus there are no token held by any states which implies that the word  $w$  is a mortal word for the automaton.

If the automaton is  $D_2^1$ -synchronizing;  $q.w = q'.w$  for all  $q, q' \in Q$ . In our encoding this means that, there is some state  $q_j$  that holds all  $n$  tokens  $\mathbf{0}, \mathbf{1}, \dots, \mathbf{n} - \mathbf{1}$  and hence all the variables  $y_{ij}^\ell, 0 \leq i \leq n-1$ , must be true. So the SAT problem is satisfiable. If the word  $w$  is nowhere defined, then after the move  $\ell, 0 \leq i \leq n-1$ , the variables  $y_{ij}^\ell$  must be false which means the SAT is satisfiable. This is the same situation for  $D_2^3$ -synchronizing.

Let the automaton is  $D_2^2$ -synchronizing;  $q.w = q'.w \neq \emptyset$  for all  $q, q' \in Q$ . In our encoding this means that, there is at least one state  $q_j$  that holds all  $n$  tokens  $\mathbf{0}, \mathbf{1}, \dots, \mathbf{n} - \mathbf{1}$ . Thus all clauses in the set  $S_2$  must be true. So the SAT problem is satisfiable.

The whole set  $C = S \cup \bigcup_{t=0}^{\ell} C_t$  has at most  $2n^2((n+1)\ell + 1) + 1$

clauses. The number of variables is less than the number of variables of D3W problem as in D2W we do not use the synchronization variables. Thus, the polynomial  $2x^2((x+1)y+1)+1$  can be taken as  $c(x, y)$  from the definition of polynomial reduction.  $\square$

### 4.5.3 $D_1$ -synchronization

In this subsection, we search for a word of specified length that sends the NFA automaton from any state to a unique state. This is the version of NFAs synchronization that is very similar to DFAs synchronization. The problem that we study is the following D1W problem.

D1W: the existence of a  $D_1$ -synchronizing word of a given length  
 INPUT: A NFA  $\mathcal{A}$  with 2 input symbols and a positive integer  $\ell$ .  
 OUTPUT: YES if  $\mathcal{A}$  has a  $D_1$ -synchronizing word of length  $\ell$ ; NO otherwise.

A word  $w$  is a  $D_1$ -synchronizing for the automaton  $\mathcal{A} = (Q, \Sigma, \delta)$  if  $\exists p \in Q$  such that  $\forall q \in Q, q.w = p$ . In our encoding, the word  $w \in \Sigma^\ell$  is a  $D_1$ -synchronizing word if and only if after the move  $\ell$  there is at least one state holds all tokens and this state must be the only state that holds tokens. This requirement can be simulated in the coding in two ways.

The first way is dependent on the synchronization variables. Any  $D_1$ -synchronizing automaton is  $D_3$ -synchronizing, whence the synchronization clauses of D3W problem are used here. These clauses are necessary for D1W problem but no sufficient. In order to simulate the definition of  $D_1$ -synchronization and have a guarantee that there is exactly one state that holds all  $n$  tokens, we put the set of synchronization clauses as in

the following formulas.

$$\bigvee_{j=1}^n z_j \tag{4.21}$$

$$\neg z_j \vee y_{ij}^\ell \text{ for all } i, j = 1, \dots, n. \tag{4.22}$$

$$\neg z_i \vee \neg z_j \text{ for all } i, j = 0, \dots, n-1, \text{ and } i < j \tag{4.23}$$

The second way comes from the fact that the automaton is  $D_1$ -synchronizing if at least it is  $D_2^2$ -synchronizing. With that fact we have at least one state holds all  $n$  tokens and no tokens are held by other states. In order to obtain that only one state holds all tokens, the synchronization clauses described as the conjunction of all the following clauses:

$$\neg y_{ij}^\ell \vee y_{i+1 \pmod n, j}^\ell, \quad i, j = 0, \dots, n-1. \tag{4.24}$$

$$\bigvee_{0 \leq j \leq n-1} y_{0j}^\ell. \tag{4.25}$$

$$\neg y_{0i}^\ell \vee \neg y_{0j}^\ell, \quad 0 \leq i < j \leq n-1. \tag{4.26}$$

From the discussion of  $D_2$  and  $D_3$ , the following theorem is easy to prove

**Theorem 4.4.** *The automaton  $\mathcal{A}$  has a  $D_1$  synchronizing word of length  $\ell$  if the reduced SAT has a satisfying assignment and the construction takes time polynomial in the size of  $\mathcal{A}$  and the value of  $\ell$ .*

In two ways of encoding we have an instance of SAT with number of clauses no more than  $2n^2((n+1)\ell+1)+1+\binom{n}{2}$  clauses. Moreover, the

ladder encoding, see Section 2.3 can be used to decrease this number of clauses as the majority of clauses in the set of synchronization clauses are “at-most-one” constraints.

## 4.6 Example of the CNF table for DiW problems

In what follows we explain how to use the SAT solver to solve any of the described DiW problems. Given an NFA  $\mathcal{A}$  with  $n$  states, we write the SAT instance  $(V', C')$ , which corresponds to  $(\mathcal{A}, 1)$ , in DIMACS CNF format, representing the variables  $y_{ij}^0$ ,  $y_{ij}^1$ , and  $x_1$  by the numbers, respectively,  $in + j + 1$ ,  $n^2 + in + j + 2$ , and  $n^2 + 1$ . Consider, for a simple illustration, the NFA  $\mathcal{E}_2$  shown in Fig. 4.2.

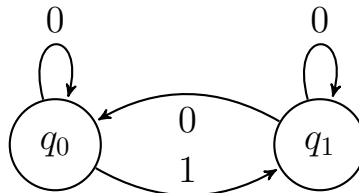


Figure 4.2: The NFA  $\mathcal{E}_2$

Table 4.1 presents our encoding of the D3W instance  $(\mathcal{E}_2, 1)$  as a SAT instance. Where we need 2 synchronization variables  $z_0$  and  $z_1$ . These variables are represented as  $2n^2 + i + 2$ . In the left column the SAT instance is shown as a list of clauses while the right column shows it in DIMACS CNF format.

Tables 4.2 and 4.3 present our encoding of the D2<sup>2</sup>W, D1W instances  $(\mathcal{E}_2, 1)$  respectively as a SAT instance.



Table 4.1: The SAT encoding of the D3W instance  $(\mathcal{E}_2, 1)$ 

Clauses	DIMACS CNF lines
	p cnf 11 29
$C'_0 \left\{ \begin{array}{l} y_{00}^0 \\ \neg y_{01}^0 \\ \neg y_{10}^0 \\ y_{11}^0 \end{array} \right.$	1 0 -2 0 -3 0 4 0
$C'_1 \left\{ \begin{array}{l} \neg y_{00}^1 \vee x_1 \vee y_{00}^0 \vee y_{01}^0 \\ \neg y_{00}^1 \vee \neg x_1 \\ y_{00}^1 \vee x_1 \vee \neg y_{00}^0 \\ y_{00}^1 \vee x_1 \vee \neg y_{01}^0 \\ \neg y_{01}^1 \vee x_1 \vee y_{01}^0 \\ \neg y_{01}^1 \vee \neg x_1 \vee y_{00}^0 \\ y_{01}^1 \vee \neg x_1 \vee \neg y_{00}^0 \\ y_{01}^1 \vee x_1 \vee \neg y_{01}^0 \\ \neg y_{10}^1 \vee x_1 \vee y_{10}^0 \vee y_{11}^0 \\ \neg y_{10}^1 \vee \neg x_1 \\ y_{10}^1 \vee x_1 \vee \neg y_{10}^0 \\ y_{10}^1 \vee x_1 \vee \neg y_{11}^0 \\ \neg y_{11}^1 \vee x_1 \vee y_{11}^0 \\ \neg y_{11}^1 \vee \neg x_1 \vee y_{10}^0 \\ y_{11}^1 \vee \neg x_1 \vee \neg y_{10}^0 \\ y_{11}^1 \vee x_1 \vee \neg y_{11}^0 \end{array} \right.$	-6 5 1 2 0 -6 -5 0 6 5 -1 0 6 5 -2 0 -7 5 2 0 -7 -5 1 0 7 -5 -1 0 7 5 -2 0 -8 5 3 4 0 -8 -5 0 8 5 -3 0 8 5 -4 0 -9 5 4 0 -9 -5 3 0 9 -5 -3 0 9 5 -4 0
$S' \left\{ \begin{array}{l} z_0 \vee z_1 \\ \neg z_0 \vee y_{00}^1 \\ \neg z_0 \vee y_{10}^1 \\ \neg z_1 \vee y_{01}^1 \\ \neg z_1 \vee y_{11}^1 \\ \neg y_{00}^1 \vee z_0 \\ \neg y_{10}^1 \vee z_0 \\ \neg y_{01}^1 \vee z_1 \\ \neg y_{11}^1 \vee z_1 \end{array} \right.$	10 11 0 -10 6 0 -10 8 0 -11 7 0 -11 9 0 10 -6 0 10 -8 0 11 -7 0 11 -9 0

Table 4.2: The SAT encoding of the D2<sup>2</sup>W instance ( $\mathcal{E}_2, 1$ )

Clauses	DIMACS CNF lines
	p cnf 9 25
$C'_0 \left\{ \begin{array}{l} y_{00}^0 \\ \neg y_{01}^0 \\ \neg y_{10}^0 \\ y_{11}^0 \end{array} \right.$	1 0 -2 0 -3 0 4 0
$C'_1 \left\{ \begin{array}{l} \neg y_{00}^1 \vee x_1 \vee y_{00}^0 \vee y_{01}^0 \\ \neg y_{00}^1 \vee \neg x_1 \\ y_{00}^1 \vee x_1 \vee \neg y_{00}^0 \\ y_{00}^1 \vee x_1 \vee \neg y_{01}^0 \\ \neg y_{01}^1 \vee x_1 \vee y_{01}^0 \\ \neg y_{01}^1 \vee \neg x_1 \vee y_{00}^0 \\ y_{01}^1 \vee \neg x_1 \neg y_{00}^0 \\ y_{01}^1 \vee x_1 \vee \neg y_{01}^0 \\ \neg y_{10}^1 \vee x_1 \vee y_{10}^0 \vee y_{11}^0 \\ \neg y_{10}^1 \vee \neg x_1 \\ y_{10}^1 \vee x_1 \vee \neg y_{10}^0 \\ y_{10}^1 \vee x_1 \vee \neg y_{11}^0 \\ \neg y_{11}^1 \vee x_1 \vee y_{11}^0 \\ \neg y_{11}^1 \vee \neg x_1 \vee y_{10}^0 \\ y_{11}^1 \vee \neg x_1 \neg y_{10}^0 \\ y_{11}^1 \vee x_1 \vee \neg y_{11}^0 \end{array} \right.$	-6 5 1 2 0 -6 -5 0 6 5 -1 0 6 5 -2 0 -7 5 2 0 -7 -5 1 0 7 -5 -1 0 7 5 -2 0 -8 5 3 4 0 -8 -5 0 8 5 -3 0 8 5 -4 0 -9 5 4 0 -9 -5 3 0 9 -5 -3 0 9 5 -4 0
$S' \left\{ \begin{array}{l} \neg y_{00}^1 \vee y_{01}^1 \\ \neg y_{01}^1 \vee y_{00}^1 \\ \neg y_{10}^1 \vee y_{11}^1 \\ \neg y_{11}^1 \vee y_{10}^1 \\ y_{00}^1 \vee y_{01}^1 \end{array} \right.$	-6 7 0 -7 6 0 -8 9 0 -9 8 0 6 7 0

Table 4.3: The SAT encoding of the D1W instance  $(\mathcal{E}_2, 1)$ 

Clauses	DIMACS CNF lines
	p cnf 9 30
$C'_0$ $\left\{ \begin{array}{l} y_{00}^0 \\ \neg y_{01}^0 \\ \neg y_{10}^0 \\ y_{11}^0 \end{array} \right.$	$\begin{array}{l} 1 \ 0 \\ -2 \ 0 \\ -3 \ 0 \\ 4 \ 0 \end{array}$
$C'_1$ $\left\{ \begin{array}{l} \neg y_{00}^1 \vee x_1 \vee y_{00}^0 \vee y_{01}^0 \\ \neg y_{00}^1 \vee \neg x_1 \\ y_{00}^1 \vee x_1 \vee \neg y_{00}^0 \\ y_{00}^1 \vee x_1 \vee \neg y_{01}^0 \\ \neg y_{01}^1 \vee x_1 \vee y_{01}^0 \\ \neg y_{01}^1 \vee \neg x_1 \vee y_{00}^0 \\ y_{01}^1 \vee \neg x_1 \vee \neg y_{00}^0 \\ y_{01}^1 \vee x_1 \vee \neg y_{01}^0 \\ \neg y_{10}^1 \vee x_1 \vee y_{10}^0 \vee y_{11}^0 \\ \neg y_{10}^1 \vee \neg x_1 \\ y_{10}^1 \vee x_1 \vee \neg y_{10}^0 \\ y_{10}^1 \vee x_1 \vee \neg y_{11}^0 \\ \neg y_{11}^1 \vee x_1 \vee y_{11}^0 \\ \neg y_{11}^1 \vee \neg x_1 \vee y_{10}^0 \\ y_{11}^1 \vee \neg x_1 \vee \neg y_{10}^0 \\ y_{11}^1 \vee x_1 \vee \neg y_{11}^0 \end{array} \right.$	$\begin{array}{l} -6 \ 5 \ 1 \ 2 \ 0 \\ -6 \ -5 \ 0 \\ 6 \ 5 \ -1 \ 0 \\ 6 \ 5 \ -2 \ 0 \\ -7 \ 5 \ 2 \ 0 \\ -7 \ -5 \ 1 \ 0 \\ 7 \ -5 \ -1 \ 0 \\ 7 \ 5 \ -2 \ 0 \\ -8 \ 5 \ 3 \ 4 \ 0 \\ -8 \ -5 \ 0 \\ 8 \ 5 \ -3 \ 0 \\ 8 \ 5 \ -4 \ 0 \\ -9 \ 5 \ 4 \ 0 \\ -9 \ -5 \ 3 \ 0 \\ 9 \ -5 \ -3 \ 0 \\ 9 \ 5 \ -4 \ 0 \end{array}$
$S'$ $\left\{ \begin{array}{l} z_0 \vee z_1 \\ \neg z_0 \vee \neg z_1 \\ \neg z_0 \vee y_{00}^1 \\ \neg z_0 \vee y_{10}^1 \\ \neg z_1 \vee y_{01}^1 \\ \neg z_1 \vee y_{11}^1 \\ \neg y_{00}^1 \vee z_0 \\ \neg y_{10}^1 \vee z_0 \\ \neg y_{01}^1 \vee z_1 \\ \neg y_{11}^1 \vee z_1 \end{array} \right.$	$\begin{array}{l} 10 \ 11 \ 0 \\ -10 \ -11 \ 0 \\ -10 \ 6 \ 0 \\ -10 \ 8 \ 0 \\ -11 \ 7 \ 0 \\ -11 \ 9 \ 0 \\ 10 \ -6 \ 0 \\ 10 \ -8 \ 0 \\ 11 \ -7 \ 0 \\ 11 \ -9 \ 0 \end{array}$

## Chapter 5

# Experiments in NFA synchronization

In this chapter we overview the application of our model on a general NFA to find the length of its shortest  $D_i$ -synchronizing word. Our basic procedure has been organized as follows.

1. A positive integer  $n$  (the number of states) is fixed. In the experiments which results we report here, we have considered  $n \leq 100$ .
2. A random NFA  $\mathcal{A}$  with  $n$  states and 2 input symbols is generated. We have used two models of random generation that are specified below.
3. We check whether  $\mathcal{A}$  has an input symbol whose action is defined at each state. If it is not the case, the NFA  $\mathcal{A}$  cannot be  $D_i$ -synchronizing (where we consider  $D_2^2$ ), and we return to Step 2 to generate another random NFA.

4. A positive integer  $\ell_0$  (the hypothetical length of the shortest  $D_i$ -synchronizing word for  $\mathcal{A}$ ) is chosen. Initially, we chose  $\ell_0$  to be close to  $n$  but, as our early experiments have revealed, it is much more practical to start with smaller values of  $\ell_0$ . We introduce three integer variables  $\ell_{\min}$ ,  $\ell$ , and  $\ell_{\max}$  and initialize them as follows:  $\ell_{\min} := 1$ ,  $\ell := \ell_0$ ,  $\ell_{\max} := 2\ell_0$ . Taking into account the fact that  $D_2^2$ - and  $D_1$ -synchronization is more restrictive than  $D_3$ -synchronization, we have used for  $D_2^2$ - and  $D_1$  slightly larger values of  $\ell_0$  than that for  $D_3$
5. With the aid of a scaling procedure, the pair  $(\mathcal{A}, \ell)$  is encoded into a SAT instance as described in Chapter 4.
6. A SAT solver is invoked to solve the SAT instance obtained in Step 5.
7. The binary search on  $\ell$  is performed. In more detail, if the SAT solver returns YES on the encoding of the pair  $(\mathcal{A}, \ell)$ , we first check whether or not  $\ell = \ell_{\min}$ . If  $\ell = \ell_{\min}$ , then  $\ell$  is the length of the shortest  $D_3$ -synchronizing word for  $\mathcal{A}$ , and we go to Step 2 to generate another random NFA. If  $\ell > \ell_{\min}$ , we update the variables  $\ell_{\max}$  and  $\ell$  by letting

$$\ell_{\max} := \ell, \quad \ell := \lfloor \frac{\ell_{\min} + \ell_{\max}}{2} \rfloor,$$

keep the value of  $\ell_{\min}$  and go to Step 5. If the SAT solver returns NO on the encoding of the pair  $(\mathcal{A}, \ell)$ , we check whether or not  $\ell = \ell_{\max}$ . If  $\ell = \ell_{\max}$ , we interpret this as the evidence that the

NFA  $\mathcal{A}$  fails to be  $D_i$ -synchronizing<sup>1</sup> and go to Step 2 to generate another random NFA. If  $\ell < \ell_{\max}$ , we update the variables  $\ell_{\min}$  and  $\ell$  by letting

$$\ell_{\min} := \ell + 1, \quad \ell := \lceil \frac{\ell_{\min} + \ell_{\max}}{2} \rceil,$$

keep the value of  $\ell_{\max}$  and go to Step 5.

## 5.1 NFA Generation

Here, we describe the methods used for random generation of NFAs. It turns out that the literature on random NFAs is sparse and no “standard” notion of a random NFA seems to have been developed so far. For instance, Tabakov and Vardi, in their widely cited paper [83], defined a random NFA  $\mathcal{A} = (Q, \{a, b\}, \delta)$  as the pair  $(D_0, D_1)$  of directed graphs sharing  $Q$  as the vertex set, where the edges of  $D_0$  and  $D_1$  represented the transitions caused by the inputs 0 and 1, respectively, and for some integer parameter  $k > 0$ , each of the directed graphs  $D_0$  and  $D_1$  had  $k$  edges chosen uniformly at random from all possible  $|Q|^2$  directed edges that could be drawn between the vertices in  $Q$ . While this definition was well suited for the purposes of [83], it does not look natural for us since there is no obvious reason why different input symbols should label the same number of transitions.

We have suggested and examined two other models to produce a ran-

---

<sup>1</sup>Of course, the equality  $\ell = \ell_{\max}$  only means that  $\mathcal{A}$  has no  $D_i$ -synchronizing word of length  $\leq 2\ell_0$ , and it is not excluded, in principle, that the NFA is  $D_i$ -synchronizing but its shortest  $D_i$ -synchronizing word is very long. However, by suitable preprocessing and choosing an appropriate value of the parameter  $\ell_0$ , we have got rid of the “bad” cases when the SAT solver returns NO and  $\ell = \ell_{\max}$  in our experiments.

dom NFA  $\mathcal{A} = (Q, \Sigma, \delta)$  with  $n$  states and 2 input symbols. We called these models as *uniform model* and *Poisson model*. With the uniform model, the random generation of  $\mathcal{A}$  based on the uniform distribution, while in the Poisson model, the generation based on the Poisson distribution with some parameter  $\lambda$ .

For each state  $q \in Q$  and each symbol  $a \in \Sigma$ , we first choose a number  $k \in \{0, 1, 2, \dots, n\}$  that serves as the cardinality of the set  $\delta(q, a)$ . In the uniform model, each  $k$  is chosen with probability  $\frac{1}{n+1}$  while in the Poisson model with parameter  $\lambda$ , each  $k < n$  is chosen with probability  $e^{-\lambda} \frac{\lambda^k}{k!}$  and  $n$  is chosen with probability  $1 - e^{-\lambda} \sum_{k=0}^{n-1} \frac{\lambda^k}{k!}$ .

With  $k$  being chosen, we proceed the same in both models, by choosing a  $k$ -element subset from all  $\binom{n}{k}$  subsets of  $Q$  with cardinality  $k$  uniformly at random and letting  $\delta(q, s)$  be the chosen subset. In each of the two models, it is easy to estimate the fraction of automata that survive Step 3. The corresponding results are stated in the following

**Proposition 5.1.** *The probability that a random NFA with  $n$  states and 2 input symbols has an input symbol whose action is defined at each state is*

$$2 \left(1 - \frac{1}{n+1}\right)^n - \left(1 - \frac{1}{n+1}\right)^{2n} \quad (5.1)$$

*if the NFA is generated under the uniform model and*

$$2(1 - e^{-\lambda})^n - (1 - e^{-\lambda})^{2n} \quad (5.2)$$

*if the NFA is generated under the Poisson model with parameter  $\lambda$ .*

*Proof.* We will introduce  $2n$  discrete variables  $k_a^q, k_b^q$  representing the cardinality  $k$  of the sets  $q.a$  and  $q.b$  respectively for each  $q \in Q$ . In

the uniform model each of these variables has a uniform distribution on  $[0, n]$ . Then, the probability that the symbol  $a$  is defined at state  $q$  or  $(q.a \neq \emptyset) = p(k_a^q \neq 0) = 1 - \frac{1}{n+1}$ . Therefore the probability that  $a$  is defined at each state is

$$p\left(\bigwedge_{q \in Q} k_a^q \neq 0\right) = \left(1 - \frac{1}{n+1}\right)^n. \quad (5.3)$$

Similarity, the probability that  $b$  is defined at each state is

$$p\left(\bigwedge_{q \in Q} k_b^q \neq 0\right) = \left(1 - \frac{1}{n+1}\right)^n. \quad (5.4)$$

When the number of states grows,  $\lim_{n \rightarrow \infty} \left(1 - \frac{1}{n+1}\right)^n = e^{-1}$  and hence the probability that the automaton has at least one symbol is defined everywhere is

$$\begin{aligned} \lim_{n \rightarrow \infty} p\left(\left(\bigwedge_{q \in Q} k_a^q \neq 0\right) \vee \left(\bigwedge_{q \in Q} k_b^q \neq 0\right)\right) \\ = e^{-1} + e^{-1} - \frac{1}{e^2} \approx 0.6. \end{aligned}$$

This ratio increases as the number of input symbols increases.

In Poisson model, the probability that  $a$  is defined at each state is  $p(k_a^q \neq 0) = (1 - e^{-\lambda})^n$ . The probability that the automaton has at least one symbol is defined everywhere is  $2(1 - e^{-\lambda})^n - (1 - e^{-\lambda})^{2n}$  that tends to 0 as  $n$  grows.  $\square$

In the further discussion, we always assume that the NFA considered have passed Step 3.



## 5.2 Uniform Model results

For NFAs generated under the uniform model, our experiments produced results that may seem surprising at the first glance. Namely, we have observed that for an overwhelming majority of  $D_3$  and  $D_2^2$ -synchronizing NFAs, the length of the shortest  $D_3$  and  $D_2^2$ -synchronizing word is 2 and 3 respectively, and this conclusion does not depend on the number of states within the range of our experiments. For an illustration, see Figure 5.1 in which the horizontal axis is the length of the shortest  $D_3$ -synchronizing word and the vertical axis is the number of NFAs. The blue and the yellow circles represent NFAs with 20 and 30 states respectively. Insofar, we have got no rigorous theoretical explanation of the

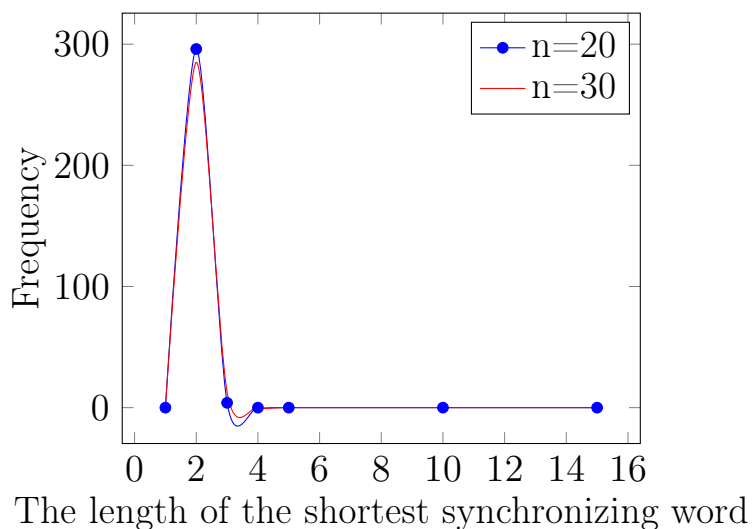


Figure 5.1: Distributions of 20- and 30-state NFAs generated under the uniform model according to the length of their shortest  $D_3$ -synchronizing words

observed phenomenon. However, even a quick analysis of the uniform

model reveals that NFAs it produces should tend to have rather short  $D_3$ -synchronizing words. Indeed, if an NFA  $\mathcal{A} = (Q, \Sigma, \delta)$  with  $n$  states and 2 input symbols is generated under the uniform model, then the expected cardinality of the set  $\delta(q, s)$  is  $\frac{n}{2}$  for every  $q \in Q$  and  $s \in \Sigma$ . Therefore the expected size of every set of the form  $q.w$  with  $w \in \Sigma^2$  is close to  $n$ . Hence it is quite likely that  $\bigcap_{q \in Q} q.w \neq \emptyset$  for some word  $w$  of length 2, which is then a  $D_3$ -synchronizing word for  $\mathcal{A}$ . The same situation holds for the case of  $D_2^2$ .

Thus, the uniform model fails to produce any “slowly synchronizing” NFAs. This indicates that using SAT-solvers in the uniform setting was not really necessary since a brute-force approach would suffice. Indeed, given an NFA  $\mathcal{A} = (Q, \Sigma, \delta)$ , one can write all words over  $\Sigma$  up to a given length in the short-lex order and apply each of these words to  $\mathcal{A}$  until one finds a  $D_3$ - or  $D_2$ -synchronizing word. As our experiments reveal, for a majority of NFAs generated under the uniform model, the brute-force approach requires to check only words up to length 3.

### 5.3 Poisson Model results

Some sample experimental results for the Poisson model are presented in Figure 5.2. The three histograms in Figure 5.2 correspond to 60-state NFAs generated under the Poisson models with three different values of the parameter  $\lambda$  and demonstrate how these NFAs are distributed according to the length of their shortest  $D_3$ -synchronizing words. In the following Subections, we use  $E_i(\lambda, n)$  for the average length of the shortest  $D_i$ -synchronizing words for  $n$ -state binary NFAs generated under the Poisson model with parameter  $\lambda$ .

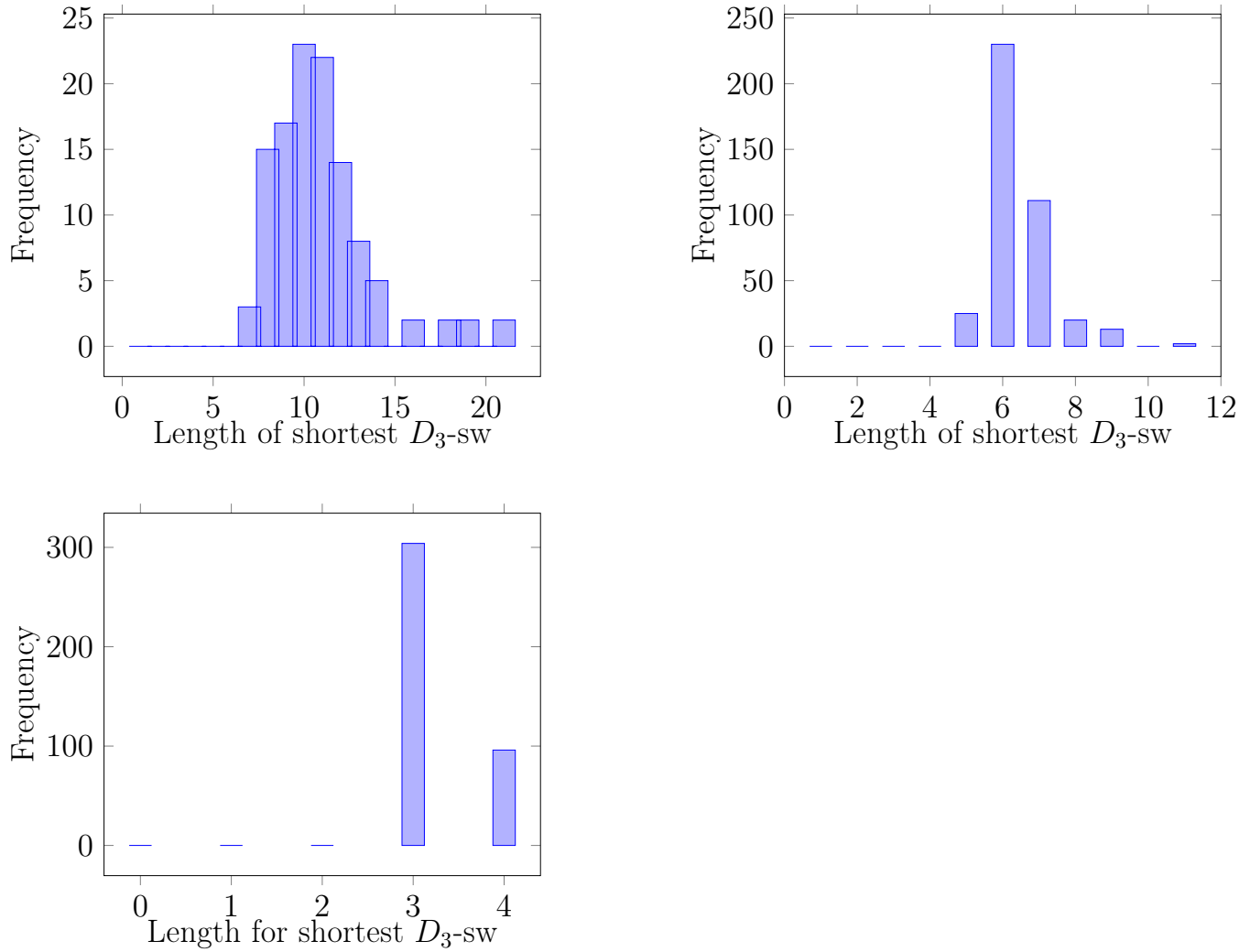


Figure 5.2: Distributions of 60-state NFAs generated under the Poisson models with  $\lambda = 1$  (top left),  $\lambda = 2$  (top right), and  $\lambda = 5$  (bottom) according to the length of their shortest  $D_3$ -synchronizing words

### 5.3.1 $D_3$ results

We see that if the number of states is fixed, the expected length of the shortest  $D_3$ -synchronizing word decreases as the parameter  $\lambda$  grows. This can be explained by an informal argument of the same flavour as the reasoning used above to explain the outcome of our experiments with NFAs generated under the uniform model. Indeed, if an NFA  $\mathcal{A} = (Q, \Sigma, \delta)$  with  $n$  states and 2 input symbols is generated under the Poisson model with parameter  $\lambda$ , it follows from a basic property of the Poisson distribution that  $\lambda$  is close to the expected cardinality of sets  $\delta(q, s)$  for every  $q \in Q$  and  $s \in \Sigma$ . The larger are these sets, the smaller is the value of  $\ell$  such that the expected size of sets of the form  $q.w$  with  $w \in \Sigma^\ell$  becomes close to  $n$ .

Our experiments also show that if the parameter  $\lambda$  is fixed, the expected length of the shortest  $D_3$ -synchronizing word grows with the number of states but the growth rate is rather small. For each  $n \leq 100$ , we have calculated the average length  $E_1(n)$  of the shortest  $D_3$ -synchronizing words for  $n$ -state NFAs generated under the Poisson model with  $\lambda = 1$ . Then, using the method of least squares, we have searched for an explicit function of  $n$  that approximates  $E_1(n)$  and found the following solution:

$$E_3(1, n) \approx (0.57 + 0.66 \ln n)^2.$$

For  $\lambda = 2$ , the same procedure has led to the following approximation of the similarly defined quantity  $E_2(n)$  calculated from our experimental data:

$$E_3(2, n) \approx (0.77 + 0.43 \ln n)^2.$$

### 5.3.2 $D_2$ results

For random NFAs generated under the Poisson model, our experiments show that if the parameter  $\lambda$  is fixed, the length of the shortest proper  $D_2$ -synchronizing word grows with the number of states but the growth rate is rather small. Some sample experimental results are presented in Fig. 5.3. The three graphs in Fig. 5.3 correspond to NFAs with 30, 45, and 60 states generated under the Poisson models with  $\lambda = 2$  and demonstrate how these NFAs are distributed according to the length of their shortest proper  $D_2$ -synchronizing words. The horizontal axis is the minimum length of proper  $D_2$ -synchronizing words and the vertical axis is the number of NFAs.

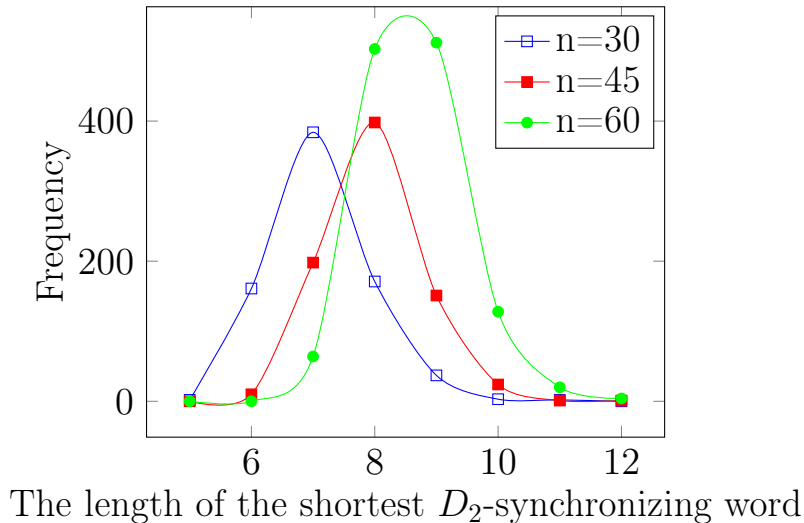


Figure 5.3: Distributions of random NFAs with 30, 45, and 60 states generated under the Poisson model with  $\lambda = 2$  according to the minimum lengths of their proper  $D_2$ -synchronizing words

We have applied the method of least squares to our experimental data, searching for an explicit function of  $n$  that approximates the mean

value  $E_\lambda(n)$  of the minimum lengths of proper  $D_2$ -synchronizing words for  $n$ -state NFAs generated under the Poisson model with a given parameter  $\lambda$ . The best approximations have been provided by logarithmic functions; for instance, for  $\lambda = 2$ , we have found the following solution:

$$E_2(2, n) \approx -0.39 + 2.2 \ln(n).$$

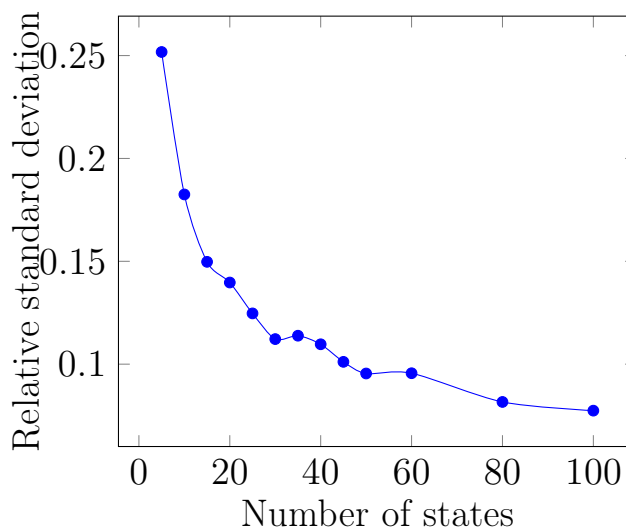


Figure 5.4: The relative standard deviation of the minimum lengths of proper  $D_2$ -synchronizing words for  $n$ -state NFAs as a function of  $n$

Fig. 5.4 shows the relation between the relative standard deviation of our datasets and the number of states (for  $\lambda = 2$ ).

## 5.4 Enhancement of the algorithm

We see several resources for improvements. First of all, we may try to modify the basic encoding described above. There are several options

for such modifications that all look promising but it is hard to predict a priori which one will prove to be the most efficient, and we have to go through several rounds of trial-and-error. As an example of a relatively successful trial; we present here the following modification of the basic encoding.

As mentioned in the description of our basic algorithm, every  $D_i$ -synchronizing NFA  $\mathcal{A}$  must have an everywhere defined input symbol. If all input symbols of  $\mathcal{A}$  are everywhere defined, one can use the transformations described in [35, Lemma 8.3.8] or [19, Section 2] to convert  $\mathcal{A}$  into a DFA  $\mathcal{A}'$  such that  $\mathcal{A}$  is  $D_3$ -synchronizing if and only if  $\mathcal{A}'$  is synchronizing and the minimum length of  $D_3$ -synchronizing words for  $\mathcal{A}$  is the same as the minimum length of synchronizing words for  $\mathcal{A}'$ . Since there are powerful methods to compute shortest synchronizing words for DFAs with up to 350 states (see, e.g., [43]), we can apply one of these methods to  $\mathcal{A}'$ . Hence, we can restrict ourselves to the case when one of the input symbols of  $\mathcal{A}$  is not everywhere defined.

If we consider only NFAs with 2 input symbols, 0 and 1, say, we conclude that we may assume that 0 is everywhere defined while 1 is not. Every  $D_3$ -synchronizing word for such an NFA should start with the symbol 0. Therefore one can start our solitaire-like game  $\Gamma$  described in Section 4.1 from the position that arises after the first application of 0, and the basic encoding can be modified accordingly<sup>2</sup>. For an NFA with  $n$  states and  $m$  transitions, this preprocessing allows one to save  $n^2$  variables and around  $n^2 + 2m$  clauses in the resulting instance of SAT. Our experiments show that this modification indeed reduces the execution time of solving D3W-instances for NFAs with  $\geq 20$  states, and

---

<sup>2</sup>If we re-use the illustrative example in Figure 4.1, the new initial position for this example will be the one shown in bottom left.

the average time decrease reaches 50% for NFAs with  $\geq 50$  states. Also, the modification has allowed us to solve D3W for NFAs with more than 100 states which size was out of reach with the basic encoding.

Table 5.1: Comparison of running time between two versions of coding

n	without modification	with modification
10	41.54 sec	46.88sec
20	140.9 sec	121.5 sec
25	273.9 sec	210.9sec
30	400.9 sec	302.5 sec
40	1218 sec	595.8 sec
50	2357 sec	1377 sec
60	4351 sec	2157 sec
100	?	$1.027e^4$ sec

*Remark 5.1.* It is possible to reduce the number of variables by getting rid of the letter variables. Namely, for each pair of  $i, j \in \{1, \dots, n\}$  and each  $t \in \{1, \dots, \ell\}$ , one could take the clause

$$\neg y_{ij}^t \vee \bigvee_{q_h \in P_0(q_j)} y_{ih}^{t-1} \vee \bigvee_{q_k \in P_1(q_j)} y_{ik}^{t-1} \quad (5.5)$$

instead of the clauses in (4.5) and the set of clauses of the form

$$y_{ij}^t \vee \neg y_{ih}^{t-1} \vee \neg y_{ik}^{t-1} \text{ for } h \text{ and } k \text{ such that } q_h \in P_0(q_j) \text{ and } q_k \in P_1(q_j) \quad (5.6)$$

instead of the ones in (4.6) and (4.7). It is easy to see that (4.5) and



(5.5) are equisatisfiable, and so are the sets of clauses in (4.6), (4.7) on the one hand and in (5.6) on the other.

We have preferred to keep the letter variables because of the fact mentioned in Theorems 4.2, 4.3, and 4.4: if a  $D_i$ -synchronizing word of length  $\ell$  exists, we can immediately recover it from the the restrictions of a satisfying assignment to the letter variables.

# Conclusion

Through the thesis we studied the synchronization of nondeterministic automata and partial deterministic automata. The synchronization of such automata is more complicated than that of complete deterministic automata. There are more than one formalization of synchronization for nondeterministic and partial deterministic automata. We have considered five formalization. For PFAs we gave attention to Careful and Exact synchronization. In NFAs we studied three versions of synchronization;  $D_1$ -,  $D_2$ -, and  $D_3$ -synchronization. Each version of synchronization appears in some applications. The length of the synchronizing word is significant and the minimum length of such words has got a lot of attention from the practical and theoretical point of view. We introduced a technique that takes the problem  $(\mathcal{A}, \ell)$  as an input and the output of it determines if the automaton  $\mathcal{A}$  has a synchronizing word of length  $\ell$  or not. We proved the validity of this technique and used some of benchmarks to prove the efficiency of it comparing to the other known algorithms.

With successful application of this technique we can find the length of the shortest synchronizing word for a given automaton. We performed a series of experiments on randomly generated NFAs and PFAs with  $n$  states. The results of this experimental study were the approximation of

the length of the shortest synchronizing word for the given automaton. We study parameters that affect this length. Two new infinite series of slowly synchronizing PFAs have been found.

It turns out that our approach works reasonably well even though our implementations have employed a very basic SAT solver (MiniSat) and very modest computational resources (an ordinary personal computer). In order to expand the range of future experiments, it makes sense to use more advanced SAT solvers. Using more powerful computers constitutes another obvious direction for improvements. Clearly, the approach is amenable to parallelization since computations needed for different automata are completely independent so that one can process in parallel as many automata as many processors are available. Still, we think that the present results, obtained without any advanced tools, do provide some evidence for our approach to be feasible in principle.

# Bibliography

- [1] Altun, Ö.F., Atam, K.T., Karahoda, S., Kaya, K.: Synchronizing heuristics: Speeding up the slowest. In *Testing Software and Systems, 29th Int. Conf., ICTSS 2017*, volume 10533 of LNCS, pages 243–256. Springer, 2017.
- [2] Ananichev, D.S., Volkov, M.V.: Some results on Černý type problems for transformation semigroups. In *Semigroups and Languages*, pages 23–42, 2004.
- [3] Ananichev, D.S., Volkov, M.V., Gusev, V.V.: Primitive digraphs with large exponents and slowly synchronizing automata. *J. Math. Sci.*, 192(3):263–278, 2013.
- [4] Berlinkov, M.V.: On two algorithmic problems about synchronizing automata. In *Developments in Language Theory, 18th Int. Conf., DLT 2014*, volume 8633 of LNCS, pages 61–67. Springer, 2014.
- [5] Berlinkov, M.V.: On the probability of being synchronizable. In *Algorithms and Discrete Applied Mathematics, 2nd Int. Conf., CALDAM 2016*, volume 9602 of LNCS, pages 73–84. Springer, 2016.

- [6] Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook on Satisfiability. *IOS Press*, 2009.
- [7] Blondel, V.D., Jungers, R.M., Olshevsky, A.: On primitivity of sets of matrices. *Automatica*, 61(C):80–88, 2015.
- [8] Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A.: Model-Based Testing of Reactive Systems. volume 3472 of LNCS. Springer, 2005.
- [9] Berstel, J., Perrin, D., Reutenauer, C.: Codes and Automata. *Cambridge University Press*, 2009.
- [10] Bonizzoni, P., Jonoska, N.: Existence of constants in regular splicing languages. *Information and Computation*, 242:340–353, 2015.
- [11] Burkhard, H.-D.: Zum Längenproblem homogener Experimente an determinierten und nicht-deterministischen Automaten. *Elektronische Informationsverarbeitung und Kybernetik*, 12:301–306, 1976 (in German).
- [12] Chang, Y., Studený, J., Suomela, J.: Distributed graph problems through an automata-theoretic lens. Available at <https://arxiv.org/abs/2002.07659>, 2020.
- [13] Capocelli, R.M., Gargano, L., Vaccaro, U.: On the characterization of statistically synchronizable variable-length codes. *IEEE Transactions on Information Theory*, 34(4):817–825, 1988.
- [14] Catalano, C., Jungers, R.M.: On random primitive sets, directable NFAs and the generation of slowly synchronizing DFAs. *J. Automata, Languages and Combinatorics*, 24(2-4):185–217, 2019.

- [15] Černý, J.: Poznámka k homogénnym experimentom s konečnými automatami. *Matematicko-fyzikalny Časopis Slovenskej Akadémie Vied* 14(3): 208–216, 1964 (in Slovak); Engl. translation: A note on homogeneous experiments with finite automata. *J. Automata, Languages and Combinatorics*, 24(2-4):121–130, 2019.
- [16] Chow, T.S.: Testing software design modeled by finite state machines. *IEEE Transactions on Software Engineering*, 4:178–178, 1978.
- [17] Cormen, T.H., Leiserson, Ch.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. *MIT Press*, 2001.
- [18] de Bondt, M., Don, H., Zantema, H.: Lower bounds for synchronizing word lengths in partial automata. *Int. J. Found. Comput. Sci.*, 30(1):29–60, 2019.
- [19] Don, H., Zantema, H.: Synchronizing non-deterministic finite automata. *J. Automata, Languages and Combinatorics*, 23(4):307–328, 2018.
- [20] Eén, N., Sörensson, N.: An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 of LNCS pages 502–518. Springer, 2004.
- [21] Eén, N., Sörensson, N.: The MiniSat Page. <http://minisat.se>.
- [22] Eppstein, D.: Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19:500–510, 1990.
- [23] Frankl, P.: An extremal problem for two families of sets. *European J. Combinatorics*, 3:125–127, 1982.

- 
- [24] Gawrychowski, P.: Complexity of shortest synchronizing word. Private communication, 2008.
- [25] Gazdag, Z., Iván, S., Nagy-György, J.: Improved upper bounds on synchronizing nondeterministic automata. *Inf. Process. Lett.*, 109:986–990, 2009.
- [26] Geldenhuys, J., van der Merwe, B., van Zijl, L.: Reducing nondeterministic finite automata with SAT solvers. In *Finite-State Methods and Natural Language Processing, 8th Int. Workshop, FSMNLP 2009*, volume 6062 of LNCS, pages 81–92. Springer, 2010.
- [27] Gent, I.P., Nightingale, P.: A new encoding of AllDifferent into SAT. In *Modelling and Reformulating Constraint Satisfaction Problems: Towards Systematisation and Automation, 3rd Int. Workshop*, pages 95–110, 2004. Available at <http://www-users.cs.york.ac.uk/~frisch/ModRef/04/proceedings.pdf>
- [28] Gerencsér, B., Gusev, V.V., Jungers, R.M.: Primitive sets of non-negative matrices and synchronizing automata. *SIAM J. Matrix Analysis and Applications*, 39(1):83–98, 2018.
- [29] Gomes, C.P., Kautz, H., Sabharwal, A., Selman, B.: Satisfiability solvers. Chapter 2 in *Handbook of Knowledge Representation*. Elsevier, 89–134, 2008.
- [30] Goldberg, K.Y.: Orienting polygonal parts without sensors. *Algorithmica*, 10(2-4):210–225, 1993.
- [31] Goralčík, P., Hedrlín, Z., Koubek, V., Ryšlinková, J.: A game of composing binary relations. *RAIRO Inform. Théor.*, 16(4):365–369, 1982.

- [32] Güniçen, C., Erdem, E., Yenigün, H.: Generating shortest synchronizing sequences using Answer Set Programming. In *Proceedings of Answer Set Programming and Other Computing Paradigms (ASPOCP), 6th Int. Workshop*, pages 117–127, 2013. Available at <https://arxiv.org/abs/1312.6146>
- [33] Harris, B.: Probability distributions related to random mappings. *Ann. Math. Statist.*, 31(4), 1045–1062, 1960.
- [34] Imreh, B., Steinby, M.: Directable nondeterministic automata. *Acta Cybernetica*, 14:105–115, 1999.
- [35] Ito, M.: Algebraic Theory of Automata and Languages. *World Scientific*, 2004.
- [36] Ito, M., Shikishima-Tsuji, K.: Some results on directable automata. In *Theory Is Forever. Essays dedicated to Arto Salomaa on the occasion of his 70th birthday*, volume 3113 of LNCS, pages 125–133. Springer, 2004.
- [37] Ito, M., Shikishima-Tsuji, K.: Shortest directing words of nondeterministic directable automata. *Discrete Mathematics*, 308(21):4900–4905, 2008.
- [38] Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. In *Automata, Languages and Programming, 18th Int. Colloquium, ICALP 1991*, volume 510 of LNCS, pages 629–640. Springer, 1991.
- [39] Karahoda, S., Erenay, O.T., Kaya, K., Türker, U.C., Yenigün, H.: Parallelizing heuristics for generating synchronizing sequences. In *Testing Software and Systems, 28th Int. Conf., ICTSS 2016*, volume 9976 of LNCS, pages 106–122. Springer, 2016.



- [40] Karahoda, S., Erenay, O.T., Kaya, K., Türker, U.C., Yenigün, H.: Multicore and manycore parallelization of cheap synchronizing sequence heuristics. *J. Parallel and Distributed Computing*, 140:13–24, 2020.
- [41] Karahoda, S., Kaya, K., Yenigün, H.: Synchronizing heuristics: Speeding up the fastest. *Expert Syst. Appl.*, 94:265–275, 2018.
- [42] Kari, J., Volkov, M.V.: Černý’s conjecture and the Road Coloring Problem. *Chapter 15 in Handbook of Automata Theory*, Volume I of EMS Publishing House (in print).
- [43] Kisielewicz, A., Kowalski, J., Szykuła, M.: Computing the shortest reset words of synchronizing automata. *J. Comb. Optim.*, 29(1):88–124, 2015.
- [44] Kowalski, J., Roman, A.: A new evolutionary algorithm for synchronization. In *Applications of Evolutionary Computation, 20th European Conf., EvoApplications 2017, Part I*, volume 10199 of LNCS, pages 620–635. Springer, 2017.
- [45] Kushik, N., El-Fakih, Kh., Yevtushenko, N.: Preset and adaptive homing experiments for nondeterministic finite state machines. In *Implementation and Application of Automata, 16th Int. Conf., CIAA 2011*, volume 6807 of LNCS, pages 215–224. Springer, 2011.
- [46] Kushik, N., El-Fakih, Kh., Yevtushenko, N.: Adaptive homing and distinguishing experiments for nondeterministic finite state machines. In *Testing Software and Systems, 25th Int. Conf., ICTSS 2013*, volume 8254 of LNCS, pages 33–48. Springer, 2013.
- [47] Kushik, N., Yevtushenko, N.: On the length of homing sequences for nondeterministic finite state machines. In *Implementation and*

- Application of Automata, 18th Int. Conf., CIAA 2013*, volume 7982 of LNCS, pages 220–231. Springer, 2013.
- [48] Kushik, N., Yevtushenko, N.: Describing homing and distinguishing sequences for nondeterministic finite state machines via synchronizing automata. In *Implementation and Application of Automata, 20th Int. Conf., CIAA 2015*, volume 9223 of LNCS, pages 188–198. Springer, 2015.
- [49] Kushik, N., Yevtushenko, N., Bourdonov, I.B. Kossatchev, A.S.: Deriving synchronizing and homing sequences for input/output automata, *Automatic Control and Comput. Sci.*, 52(7), 589–595, 2018.
- [50] Kushik, N., Yevtushenko, N., Yenigün, H.: Reducing the complexity of checking the existence and derivation of adaptive synchronizing experiments for nondeterministic FSMs. In *Proc. Int. Workshop on domain specific Model-based Approaches to verification and validation, AMARETTO@MODELSWARD 2016*, pages 83–90. SciTePress, 2016.
- [51] Lee, D., Yannakakis, M.: Testing finite-state machines: State identification and verification. *IEEE Transactions on Computers*, 43(3):306–320, 1994.
- [52] Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines. *Proceedings of the IEEE*, 84(8):1090–1126, 1996.
- [53] Liu, C.L.: Some memory aspects of finite automata. *Technical Report 411, Research Lab. Electronics, Massachusetts Inst. Technology, Cambridge, MA*, 1963.

- [54] Martyugin, P.V.: Problems concerning synchronization of finite automata. *PhD thesis, Ural State University. Yekaterinburg*, 2008 (in Russian)
- [55] Martyugin, P.V.: Lower bounds for the length of the shortest carefully synchronizing words for two- and three-letter partial automata. *Diskretn. Anal. Issled. Oper.*, 15(4):44–56, 2008 (in Russian).
- [56] Martyugin, P.V.: Complexity of problems concerning reset Words for some partial cases of automata. *Acta Cybernetica* 19:517–536, 2009.
- [57] Martyugin, P.V.: A lower bound for the length of the shortest carefully synchronizing words. *Russian Math. (Iz. VUZ)*, 54(1):46–54, 2010.
- [58] Martyugin, P.V.: Synchronization of automata with one undefined or ambiguous transition. In *Implementation and Application of Automata, 17th Int. Conf., CIAA 2012*, volume 7381 of LNCS, pages 278–288. Springer, 2012.
- [59] Martyugin, P.V.: Careful synchronization of partial automata with restricted alphabets. In *Computer Science – Theory and Applications, 8th Int. Comput. Sci. Symposium in Russia, CSR 2013*, volume 7913 of LNCS, pages 76–87. Springer, 2013.
- [60] Martyugin, P.V.: Complexity of problems concerning carefully synchronizing words for PFA and directing words for NFA. *Theory Comput. Syst.*, 54(2):293–304, 2014.

- 
- [61] Natarajan, B.K.: An algorithmic approach to the automated design of parts orienters. In *27th Annual Symposium on Foundations of Computer Science*, pages 132–142, 1986.
- [62] Natarajan, B.K.: Some paradigms for the automated design of parts feeders. *Int. J. Robotics Research*, 8(6):89–109, 1989.
- [63] Nicaud, C.: Fast synchronization of random automata. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. APPROX/RANDOM 2016*, volume 60 of LIPIcs, pages 43:1–43:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
- [64] Nicaud, C.: The Černý Conjecture holds with high probability. *J. Automata, Languages and Combinatorics*, 24(2-4), 343–365, 2019.
- [65] Olschewski, J., Ummels, M.: The complexity of finding reset words in finite automata. In *Mathematical Foundations of Computer Science*, volume 6281 of LNCS, pages 568–579. Springer, 2010.
- [66] Papadimitriou, C.H.: Computational Complexity. *Addison-Wesley*, 1994.
- [67] Pin, J.-E.: On two combinatorial problems arising from automata theory. *Annals of Discrete Mathematics*, 17:535–548, 1983.
- [68] Pixley, C., Jeong, S.-W., Hachtel, G.D.: Exact calculation of synchronization sequences based on binary decision diagrams. In *Proceedings 29th Design Automation Conference*, pages 620–623, 1992.
- [69] Podolak, I.T., Roman, A., Jędrzejczyk, D.: Application of hierarchical classifier to minimal synchronizing word problem. In *Artificial*

- Intelligence and Soft Computing, 11th Int. Conf., ICAISC 2012*, volume 7267 of LNCS, pages 421–429. Springer, 2012.
- [70] Podolak, I.T., Roman, A., Szykuła, M., Zieliński, B.: A machine learning approach to synchronization of automata. *Expert Syst. Appl.*, 97:357–371, 2018.
- [71] Rabin, M.O., Scott, D.: Finite automata and their decision problems. *IBM J. Research and Development*, 3(2):114–125, 1959.
- [72] Ramírez Alfonsín, J.L.: The Diophantine Frobenius Problem. *Oxford University Press*, 2005.
- [73] Roman, A.: Genetic algorithm for synchronization. In *Language and Automata-Theory and Applications, 3rd Int. Conf., LATA 2009*, volume 5457 of LNCS, pages 684–695. Springer, 2009.
- [74] Rystsov, I.K.: Reset words for commutative and solvable automata. *Theor. Comput. Sci.*, 172(1):273–279, 1997.
- [75] Rystsov, I.K.: Asymptotic estimate of the length of a diagnostic word for a finite automaton. *Cybernetics*, 16(1):194–198, 1980.
- [76] Rystsov, I.K.: Polynomial complete problems in automata theory. *Inf. Process. Lett.*, 16(3):147–151, 1983.
- [77] Skvortsov, E., Tipikin, E.: Experimental study of the shortest reset word of random automata. In *Implementation and Application of Automata, 16th Int. Conf., CIAA 2011*, volume 6807 of LNCS, pages 290–298. Springer, 2011.
- [78] Samotij, W.: A note on the complexity of the problem of finding shortest synchronizing words. In *Proceedings of AutoMathA: Au-*

- 
- tomata from Mathematics to Applications*, University of Palermo (CD), 2007.
- [79] Sandberg, S.: Homing and synchronizing sequences. In *Model-Based Testing of Reactive Systems*, volume 3472 of LNCS, pages 5–33. Springer, 2005.
- [80] Steinby, M.: Directable fuzzy and nondeterministic automata. Available at <https://arxiv.org/abs/1709.07719>, 2017.
- [81] Shitov, Y.: An improvement to a recent upper bound for synchronizing words of finite automata. *J. Automata, Languages and Combinatorics*, 24(2–4):367–373, 2019.
- [82] Szykuła, M.: Improving the upper bound on the length of the shortest reset word. In *35th Symposium on Theoretical Aspects of Computer Science*, volume 96 of LIPIcs, pages 56:1–13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
- [83] Tabakov, D., Vardi, M.Y.: Experimental evaluation of classical automata constructions. In *Logic for Programming, Artificial Intelligence, and Reasoning, 11th Int. Conf., LPAR 2004*, , volume 3835 of LNCS, pages 396–411. Springer, 2005.
- [84] Travers, N., Crutchfield, J.: Exact Synchronization for finite-State Sources. *J. Stat. Phys.*, 145(5):1181–1201, 2011.
- [85] Travers, N., Crutchfield J.: Asymptotic synchronization for finite-state sources. *J. Stat. Phys.*, 145(5):1202–1223, 2011.
- [86] Türker, U.C.: Parallel brute-force algorithm for deriving reset sequences from deterministic incomplete finite automata. *Turk. J. Elec. Eng & Comput. Sci.*, 27:3544–3556, 2019.

- [87] Vorel, V.: Subset synchronization and careful synchronization of binary finite automata. *Int. J. Found. Compu. Sci.*, 27(5):557–577, 2016.
- [88] Volkov, M.V.: Synchronizing automata and the Černý conjecture. In *Language and Automata Theory and Applications*, volume 5196 of LNCS, pages 11–27. Springer, 2008.
- [89] Yenigün, H., Yevtushenko, N., Kushik, N.: The complexity of checking the existence and derivation of adaptive synchronizing experiments for deterministic FSMs. *Inf. Process. Lett.*, 127: 49–53, 2017.
- [90] Yevtushenko, N., Kuli Amin, V.V., Kushik, N.: Evaluating the complexity of deriving adaptive homing, synchronizing and distinguishing sequences for nondeterministic FSMs, In *Testing Software and Systems, 31st Int. Conf., ICTSS 2019*, volume 11812 of LNCS, pages 86–103. Springer, 2019.
- [91] Zubkov, A.M.: Computation of distributions of the numbers of components and cyclic points for random mappings. *Mat. Vopr. Kriptogr.*, 1(2):5–18, 2010 (in Russian).

РОССИЙСКАЯ ФЕДЕРАЦИЯ



## СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2018663225

**NFASync: Программный комплекс для вычисления порога синхронизации недетерминированных конечных автоматов**

Правообладатель: *Федеральное государственное автономное образовательное учреждение высшего образования «Уральский федеральный университет имени первого Президента России Б.Н.Ельцина» (RU)*

Автор: *Шабана Ханан Магди Дарвиш (EG)*

Заявка № **2018616560**

Дата поступления **26 июня 2018 г.**

Дата государственной регистрации

в Реестре программ для ЭВМ **24 октября 2018 г.**

*Руководитель Федеральной службы  
по интеллектуальной собственности*

 *Г.П. Ивлиев*





РОССИЙСКАЯ ФЕДЕРАЦИЯ



## СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2019663027

**Программа OSW для вычисления оптимального  
синхронизирующего слова для частичного  
детерминированного автомата.**

Правообладатель: *Федеральное государственное автономное  
образовательное учреждение высшего образования «Уральский  
федеральный университет имени первого Президента России  
Б.Н. Ельцина» (RU)*

Автор: *Шабана Ханан Магди Дарвиш (EG)*



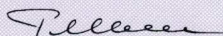
Заявка № 2019661853

Дата поступления 25 сентября 2019 г.

Дата государственной регистрации

в Реестре программ для ЭВМ 08 октября 2019 г.

Руководитель Федеральной службы  
по интеллектуальной собственности

 Г.П. Ивлиев