# Synchronizing automata and coding theory

Andrew Ryzhikov

UNIVERSITÉ PARIS-EST

ÉCOLE DOCTORALE MATHÉMATIQUES ET STIC

Thèse de doctorat
Informatique

# SYNCHRONIZING AUTOMATA AND CODING THEORY

Andrew RYZHIKOV

Thèse dirigée par Prof. Dominique PERRIN
Soutenue le 25 novembre 2020 devant les membres du jury :

| | | |
|---|---|---|
| Président | Prof. Jean-Éric PIN | CNRS |
| Directeur de thèse | Prof. Dominique PERRIN | Université Gustave Eiffel |
| Rapporteur | Prof. Antonio RESTIVO | University of Palermo |
| Rapporteur | Prof. Mikhail VOLKOV | Ural Federal University |
| Examinatrice | Prof. Marie-Pierre BÉAL | Université Gustave Eiffel |
| Examinatrice | Dr. Laure DAVIAUD | City, University of London |
| Examinatrice | Dr. Mahsa SHIRMOHAMMADI | Université Paris Diderot |

# Abstract

This thesis is devoted to studying synchronizing words for finite automata and variable-length codes. Intuitively, a synchronizing word is a word such that its application brings a system into some particular state regardless of its initial state. A code or automaton is synchronizing if it admits a synchronizing word. There is a deep connection between the concrete implementations of this notion for finite automata and variable-length codes, and the use of it is one of the leitmotifs of this thesis.

One of the main tools for dealing with variable-length codes is assigning a special automaton preserving many synchronization properties to it. For finite codes, this is done using prefix automata.

We investigate two fundamental problems about synchronization. The first one is measuring the length of shortest synchronizing words for synchronizing automata and codes. For the case of complete deterministic finite automata, the Černý conjecture, one of the oldest problems in combinatorial automata theory, proposes a quadratic upper bound in terms of the number of states in the automaton. We broaden this conjecture to partial deterministic finite automata and then to unambiguous non-deterministic finite automata. We show polynomial upper bounds for mentioned classes in the presence of strong connectivity, and discuss their connections with the original Černý conjecture.

The relation with finite codes allows us to show similar bounds in terms of the sum of the lengths of the codewords, as well as in terms of the length of the longest codeword. The second case is related to another important conjecture stated by Restivo. It deals with mortal words instead of synchronizing ones, the case which we also investigate.

For most of these settings we also examine the computational complexity of finding short synchronizing words, which is the second fundamental question investigated in this thesis. Besides that, we study various generalization of the notion of a synchronizing word, which allow to use some partial information about the current state of an automaton. For several such generalizations we show strong computational complexity lower bounds even in very restricted cases.



**Keywords:** synchronizing automaton, variable-length code, finite code, Černý conjecture, Restivo's conjecture, weakly acyclic automaton, monotonic automaton, synchronizing word, mortal word

# Résumé

Cette thèse est consacrée a l'étude des mots synchronisants pour les automates finis et les codes. Intuitivement, un mot synchronisant est tel que son application amène le système dans un état indépendant de létat initial. Un code ou un automate est synchronisant s'il admet un mot synchronisant. Il existe une forte relation entre l'utilisation de cette notion pour les automates ou les codes et son usage est l'un des leitmotifs de cette thèse.

Un des principaux outils pour traiter les codes est l'usage d'un automate particulier préservant ses propriétés de synchronisation. Pour les codes préfixes, ceci est obtenu en utilisant un automate déterministe.

Nous avons exploré deux problèmes principaux concernant la synchronisation. Le premier consiste à mesurer la longueur du plus court mot synchronisant pourun automate ou un code. Dans le cas d'un automate fini complet et déterministe, un des problèmes les plus anciens de la théorie combinatoire des automates propose une borne quadratique en termes du nombre détats de l'automate. Nous étendons cette conjecture aux automates partiels non-déterministes. Nous établissons des bornes polynomiales pour ces classes en présence de la forte connexité, et nous discutons leur lien avec la conjecture originale de Černý.

La relation avec les codes finis nous permet de prouver l'existence de bornes similaires en termes de la somme des longueurs des mots du code, et aussi de la longueur du plus long mot. Le deuxième cas est lié à une autre importante conjecture, proposée par Restivo. Elle concerne les mots mortels plutot que synchronisants.

Dans la plupart de ces cas, nous examinons aussi la complexité de calcul d'un mot synchronisant court, un deuxième aspect fondamental de cette thèse. Au-delà, nous étudions plusieurs généralisations de la notion de mot synchronisant, qui permettent d'utiliser une information partielle sur sur l'état dand lequel se trouve l'automate. Pour plusieurs de ces généralisations, nous prouvons des bornes inférieures de complexité, même dans dans cas très particuliers.

**Mots clé**: Automate synchronisant, code à longueur variable, code fini, conjecture de Černý, conjecture de Restivo, automate faiblement acyclique, automate monotone, mot synchronisant, mot mortel.

# Acknowledgments

First and most of all, I am grateful to my advisor, Dominique Perrin, who was always responsive, helpful and patient. You are a constant inspiration to me, and a great illustration of how things should be done. I also want to thank all my coauthors: besides Dominique, they are Mikhail V. Berlinkov, Robert Ferens, Ilia Fridman, Jarkko Kari, Yury Kartynnik, Mikhail Y. Kovalyov, Clemens Müllner, Erwin Pesch, Anton Shemyakov, Marek Szykuła, Anton Varonka. I've learned so much from all of you, and without you this thesis would simply never exist. I am also grateful to Stefan Kiefer, who has helped me in working on Chapter 5, and to all anonymous reviewers of my papers, who contributed a lot to improving every part of this thesis.

I am deeply grateful to people who advised my research investigations before my PhD studies, both formally and informally: Nadia Brauner, Boris Doubrov, Vincent Jost, Mikhail Y. Kovalyov, Alantha Newman, Yury L. Orlovich, Michel Rigo, András Sebő, Yakov M. Shafransky. I am indebted to Peter Cameron for his invited talk on synchronizing automata in Derby in 2016; this was the first time I heard about this topic, and two years later I found myself writing a PhD thesis on it.

I am very thankful to the reviewers of this thesis, Antonio Restivo and Mikhail V. Volkov, as well as the other jury members: Marie-Pierre Béal, Laure Daviaud, Jean-Éric Pin, Mahsa Shirmohammadi.

My deep appreciation goes to my colleagues and friends Mèlodie Andrieu, Francesco Dolce, Thibault Godin, Vladimir V. Gusev, Pavel Heller, Revekka Kyriakoglou, Olga G. Parshina, Michaël Postic, Elena V. Pribavkina, Daria Tieplova.

Special thanks go to the person who first showed and taught me real mathematics, my high school math teacher Mikhail N. Volkov.

Finally, I would like to thank all the coffee and tea manufacturers, coffee shops, and coffee machine companies that made this work possible in the given time period, as well as one particular bird-decorated website for making it so much easier.

This thesis was prepared in Laboratoire d'Informatique Gaspard-Monge UMR 8049, Cité Descartes, Bâtiment Copernic - 5, bd Descartes Champs sur Marne 77454 Marne-la-Vallée Cedex 2, France.

# Contents

**Abstract** 2

**Résumé** 3

**Acknowledgments** 4

**Introduction** 9

**1 Introduction** 9
- 1.1 Testing of Reactive Systems . . . 9
- 1.2 Orienting Parts in Manufacturing . . . 10
- 1.3 Error Recovery for Variable-Length Codes . . . 11
- 1.4 Mortal Words as Resynchronization Markers . . . 13
- 1.5 Reachability in Automata . . . 14

**2 Main Definitions and Existing Results** 15
- 2.1 Synchronizing DFAs . . . 15
- 2.2 Synchronizing NFAs . . . 17
- 2.3 Codes . . . 18
- 2.4 Automata for Codes . . . 19
- 2.5 Monotonic and Weakly Acyclic Automata . . . 20
- 2.6 Subset Synchronization . . . 21
- 2.7 Careful Synchronization . . . 22
- 2.8 Černý and Restivo's Conjectures . . . 22
- 2.9 Finding Shortest Synchronizing Words . . . 23
- 2.10 Organization of the Thesis . . . 24

**Extremal Bounds** 26

**3 Extremal Bounds for Complete DFAs** 26
- 3.1 Introduction . . . 26
- 3.2 Digraphs and Automata . . . 27
- 3.3 Strongly Connected Automata . . . 27
  - 3.3.1 A Lower Bound for Ternary Automata . . . 27
  - 3.3.2 Lower Bounds for Binary Automata . . . 28

3.3.3 Upper Bound in the Case When the Rank Equals the Period . . . 30
3.4 Eulerian Automata . . . 31
3.4.1 The Rank Conjecture . . . 31
3.4.2 A Corollary for Circular Automata . . . 32
3.4.3 A Road Coloring Algorithm . . . 33

**4 Extremal Bounds for Partial DFAs and Prefix Codes 35**
4.1 Introduction . . . 35
4.2 Preliminaries . . . 36
4.3 Upper Bounds . . . 37
4.3.1 Inseparability Equivalence . . . 37
4.3.2 Fixing Automaton . . . 38
4.3.3 Rank Conjecture . . . 39
4.3.4 Collecting Automaton . . . 40
4.3.5 Algorithmic Issues . . . 41
4.3.6 Černý Conjecture . . . 42
4.3.7 Induced Automaton . . . 43
4.3.8 The Literal Automaton of a Finite Prefix Code . . . 45
4.4 Lower Bounds for Properly Incomplete Automata . . . 48

**5 Extremal Bounds for Unambiguous NFAs and General Codes 50**
5.1 Introduction . . . 50
5.2 The result of Kiefer and Mascle . . . 50
5.3 Words of minimum non-zero rank . . . 51
5.4 Prefix automata . . . 53
5.5 Mortality lower bounds for deterministic automata . . . 54
5.6 Codes of full combinatorial rank . . . 57

**Computational Complexity and Algorithms 60**

**6 Weakly Acyclic Automata 60**
6.1 Introduction . . . 60
6.2 Bounds on the Length of Shortest Synchronizing Words . . . 60
6.3 Finding a Synchronizable Set of Maximum Size . . . 62
6.4 Computing the Rank of a Subset of States . . . 66
6.5 Subset Synchronization . . . 68

**7 Monotonic Automata 72**
7.1 Introduction . . . 72
7.2 Structure of Synchronizing Sets . . . 72
7.3 Lower Bounds for Synchronizing Words . . . 74
7.4 Complexity Results . . . 78

**8 Finite Prefix Codes 82**
8.1 Introduction . . . 82
8.2 The Construction of Gawrychowski and Straszak . . . 83
8.3 Acyclic Automata . . . 84
8.4 Huffman Decoders . . . 85

8.5 Partial Huffman Decoders . . . . . 87
8.6 Literal Huffman Decoders . . . . . 89
8.7 Mortal and Avoiding Words . . . . . 90
8.8 Concluding Remarks . . . . . 91

**Conclusions and Further Directions** **93**

**9 Conclusions and Further Directions** **93**

**Bibliography** **94**

# Introduction and Definitions

# Chapter 1

# Introduction

In this chapter we provide an informal introduction to the main objects and concepts of the thesis, and describe their motivation and applications. Formal definitions are given in the next chapter.

## 1.1 Testing of Reactive Systems

> It is more fun to talk with someone who doesn't use long, difficult words but rather short, easy words like "What about lunch?"
>
> "Winnie-the-Pooh" by Alan Alexander Milne

Consider the following simple vending machine which serves coffee. It has only two buttons "yes" and "no" and a screen printing some message for the user. The machine is able to brew just one type of coffee, and can add zero, one or two portions of sugar, and zero or one portions of milk into it. Initially, the machine is in the idle state, waiting for the user to answer the first question. Since there is only one option available, it does not need to ask anything about coffee, so the first question is already "would you like sugar?". If the user presses "yes", the machine asks "would you like more sugar?". If the user presses "no" answering the first question, or presses any button for the second question, the machine asks "would you like milk?", and then provides the user with the beverage according to their commands and goes back to its idle state (asking again about sugar).

The described machine can be modeled by the (complete deterministic) finite state automaton depicted in Figure 1.1. It starts in the "sugar?" state and then changes its state according to the button pressed. During each transition it also does necessary actions (brews coffee, adds sugar or milk) and prints the next question. In our extremely simplified example, it brews coffee as soon as any button is pressed for the first time, so that it does not have to remember any choices. After that it adds sugar or milk as soon as the user answers the corresponding question.

Consider now the situation where we need to test the behavior of the machine and to make sure it coincides with the specification (for example, the one written above). Suppose that we have already ensured that the states and the transitions of the machine are correct (which means that they completely coincide with our specification), but we still need to test the output of the machine, that is, its actions and messages. This can be easily done by applying an input sequence traversing through all possible transitions of the machine and comparing its behavior with the specification letter by letter [21]. However, if we are dealing with a real implementation of the machine, we do not have any "reset" buttons which guarantee that we start the testing procedure from the desired state. If we have a large number of (presumably) identical machines that we want to test at the same time, this problem gets even more complicated. Thus, we need the "specification" machine

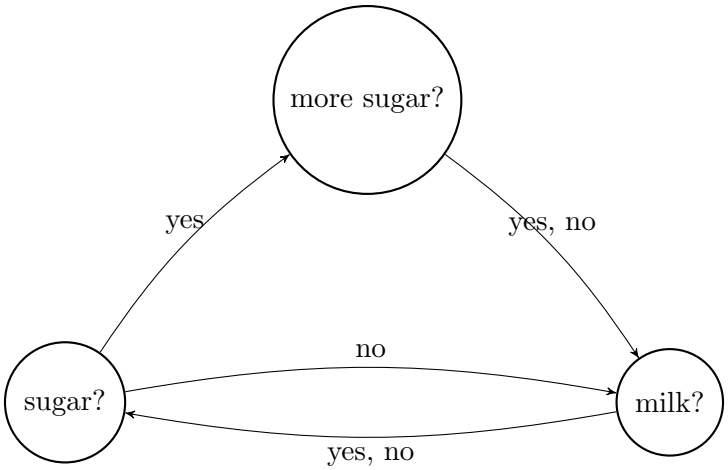


Figure 1.1: The model of a simplified vending machine.

and the machine (or machines) under tests to get to the same state no matter in which state they are at the beginning of testing.

The existence of a reset button, that is, a button sending the machine to some known state, would solve this problem trivially. However, in some cases adding this button is impossible or undesirable. For instance, its absence can decrease the cost of producing such machines or make them safer to use. In fact, in some cases it is impossible to make such a button available to the user, because the machine cannot abandon some process in the middle. However, in many cases there is a *sequence* of commands acting as a reset button. In our example, one can see that applying yes-no-yes brings the machine from each state to the "sugar?" state.

The sequences (or *words* in our terminology) which bring all the states of a finite automaton to one particular state are called *synchronizing*. This thesis is devoted to studying such sequence for various classes of automata, and the connection of these ideas with the theory of variable-length codes and testing of reactive systems (systems designed to work without halting and react to the environment, just like the coffee machine in our example).

The described testing procedure can be applied many times to many machines, so it is important for it to be as effective as possible. This means that we need to use short synchronizing words. In this regard, two important questions concerning the length of shortest synchronizing words arise. The first one is extremal: how long can shortest synchronizing words be? This question is related to the Černý conjecture, one of the oldest open problems in combinatorial automata theory (see Section 2.8 for its discussion). The second one is algorithmic: how hard is it to find a short synchronizing word?. This thesis is devoted to both questions.

## 1.2 Orienting Parts in Manufacturing

> As Navidson indicates on the recorder, he is slowly becoming more and more disoriented. He suffers from surges of nausea, like I've got a bad case of the spins. Questions plague him. Is he floating, falling, or rising? Is he right side up, upside down or on his side? Eventually, however, the spins stop and Navidson accepts that the questions are sadly irrelevant.
>
> "House of Leaves" by Mark Z. Danielewski

The same idea of finding a sequence of events bringing the system to a particular state can be also used in manufacturing. Consider a set of identical polygonal parts arriving on a conveyor belt. Suppose that the parts have a finite number of different orientations (for example, because of the previous operations that were applied to them), and each part can arrive in each of these orientations. The goal is ensure the same orientation for all the parts for further processing. One may put a set of sensors to determine the actual orientation of each part, and then apply necessary rotations individually, depending on the data from sensors. However, sensors are expensive and subject to errors. An alternative solution is to use a system of obstacles which change the position of a part depending on the orientation it has when arrives to this obstacle. The setting can be modeled by a finite automaton, where orientations of a part are states, and the results of encountering the obstacles are transitions. Desired sequences of obstacles then correspond exactly to synchronizing words.

We refer to [37,77,78] for more details, and to [113] for a nice illustrative example. In [119] this problem is considered with additional restrictions on the order in which the transformations can be applied. This setting is motivated, for example, by the presence of lids which must be closed before the end of the process, or which must be hold closed during the whole orienting process. Ideas and results of Chapter 6 are used in the mentioned work.

## 1.3 Error Recovery for Variable-Length Codes

> Despite the constant negative press covfefe
>
> *Donald J. Trump*

Consider now a maximal finite prefix code $X = \{aa, ab, ba, bba, bbb\}$. "Prefix code" here means that no word is a prefix of another word, and "maximal" means that it is maximal by inclusion among sets with this property. Codes like that are widely used for data compression, and the most important example is Huffman codes [54]. The coding and decoding processes are as follows. Suppose that you have a long message over a five-letters source alphabet $x, y, z, t, s$. To each letter you assign a codeword from $X$, and concatenate the codewords in the same order as the letters from the source alphabet in the original message. Then you can transmit this message over a channel. To recover the original message, it is enough to go from left to right and each time a codeword is read, turn it back into the original symbol. Since no codeword is a prefix of another codeword, there can be no other ways to partition the coded message into codewords, so thus obtained message will coincide with the original one.

The code $X$ is a variable-length code, which means that different codewords can have different lengths. This property is important in data compression, since it allows to exploit the information about different frequencies of coded letters: one assigns longer codewords to less frequent letters and shorter codewords to more frequent ones, thus decreasing the length of the coded message. In particular, this is the main idea of constructing Huffman's codes.

However, this property also brings some vulnerabilities. Since not all the lengths are aligned, a single error during the transmission (or just storage) of a message can completely ruin the whole decoding process starting from the codeword containing this error.

Consider the following example of using the code $X$. Let the original message be $yzzyzzy$. It will be coded then as the word in the second row of Figure 1.2. Suppose now that during the transmission, the fourth symbol is received incorrectly and is switched from $a$ to $b$. The resulting decoding of the received word can be seen in the bottom row of the same picture, so after decoding

we will see $ysxtzy$. This message is not only very different from the original one, it also has a different length and has occurrences of symbols which are absent in the original message.

```
y |z |z  |y  |z |z |y
ab|ba|b a|a b|ba|ba|ab
ab|bb b|a a|b ba|ba|ab
y |s  |x  |t  |z |y
```

Figure 1.2: The original coded message and the decoded erroneous message.

In more details, the flip of the fourth symbol cases next four codewords to be read incorrectly. Potentially this problem can propagate arbitrarily long, resulting in a major data because of a single error. To prevent this from happening, one can use, for example, error-correcting codes, which add redundant symbols for recovering up to some number of errors. However, in some situations, for example, for traffic-demanding applications such as video streaming, this is undesirable, and it is preferable to lose some (bounded) amount of information instead of requesting larger capacity from a channel.

Note that for the last two codeword the decoding process returns back to correct reading. This is not a coincidence, but instead a useful feature of the code. The word $abba$ has a special property that no matter what word is before it, any partition of a message into codewords ends right after $abba$. That is, for any word $u$ we have that $uabba$ is a sequence of codewords. Since every word has at most one partition over $X$, any decoding process will treat the position immediately after the occurrence of $abba$ as the end of a codeword. Thus, even if there were some errors before the occurrence of $abba$, the correct reading of this word returns the decoding process back to normal.

A word $w \in X^*$ with the property that $uw \in X^*$ for any $u$ is called *synchronizing* for a maximal prefix code $X$, and maximal prefix codes admitting them are also called *synchronizing*. The presence of a synchronizing word allows to return the decoding process back to the correct one, thus stopping error propagation. Note that in this setting we do not actually care what kind and what number of errors have happened: once the decoder reads a synchronizing word, it is fine again. Moreover, in terms of the required capacity, this property is obtained with adding no redundant data. In many cases, for example, in the Huffman's algorithm, there is some freedom in choosing a particular code. For any maximal finite prefix code there exists a maximal prefix code with the same lengths distribution, provided the greatest common divisor of all codewords lengths is one, which is an obvious requirement [101]. Moreover, almost all maximal finite prefix codes are synchronizing [41].

The introduction of the thesis [15] and the book [14] serve as a great survey on different aspects of constructing, studying and applying synchronizing codes.

There is in fact a clear connection between synchronizing codes and synchronizing automata. Given a maximal finite prefix code, one can build a particular automaton (called *literal* or *prefix automaton*) associated to it. This is done as follows. First, the tree of the code is constructed. For each prefix of a codeword, this tree has a separate node, and the children of a node for a prefix $u$ are nodes for prefixes $ua$, where $a$ is a letter. This includes the case of the empty prefix, which is the root of this tree. Finally, all leaf nodes of this tree are merged with the root node, see Figure 1.3 for an example.

Every synchronizing word for a maximal finite prefix code is then also synchronizing for its literal automaton. Also, every word mapping all the states to the root state of the literal automaton is synchronizing for the initial code. Moreover, the total length of all words of a maximal finite prefix code is polynomially equivalent to the number of states in its literal automaton. This shows a tight connection between synchronizing automata and synchronizing codes, which is used a lot in this

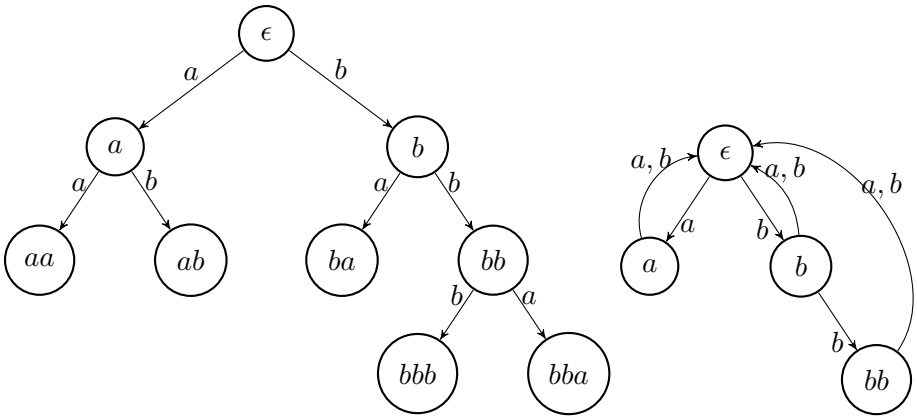


Figure 1.3: The tree of $X$ and its literal automaton.

thesis.

## 1.4 Mortal Words as Resynchronization Markers

> Yes, man is mortal, but that would be only half the trouble. The worst of it is that he's sometimes unexpectedly mortal – there's the trick!
>
> "The Master and Margarita" by Mikhail Bulgakov

A notion closely related to a synchronizing word is a mortal word. Some determiniscti finite automata may have undefined transitions, either due to undefined behavior in some configurations, or because some actions are forbidden in some states. A *mortal word* is a word which, being applied to any state, forces to use an undefined transition while reading this word. Is some sense, synchronizing words bring all states to one state, while mortal words bring all states to the void. These two notions share a number of properties and quite often can be studied by similar techniques.

Here we would like to mention one particular application of mortal words to synchronizing codes. The previous subsection describes the behavior of synchronizing codes which are maximal prefix. However, for non-maximal prefix codes it is also possible to define synchronizing words. A word $w$ is called *synchronizing* for a prefix code $X$ if for any words $u, v$ such that if $uwv$ is a sequence of codewords, then so are $uw, wv$. Hence, we require that if $uwv \in X^*$, then $uw, wv \in X^*$. In particular, this means that $v$ is then also in $X^*$. Hence, if a synchronizing word is seen in a coded message, one can start correct decoding from the position right after this word no matter what happens before it. This is a huge advantage over the usual decoding process of prefix codes, which goes sequentially from left to right starting from the beginning of the message. In our case we can run decoding in parallel for each occurrence of a synchronizing word.

The ability to use non-maximal codes extends the domain from which the codes are chosen. For any non-maximal prefix code there exists a mortal word, that is, a word which cannot occur as in any decodable message. It is easy to modify such a word to make sure that no its prefix coincides with its suffix. This can be done by adding a short word to the end of it [16]. After any such operation the word obviously remains mortal. One can then use this word as a special kind of resynchronization marker. By its construction, no two occurrences of this word can overlap, and this word cannot be a part of the original coded message. It is then easy to adapt the decoding algorithm to handle these markers, and add them with some frequency thus getting a code with very

nice synchronization properties. The algorithm described in [16] finds similar markers depending on the message to be coded. However, the described approach allows to build them universally for all messages by looking only at the code itself. A similar idea lies in the heart of extended synchronizing codewords [67], clear codewords [69] and unique codewords [55].

It makes sense to include resynchronization markers with some constant frequency to ensure that the coded message can be decoded from many positions. Thus, it is once again useful to use short mortal words in this construction. The Restivo's conjecture bounds the length of a shortest mortal word for a finite code in terms of the length of the longest codeword. Thus it can be seen as a more precise variant of the Černý conjecture for mortal words instead of synchronizing. Both these conjectures are discussed in Section 2.8.

## 1.5 Reachability in Automata

> "And I hit everything within reach," cried Tweedledum, "whether I can see it or not!"
>
> "Through the Looking-Glass, and What Alice Found There" by Lewis Carroll

Synchronization and mortality of automata belong to a general class of reachability properties. Indeed, both problems ask whether it is possible to reach a certain (one-state or empty) set of states starting from the whole set of states of the automaton.

Reachability problems are widely studied for various computational models and mathematical structures, including pushdown [20], timed and [1] one-counter [48] automata, vector addition systems [99], Petri nets [74], continuous linear dynamical systems [120], sets of integer or real matrices [80] and many others.

Reachability properties of finite automata are relatively easy to analyze, since in most cases it is enough to traverse the underline graph of the power automaton, the automaton catching the behavior of the subsets of active states in the original one. Thus, such problems are decidable in at most exponential time, and shortest words with required reachability properties also have at most exponential length. This makes such models easier than more complex problems like infinite-state systems or weighted automata. On the one hand, using the results for finite automata as steps in investigating more complex models and problems is very effective. For example, if a problem can be reduced to synchronization of a DFA, it can be solved effectively.

On the other hand, many properties of finite automata are not well-understood, and not because of lack of trying. The Černý conjecture is one of the oldest and well-known problems in automata theory [113], and since eighties no progress on it has been made until a couple of years ago [103,106]. In this thesis we provide a natural generalization of the Černý conjecture to larger classes of automata, first allowing undefined transitions, and then allowing some non-determinism. These models are naturally motivated by their importance in coding theory, but also they are a source of interesting problems, which are sometimes not "blocked" by the requirements of progress in the Černý conjecture, and are therefore have more chances to be solved. Another example is the already mentioned Restivo's conjecture. Though it has been recently disproved in its full generality (that is, for arbitrary finite sets of words [75]), it is still open in the case of arbitrary finite codes. The progress in these conjectures and related questions can be a good source of new insights about reachability in more complex structures.

# Chapter 2

# Main Definitions and Existing Results

Synchronization is a concept in various domains of computer science which consists in regaining control over a system by applying (or observing) a specific set of input instructions. These instructions are usually required to lead the system to a fixed state no matter in which state it was at the beginning. This idea has been studied for automata (deterministic [28,113], non-deterministic [56], unambiguous [7], weighted and timed [33], partially observable [68], register [6], nested word [29]), parts orienting in manufacturing [37,77], testing of reactive systems [100], variable-length codes [14], and Markov Decision Processes [34,35]. In this thesis, we concentrate on two particular and tightly connected cases: finite automata and variable-length codes.

To provide more intuition, we start with definitions concerning synchronization in deterministic finite automata (Section 2.1). After that, in Section 2.2 we provide their generalization to the case of non-deterministic unambiguous finite automata. Then in Section 2.3 we discuss main definitions about codes, and in Section 2.4 we describe the tight relation between synchronization of finite automata and variable-length codes. In Section 2.5 we provide main definitions for two particular classes of automata: monotonic and weakly acyclic. Sections 2.6 and 2.7 discuss two generalizations of the concept of synchronization: subset and careful synchronization. In Section 2.8 we discuss open problems related to bounding the length of words with some particular properties in automata and codes. Section 2.9 surveys known results about the computational complexity of finding such words. Finally, in Section 2.10 we provide the overview of the rest of the thesis and the list of conferences and journals where the results of this thesis were published.

## 2.1 Synchronizing DFAs

A *partial deterministic finite automaton* (partial DFA) is a triple $\mathcal{A} = (Q, \Sigma, \delta)$, where $Q$ is a set of *states*, $\Sigma$ is a finite non-empty *alphabet* and $\delta\colon Q \times \Sigma \to Q$ is a (possibly incomplete) *transition function*. Note that the automata we consider do not have any initial or final states.

We extend $\delta$ to partial functions $Q \times \Sigma^* \to Q$ and $2^Q \times \Sigma^* \to 2^Q$ in the usual way: $\delta(q, w) = \delta(\delta(q, v), a)$ if $w = va$ for some word $v \in \Sigma^*$ and $a \in \Sigma$, and $\delta(S, w) = \{\delta(q, w) \mid q \in S\}$ for $S \subseteq Q$. For a state $q \in Q$ and a word $w \in \Sigma^*$, if the action $\delta(q, w)$ is undefined, then we write $\delta(q, w) = \perp$, and say that $w$ *sends* $q$ *to the void*. Note that if $\delta(q, w) = \perp$ for a word $w \in \Sigma^*$, then $\delta(q, wu) = \perp$ for every word $u \in \Sigma^*$. A DFA is *complete* if all its transitions are defined, and it is *incomplete* otherwise. A DFA over an alphabet of size two is called *binary*, and over an alphabet of size three is called *ternary*.

A word $w$ is called *synchronizing* (or *reset*) for a partial DFA $\mathcal{A} = (Q, \Sigma, \delta)$ if it is defined for at least one state, and brings all such states to one particular state, that is, if $|\delta(Q, w)| = 1$. A

partial DFA admitting a synchronizing word is also called *synchronizing*. The minimum length of a synchronizing word for $\mathcal{A}$ is called the *reset threshold* of $\mathcal{A}$ and is denoted $\mathrm{rt}(\mathcal{A})$.

**Example 1.** Figure 2.1 provides an example of a synchronizing DFA. The word $aba$ is synchronizing for this automaton, since it maps $q_1$ and $q_3$ to the void, and $q_2$ to $q_3$.

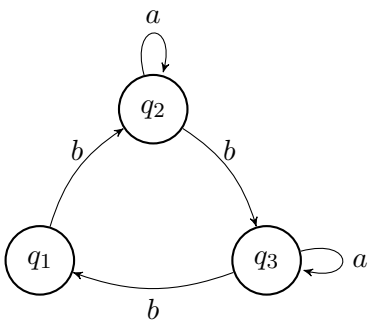


Figure 2.1: A synchronizing automaton.

For complete DFAs, a synchronizing word collects all the states in a single state. Thus, one can apply such a word at every moment and be sure that the automaton is now in a particular state. For partial DFAs the situation is a bit more complicated, since a synchronizing word can be undefined for some states. However, this setting is still very useful for recovering information about the active state. One its application is when the input of a partial DFA is not chosen by the user, but is rather observed from some source which guarantees that no word undefined for all the states is applied. Seeing a synchronizing word in the input then allows to identify the active state of this partial DFA.

A DFA is called *strongly connected* if for every state $p$ and every state $q$ there is a word mapping $p$ to $q$. While for complete DFAs the property of being strongly connected is not essential for many synchronization properties, the situation changes dramatically for partial DFAs. For example, there is a polynomial (in the number of states) upper bound on the length of shortest synchronizing words in complete DFAs, and checking whether a complete DFA is synchronizing can be done in polynomial time [113]. In contrast, partial DFAs which are not strongly connected can have exponentially long shortest synchronizing words, and the problem of checking the existence of a synchronization word for general partial DFAs is PSPACE-complete [11]. However, as discussed in Chapter 4, strongly connected partial DFAs share most of the nice synchronizing bound with complete DFAs.

Reactive systems (such as Web servers, communication protocols, operating systems and processors) are systems developed to run without termination and interact through visible events, so it is natural to assume that the system can return to any state from any other state. The probabilistic version of the synchronization problem for strongly connected partial DFAs has been considered in the context of $\varepsilon$-machines [109]. In particular, the observer knows the state of an $\varepsilon$-machine precisely if and only if a reset word for the underlying partial DFA was applied.

Given a partial DFA $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$, the *rank* of a word $w \in \Sigma^*$ with respect to $\mathcal{A}$ is the number of states active after applying it, that is, the number $|\delta(Q, w)|$. When the DFA is clear from the context, we simply call it the rank of $w$. In contrast to complete DFAs, partial DFAs may admit words of rank zero; these words are called *mortal*. Words of non-zero rank are called *non-mortal*.

The *rank* of a partial DFA is the minimum rank among all non-mortal words with respect to the automaton. A *synchronizing* word (DFA) is thus a word (DFA) of rank 1. We call the length

of a shortest word of minimum rank of an automaton $\mathcal{A}$ the *minimum rank threshold* of $\mathcal{A}$. We denote it $\mathrm{mrt}(\mathcal{A})$.

## 2.2 Synchronizing NFAs

A *non-deterministic finite automaton* (NFA) is a triple $\mathcal{A} = (Q, \Sigma, \Delta)$, where $Q$ is a finite set of *states*, $\Sigma$ is a finite non-empty *alphabet*, and $\Delta$ is a transition relation $Q \times \Sigma \to 2^Q$, where $2^Q$ is the family of all subsets of $Q$. Note that our definition of an automaton does not include any initial or accepting states.

An NFA is called *strongly connected* if for every state $p$ and every state $q$ there is a word $w$ such that $q \in p.w$. As for DFAs, an NFA over an alphabet of size two is called *binary*, and over an alphabet of size three is called *ternary*.

We say that a word $w$ *kills* a state $q$ if $\Delta(q, w)$ is empty, otherwise we say that $q$ *survives* $w$. The transition relation is naturally extended to $2^Q \times \Sigma \to 2^Q$ by taking $\Delta(S, a) = \cup_{q \in S} \Delta(q, a)$ for $S \subseteq Q$ and $a \in \Sigma$, and then to $Q \times \Sigma^* \to 2^Q$ by taking $\Delta(q, wa) = \Delta(\Delta(q, w), a)$ for $w \in \Sigma^*, a \in \Sigma$ and $q \in Q$. When the transition relation is clear from the context, we denote $\Delta(q, w)$ by $q.w$. We also denote by $w.q$ the set $\{p \mid q \in p.w\}$. More generally, we denote by $S.w$ and $w.S$ the sets $\{q.w \mid q \in S\}$ and $\{w.q \mid q \in S\}$ for $S \subseteq Q$, $w \in \Sigma^*$. An NFA is *unambiguous* if for any two states $p, q \in Q$ and any word $w \in \Sigma^*$ there is at most one path from $p$ to $q$ labeled by $w$.

Recall that an NFA is *deterministic* if for every state $q \in Q$ and every letter $a \in \Sigma$ the set $\Delta(q, w)$ has cardinality at most one. Clearly, every deterministic finite automaton is unambiguous.

Provided a word $w \in \Sigma^*$ for an NFA $\mathcal{A} = (Q, \Sigma, \Delta)$, one can naturally assign a matrix $M(w)$ to it as follows. Fix some ordering $q_1, \ldots, q_n$ of the set $Q$ of states. The entry $M(w)_{i,j}$ of the matrix $M(w)$ is the number of paths from $q_i$ to $q_j$ labeled by $w$. As follows from the definition, each entry of such a matrix for an unambiguous NFA belongs to the set $\{0, 1\}$. For the mapping $M : \Sigma^* \to \{0, 1\}^{n \times n}$ for any words $w_1, w_2 \in \Sigma^*$ we have $M(w_1 w_2) = M(w_1)M(w_2)$. We denote $\mathcal{M}(\mathcal{A}) = \{M(w) \mid w \in \Sigma^*\}$.

Using this morphism, we define the *rank* of a word $w$ in an unambiguous NFA $\mathcal{A}$ as the Boolean rank of the matrix $M(w)$ [7]. By Boolean rank we mean the rank over the Boolean semiring $\{0, 1\}$ (having $1 + 1 = 1$), that is, the smallest $r$ such that $M(w)$ can be represented as a product of an $n \times r$ and $r \times n$ zero-one matrices. Given an NFA $\mathcal{A}$, we call a word *synchronizing* if it has rank 1, and *mortal* if it has rank 0. An NFA is called *non-complete* if it admits a mortal word, otherwise it is called *complete*. An NFA admitting a synchronizing word is called *synchronizing*. The paper [7] surveys some questions on the structure of unambiguous automata and words of particular rank for them.

**Example 2.** Figure 2.2 provides an example of a synchronizing unambiguous NFA. The matrix of the word $aa$ is

$$M(aa) = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Thus this word is synchronizing.

It follows from the definition that a word $w$ is synchronizing for an unambiguous NFA if and only if there exist two non-empty subsets $C, R$ of its states such that the action of $w$ maps every state of $C$ to the set $R$, and is undefined for all the states outside $C$. For partial DFAs, the set $R$ has size one, and, for complete DFAs, the set $C$ is the whole set of states. In Example 2 the

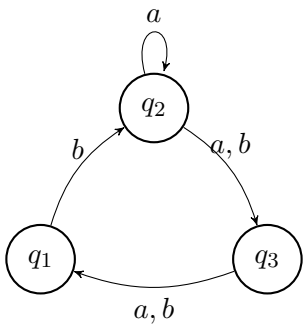


Figure 2.2: A synchronizing NFA.

state $q_1$ is mapped by a synchronizing word $aa$ to the states $q_1, q_2, q_3$, while the states $q_2$ and $q_3$ are mapped to the void.

## 2.3 Codes

Let $\Sigma$ be a finite non-empty *alphabet*, consisting of *letters*. A *code* $X$ over $\Sigma$ is a set of words over $\Sigma$ (called *codewords*) such that no word over $\Sigma$ can be represented as a concatenation of codewords from $X$ in two different ways. A word $w$ is called a *factor* of $w'$ if there exist words $u, v$ such that $w' = uwv$. A word $w \in \Sigma^*$ is called *non-mortal* for a code $X$ over $\Sigma$ if it is a factor of a word in $X^*$, otherwise it is called *mortal*. A code is called *complete* if it does not admit a mortal word. Thus, complete codes are exactly the codes such that every word is a factor of a decodable message.

**Example 3.** The set $\{aba, ab, ba\}$ is not a code, since $(aba)(ba) = (ab)(aba)$. The set $\{aba, a\}$ is a code. It is not maximal, since $bb$ is not a factor of any sequence of codewords. The set $\{a, ab, bb\}$ is a complete code.

Since the lengths of codewords can be different, one transmission error can make the whole decoding process incorrect, causing a major data loss. Also, decoding a part of a message (e.g., a segment of a compressed video stream) is not possible without decoding the whole message. These issues can be addressed by using synchronizing codes. A word $w \in X^*$ is called *synchronizing* for a code $X$ if for every pair of words $u, v$ such that $uwv \in X^*$ we have $uw, wv \in X^*$. A code admitting a synchronizing word is also called *synchronizing*. If the word $w$ is synchronizing, then seeing a word $ww$ in a decodable message $uwwv$ allows to separate the decoding process into two independent parts for $uw$ and $wv$. A presence of a word $ww$ in a message allows to restart the decoding process after transmission errors or when a part of a message is skipped.

**Example 4.** The code $\{a, ab, bb\}$ is synchronizing, and the word $a$ is synchronizing for it. The code $\{a, aba\}$ is synchronizing with a synchronizing word $aba$. The code $\{a, b\}^2$ is not synchronizing.

A code is called *prefix* if none of its codewords is a prefix of another codeword. Such codes allow obtaining the correct partition of a message into codewords one by one by going from left to right.

Even if a code is synchronizing, there are no guarantees that a synchronizing word will appear in a message. Codes where every long enough concatenation of codewords is synchronizing are called *uniformly synchronizing* [14, 22]. A prefix code is called *maximal* if it is not a subset of another prefix code. All non-trivial uniformly synchronizing finite prefix codes are non-maximal [14]. A recognizable (by some NFA) code is complete if and only if it is maximal [14].

**Example 5.** The set $\{aba, a\}$ is not a prefix code, since $a$ is the prefix of $aba$. The set

$$\{aa, ab, baa, bab, bba, bbb\}$$

is a finite maximal prefix code.

## 2.4 Automata for Codes

Automata serve as a very powerful tool for studying codes. Let $\mathcal{A}$ be a strongly connected unambiguous NFA, and $q$ be some its state. The set of *first return words* of $q$ is then defined as the set of all words labeling paths from $q$ to $q$ and not containing $q$ as an intermediate state. For a strongly connected unambiguous NFA $\mathcal{A}$ and a state $q$ in it, the set of first return words is always a recognizable (by some NFA) code [14]. The converse is also true: for every recognizable code $X$ there is a strongly connected unambiguous NFA $\mathcal{A}$ such that $X$ is the set of first return words of some state of $\mathcal{A}$ [14, Chapter 4].

Moreover, every mortal (synchronizing) word for $X$ is also mortal (synchronizing) for $\mathcal{A}$. The converse is true for mortal words. Provided a synchronizing word $w$ for $\mathcal{A}$, there are words $u, v$ of length at most $n$ each such that $uwv$ is synchronizing for $X$, where $n$ is the number of states of $\mathcal{A}$. This follows from the fact that synchronizing words for codes are in fact synchronizing words for corresponding automata with the additional property that the state $q$ must be mapped to itself. More generally, the minimum non-zero rank of words for $\mathcal{A}$ is related to the minimum number of interpretations of words with respect to $X$ (called the degree of $X$, see Section 9.6 of [14]).

The described correspondence between strongly connected NFAs and recognizable codes can be made more precise for special classes of codes. In particular, recognizable codes which are, respectively, maximal, prefix and maximal prefix are exactly the families of first return words of strongly connected unambiguous NFAs which are, respectively, complete, deterministic and deterministic complete .

If a state $q$ can be picked in a DFA in such a way that the set of all first return words is a finite prefix code, we call the DFA a *partial Huffman decoder*. If such a DFA is complete (and thus the finite prefix code is maximal), we call it simply a *Huffman decoder*.

**Example 6.** Consider the maximal finite code $\{aa, ab, baa, bab, bba, bbb\}$. It is the set of first return words of the state $q_1$ in the complete DFA from Figure 2.3. This DFA is thus a Huffman decoder.

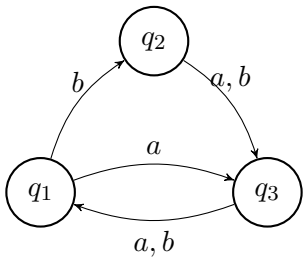


Figure 2.3: A Huffman decoder.

For finite codes, this construction can be further simplified. Let $X$ be a finite code over an alphabet $\Sigma$. One can then construct the *prefix automaton* $\mathcal{A} = (Q, \Sigma, \Delta)$ of $X$ as follows. The states of $\mathcal{A}$ are the proper prefixes of words in $X$. For a state $q \in \Sigma^*$ and a letter $a \in \Sigma$ the set $\Delta(q, a)$ contains $qa$ if $qa$ is a proper prefix of a word in $X$ and contains $\epsilon$ if $qa$ is a word in $X$

(these situations can occur at the same time). It is easy to see that the prefix automaton of a finite code is always strongly connected and unambiguous. The number of its states provides a lower bound on the total length of all words in the code. The prefix automaton of a prefix code is always deterministic, and is also sometimes called its *literal* automaton.

## 2.5 Monotonic and Weakly Acyclic Automata

A complete DFA $\mathcal{A} = (Q, \Sigma, \delta)$ is called *monotonic* if there is a linear order $\leq$ of its states such that for each $x \in \Sigma$ if $q_1 \leq q_2$ then $\delta(q_1, x) \leq \delta(q_2, x)$. In this case we say that the transitions of the automaton *preserve*, or *respect* this order. Monotonic automata play an important role in the part-orienting process in manufacturing [3], some connections of monotonic automata with infinite games are described in [64] and [65]. Checking that a complete DFA is monotonic is NP-complete even for binary automata [105]. Once the order $q_1, \ldots, q_n$ of the states is fixed, we denote $[q_i, q_j] = \{q_\ell \mid i \leq \ell \leq j\}$, and $\min S$, $\max S$ as the minimum and maximum states of $S \subseteq Q$ with respect to the order. The following open problem is mentioned in [102], showing that monotonic automata are not fully understood, and require more investigation.

**Question 7.** *Find a combinatorial characterization (for example, using regular expressions) of languages recognized by monotonic automata.*

For $n$-state monotonic complete DFAs of rank at most $r$, Ananichev and Volkov [3] show an upper bound of $n - r$ on the length of a shortest word of rank at most $r$, and also provide bounds on the length of a shortest word of so-called interval rank at most $r$. Shcherbak [102] continues the investigation of words of bounded interval rank in monotonic automata. Ananichev [4] provides bounds on the length of a shortest mortal word in partial monotonic DFAs (a partial DFA is called *monotonic* if there exists a linear order of its states such that each letter preserves the restriction of the order to the domain of the respective transformation).

A *simple cycle* in a partial DFA $\mathcal{A} = (Q, \Sigma, \delta)$ is a sequence $q_1, \ldots, q_k$ of its states such that all the states in the sequence are different and there exist letters $x_1, \ldots, x_k \in \Sigma$ such that $\delta(q_i, x_i) = q_{i+1}$ for $1 \leq i \leq k-1$ and $\delta(q_k, x_k) = q_1$. A simple cycle is a *self-loop* if it consists of only one state. A partial DFA is called *weakly acyclic* if all its simple cycles are self-loops. In other words, a partial DFA is weakly acyclic if and only if there exists an ordering $q_1, q_2, \ldots, q_n$ of its states such that if $\delta(q_i, x) = q_j$ for some letter $x \in \Sigma$, then $i \leq j$ (such ordering is called a *topological sort*). Since a topological sort can be found in polynomial time [31], this class can be recognized in polynomial time. Weakly acyclic complete DFAs are called acyclic in [57] and partially ordered in [23], where the class of languages recognized by such automata is characterized. A partial DFA is called *strongly acyclic* if it is weakly acyclic and has self-loops only in sink states

Weakly acyclic DFAs arise naturally in synchronizing automata theory. For example, the complete DFAs in the reductions showing the hardness of the problem of finding a shortest synchronizing word of Eppstein [37] and Berlinkov [11] are weakly acyclic. Surprisingly, most of the computational problems that are hard for general DFAs remain very hard in this class despite its very simple structure. Thus, investigation of weakly acyclic DFAs provides good lower bounds on the complexity of many problems for general automata. A complete DFA is called *aperiodic* if for any word $w \in \Sigma^*$ and any state $q \in Q$ there exists $k$ such that $\delta(q, w^k) = \delta(q, w^{k+1})$, where $w^k$ is a word obtained by $k$ concatenations of $w$ [108]. Obviously, weakly acyclic complete DFAs form a proper subclass of aperiodic complete DFAs, thus all hardness results hold for the class of aperiodic complete DFAs.

Note that a monotonic DFA does not have to be weakly acyclic, and vice versa. Monotonic complete DFAs also present a proper subclass of aperiodic complete DFAs [113]. A complete DFA

is called *orientable* if there exists a cyclic order of its states that is preserved by all the transitions of the automaton (see [113] for the discussion of this definition). Each monotonic complete DFA is obviously orientable.

## 2.6 Subset Synchronization

Synchronizing DFAs model devices that can be reset, by applying a synchronizing word, to some particular state without having any information about their current state. However, even if an automaton is not synchronizing, some partial information about its current state may be available, namely that the current state belongs to a particular set of states. A set $S \subseteq Q$ of states of a complete DFA $\mathcal{A} = (Q, \Sigma, \delta)$ is called *synchronizable* if there exists a word $w \in \Sigma^*$ and a state $q \in Q$ such that the word $w$ maps each state $s \in S$ to the state $q$. The word $w$ is then said to *synchronize* the set $S$. It follows from the definition that a complete DFA is synchronizing if and only if the set $Q$ of all its states is synchronizable.

The behavior of words synchronizing a subset of states is quite different from the behavior of words synchronizing the whole automaton. It is known that the length of a shortest word synchronizing a subset of states in a binary strongly connected complete DFA can be exponential in the number of states of the automaton [114]. For orientable $n$-state complete DFAs, a tight $(n-1)^2$ upper bound on the length of a shortest word synchronizing a subset of states is known [37]. In the general case, a trivial upper bound $2^n - n - 1$ on the length of a shortest word synchronizing a subset of states in an $n$-state complete DFA is known [114]. In [24] Cardoso considers the length of a shortest word synchronizing a subset of states in a synchronizing complete DFA.

Consider the problem Sync Set of deciding whether a given set $S$ of states of a complete DFA $\mathcal{A}$ is synchronizable.

> Sync Set
> *Input*: A complete DFA $\mathcal{A}$ and a subset $S$ of its states;
> *Output*: Yes if $S$ is a synchronizable set, No otherwise.

The Sync Set problem is PSPACE-complete [89, 100], even for binary strongly connected complete DFAs [114]. In [77] it is shown that the Sync Set problem is solvable in polynomial time for orientable complete DFAs if the cyclic order respected by the automaton is provided in the input. The problem of deciding whether the whole set of states of a complete DFA is synchronizable is also solvable in polynomial time [113].

Eppstein [37] provides a polynomial algorithm for the Sync Set problem, as well as for some other problems, in orientable complete DFAs. However, the proposed algorithms assume that a cyclic order of the states preserved by the transitions is known. Since the problems of recognizing monotonic and orientable complete DFAs are NP-complete [105], a linear or cyclic order preserved by the transitions of an automaton cannot be computed in polynomial time unless P = NP.

A synchronizable set of states can be considered as a set compressible to one element. A more general case of a set compressible to a set of size $r$ is defined by the notion of the rank of a subset. Given an complete DFA $\mathcal{A} = (Q, \Sigma, \delta)$, the *rank* of a word $w \in \Sigma^*$ with respect to a set $S \subseteq Q$ is the size of the image of $S$ under the mapping defined by $w$ in $\mathcal{A}$, i.e., the number $|\{\delta(s, w) \mid s \in S\}|$. The *rank* of a subset $S$ of states is the minimum among the ranks of all words $w \in \Sigma^*$ with respect $S$. It follows from the definition that a set of states has rank 1 if and only if it is synchronizable.

We define the Set Rank problem as follows.

> Set Rank
> *Input*: A DFA $\mathcal{A}$ and a set $S$ of its states;
> *Output*: The rank of $S$.

## 2.7 Careful Synchronization

A problem closely connected to subset synchronization is careful synchronization of partial DFAs. A word $w$ is said to *carefully synchronize* a partial DFA $\mathcal{A}$ if it maps all its states to the same state. In particular, it means that the mapping corresponding to $w$ is defined for every state. The automaton $\mathcal{A}$ is then called *carefully synchronizing*. A carefully synchronizing word can be applied to a partial DFA at any moment without the risk of using an undefined transition. This comes at a high cost: even for strongly connected partial DFAs, the shortest carefully synchronizing words can have exponential length [114, Proposition 9], and the problem of checking the existence of such a word is PSPACE-complete [114, Theorem 12], in contrast with the case of complete DFAs. On the contrary, the notion of a synchronizing partial DFA preserves most of the properties of a synchronizing complete DFA, at least in the strongly connected case. Note that every carefully synchronizing word is synchronizing, but the converse is not true.

Careful Synchronization
*Input*: A partial DFA $\mathcal{A}$;
*Output*: Yes if $\mathcal{A}$ is carefully synchronizing, No otherwise.

The length of a shortest word carefully synchronizing an $n$-state partial DFA is also a subject of research. Rystsov [88], Martyugin [72], Vorel [114] and de Bondt et al. [32] propose consecutive improvements of exponential lower bounds for this value, both in the case of constant and non-constant alphabets. Rystsov [88] provides an upper bound of $O(3^{\frac{n}{3}})$ on this value. A simple relation between careful synchronization and subset synchronization is provided by Lemma 1 of [114].

Deciding whether a partial automaton is carefully synchronizing is PSPACE-complete for binary partial DFAs [71], and moreover for binary strongly connected partial DFAs [114].

For general NFAs, synchronizability can be generalized to D$i$-directability for $i = 1, 2, 3$ [56]. As discussed in [114, Section 6.3], for partial DFAs the notions of D1- and D3-directing words both coincide with *carefully synchronizing* words. These are words sending every state of a partial DFA to the same state, not using any undefined transitions at all. A D2-directing word for a partial DFA is either carefully synchronizing or mortal (undefined for every state).

## 2.8 Černý and Restivo's Conjectures

A natural question arises: how large can the reset threshold of an $n$-state synchronizing automaton be? In 1964 Černý [28] constructed an $n$-state binary synchronizing automaton $\mathcal{C}_n$ which reset threshold is $(n-1)^2$ for all $n > 1$. The state set of $\mathcal{C}_n$ is $Q = \{1, 2, \ldots, n\}$ and the letters $a$ and $b$ act on it as follows:

$$\delta(i,a) = \begin{cases} i, & \text{if } i > 1 \\ 2, & \text{if } i = 1; \end{cases} \quad \delta(i,b) = \begin{cases} i+1, & \text{if } i < n \\ 1, & \text{if } i = n. \end{cases}$$

We refer to automata of this series as *the Černý automata*. Figure 2.4 shows $\mathcal{C}_3$.

Some time later (e.g. [28]) it was conjectured that every synchronizing automaton with $n$ states can be synchronized by a word of length $(n-1)^2$. This is known as *the Černý Conjecture* which remains open more than 50 years later (for a survey on this topic see [113]). The best known upper bound is cubic in $n$ [103, 106].

The Černý conjecture can be generalized to a much more general setting. Let $\mathcal{A}$ be an $n$-state strongly connected unambiguous NFA of rank $r$. Carpi proved an upper bound of $\frac{1}{2}rn^3$ on the length of a shortest word of minimum rank if $\mathcal{A}$ is complete [25]. Kiefer and Mascle extended his result and proved an upper bound of $n^5$ on the length of a shortest mortal word if $\mathcal{A}$ is non-complete [62].

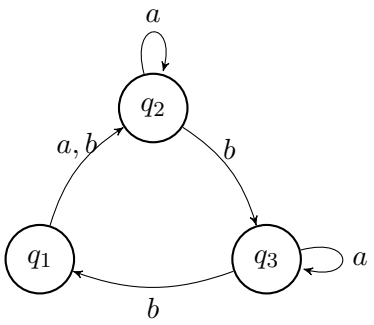


Figure 2.4: A synchronizing automaton.

Note that the Černý automata provide the best known lower bound even on the length of shortest synchronizing words even for the case of strongly connected unambiguous NFAs. This shows that the Černý conjecture can be broadened to strongly connected unambiguous NFAs [14].

Provided a finite set $X$ of words, one can also argue about the length of a shortest word of some particular rank in terms of the length $k$ of the longest codeword in $X$. The conjecture proposed by Restivo [87] stays that every non-complete code has a mortal word of length at most $2k^2$. In its full form, this lower bound was first pushed to $ck^2$ for a constant $c > 2$ [39, 47], and then completely disproved by showing an exponential lower bound [75]. However, the case where $X$ is a code still remains open. No polynomial upper bound is known in this case, and the best known lower bound is $k^2 + k - 1$, provided by a simple prefix code [86].

## 2.9 Finding Shortest Synchronizing Words

We refer to the book by Sipser [104] for the notions related to NP-completeness, and to the book by Vazirani [112] for the notions of an approximation algorithm, gap-preserving reduction and inapproximability.

We consider the following computational problem.

> Short Sync Word
> *Input*: A synchronizing partial DFA $A$;
> *Output*: The length of a shortest synchronizing word for $A$.

We shortly survey existing results and techniques in the computational complexity and approximability of finding shortest synchronizing words for complete automata. See [104] for an introduction to NP-completeness and [112] for an introduction to inapproximability and gap-preserving reductions.

There exist several techniques of proving that Short Sync Word is hard for different classes of automata. The very first and the most widely used idea is the one of Eppstein [37]. Here, the automaton in the reduction (from some variant of the satisfiability problem) is composed of a set of "pipes", and transitions define the way the active states are changed inside the pipes to reach the sink state where synchronization takes place. This idea, sometimes extended a lot, allows to prove NP-hardness of the decision version of Short Sync Word in the classes of strongly acyclic [37], ternary Eulerian [73], binary Eulerian [115], and binary circular [73] automata. It is also used to prove inapproximability within an arbitrary constant factor for binary automata [10] and for $n^{1-\varepsilon}$-inapproximability for $n$-state binary automata [43] (the last proof uses the theory of Probabilistically Checkable Proofs). In fact, the proof in [43] works for binary automata with at most linear (in the number of states of the automaton) length of a shortest synchronizing word and

a unique sink state.

Another idea is to construct a reduction from the Set Cover problem. It was used to show logarithmic inapproximability of Short Sync Word in weakly acyclic [44] and binary automata [11]. Finally, a reduction from Shortest Common Supersequence provides inapproximability of this problem within a constant factor [44].

In the class of monotonic automata, Short Sync Word is solvable in polynomial time: because of the structure of these automata, this problem reduces to a problem of finding a shortest word synchronizing a pair of states [97]. For general $n$-state automata, a $\lceil \frac{n-1}{k-1} \rceil$-approximation polynomial-time algorithm exists for every $k$ [44].

## 2.10 Organization of the Thesis

The rest of the thesis is organized as follows. Chapters 3, 4 and 5 are devoted to studying bounds on the length of shortest words of some particular rank in finite automata. This includes mortal and synchronizing words as particular examples. Hence these results are related to the Černý and Restivo's conjectures.

Chapter 3 is devoted to words of minimum rank in complete DFAs. The results of this chapter were obtained in collaboration with Jarkko Kari and Anton Varonka, and published in [61]. Chapter 4 studies shortest synchronizing words in strongly connected partial DFAs. The results of this chapter were obtained in collaboration with Mikhail V. Berlinkov, Robert Ferens and Marek Szykuła, and are prepared for submission. Chapter 5 provides results on the length of shortest words of minimum and minimum non-zero rank in strongly connected unambiguous NFAs. The results of this chapter are published in [93].

In Chapters 6, 7 and 8 we study the computational complexity side of the mentioned settings.

Chapter 6 studies the complexity of several problems related to synchronization of weakly acyclic complete DFAs. The results of this chapter are published in [95], with a preliminary version at CIAA 2017 [92]. Chapter 7 is devoted to the same questions in monotonic complete and partial DFAs. The results of this chapter were obtained in collaboration with Anton Shemyakov, and are published in [97], with a preliminary version at RuFiDiM 2017 [96]. Chapter 8 considers the complexity of synchronizing DFAs for finite prefix codes. The results of this chapter were obtained in collaboration with Marek Szykuła, and are published in [98].

A part of this thesis (namely, preliminary versions of Chapters 6 and 7) constituted a part of the authors master's thesis, defended at University Grenoble Alpes in 2017.

Finally, we would like to mention several work which were done during the PhD studies of the author, but were not included in this thesis. Decomposition of finite sets and its relation to their degree is considered in the paper [81] in collaboration with Dominique Perrin. This paper is currently submitted. The computational complexity of problems related to birecurrent sets and automata recognizing them are published in [94]. Some results on several extremal problems in combinatorics on words are published in [76] in collaboration with Clemens Müllner .

# Extremal Bounds

# Chapter 3

# Extremal Bounds for Complete DFAs

All automata in this chapter are complete DFAs, so we simply write "DFA" or "automaton" for a "complete DFA".

## 3.1 Introduction

Pin [84] proposed the following generalization of the Černý Conjecture: for every $n$-state DFA having a word of rank at most $r$, there exists such a word of length at most $(n-r)^2$. A cubic upper bound is proved for this conjecture [63]. However, Kari [59] found a counterexample to the conjectured $(n-r)^2$ bound for $r=2$, which is a binary DFA $\mathcal{K}$ with $n=6$ states. As a consequence, a modification of this generalized conjecture was proposed by Pribavkina restricting it to $r$ being the rank of the considered DFA ($\mathcal{K}$ is synchronizing but the Pin's bound is exceeded for a word of rank 2). This restricted case has not been disproved yet, and is sometimes referred to as the Rank Conjecture (or the Černý-Pin Conjecture in [2]). The case $r=1$ is the Černý Conjecture.

It was pointed out in [5] that one of the reasons why the Černý Conjecture is so hard to tackle is the lack of examples of slowly synchronizing DFAs. The same is true concerning the Rank Conjecture. Pin [83] provided the following example. The DFA with two letters consists of $r$ connected components, one of which is the Černý automaton $\mathcal{C}_{n-r+1}$ and $r-1$ others are isolated states with loops labeled with both letters. The DFA thus constructed has $n$ states, rank $r$ and its minimum rank threshold is precisely $(n-r)^2$. However, this DFA is not strongly connected (recall that a DFA is called *strongly connected* if any state can be mapped to any other state by some word), so this case in some sense reduces to the rank 1 case. No series of strongly connected DFAs with $\mathrm{mrt}(\mathcal{A})$ close to the $(n-r)^2$ bound were introduced so far.

In this chapter, we propose a number of techniques to construct strongly connected DFAs of rank $r$ with large minimum rank thresholds. The families of DFAs we obtain do not reach the conjectured bound $(n-r)^2$, but the minimum rank threshold is typically of the order $\frac{(n-r)^2}{r}$, or within a constant multiple of this. We provide families of DFAs having additional properties such as being Eulerian or circular, or having rank equal to the period (see Section 3.2 for definitions of these concepts). We also consider upper bounds: we prove the Rank Conjecture for Eulerian DFAs, and obtain an upper bound on the minimum rank threshold of circular DFAs.

The chapter is organized as follows. In Section 3.2 we provide the main definitions and preliminary results. In Section 3.3 we provide constructions for turning a binary synchronizing DFA into a higher rank ternary (Section 3.3.1) or binary (Section 3.3.2) DFA having its minimum rank threshold close to the reset threshold of the original DFA. Applying these constructions on known series of synchronizing automata yield new series of DFAs of higher ranks $r>1$. In Section 3.3.3 we

show how upper bounds on the reset threshold can be turned into upper bounds on the minimum rank thresholds on DFAs with period equal to rank. In Section 3.4 we prove the Rank Conjecture for DFAs based on Eulerian digraphs, along with exhibiting lower bounds on minimum rank thresholds. In Section 3.4.2 we present a way to transform known bounds from Eulerian DFAs to circular DFAs. In particular, quadratic upper bounds on minimum rank thresholds for circular DFAs (including the reset threshold) are proved. In Section 3.4.3 we contribute to the Road Coloring Problem, presenting a nearly-linear algorithm of finding a coloring of minimum rank for an Eulerian digraph.

## 3.2 Digraphs and Automata

All our digraphs are multigraphs and they are allowed to have loops. The *underlying digraph* $D(\mathcal{A})$ of a DFA $\mathcal{A} = (Q, \Sigma, \delta)$ has vertex set $Q$, and for any $q, p \in Q$, there are as many edges from $q$ to $p$ as there are letters $a \in \Sigma$ such that $\delta(q, a) = p$. A DFA $\mathcal{A}$ is called a *coloring* of its multigraph $D(\mathcal{A})$. The underlying digraph of every DFA has the same outdegree at all its vertices. From now on, we consider only digraphs with this property.

A digraph $D$ is called *strongly connected* if for every pair $(v, v')$ of vertices there exists a directed path from $v$ to $v'$. Thus a DFA is *strongly connected* if and only if its underlying digraph is strongly connected.

The *period* of a digraph $D$ is the greatest common divisor of the lengths of its cycles, and the period of a DFA is defined as the period of its underlying digraph. Let us remark explicitly that digraphs with period $p > 1$ do not have synchronizing colorings. The following lemma is essential to understand the period of a digraph.

**Lemma 8** ( [13], p.29)**.** *Let $D$ be a digraph with period $p$. Then the set $V$ of vertices of $D$ can be partitioned into $p$ nonempty sets $V_1, V_2, \ldots, V_p$ where each edge of $D$ goes from a vertex from $V_i$ and enters some vertex in $V_{i+1}$ for some $i$ (the indices are taken modulo $p$).*

We will call this partition a *$p$-partition* of a digraph or of its coloring.

Much of the literature on synchronizing automata concentrates on the primitive case. A digraph is called *primitive* if it is strongly connected and the period is $p = 1$. In this chapter we are interested in DFAs with underlying digraphs which are strongly connected but not primitive.

A digraph is *Eulerian* if for each vertex the outdegree is equal to the indegree. A DFA is *Eulerian* if it is strongly connected and its underlying digraph is Eulerian. Equivalently, at every state there must be exactly $|\Sigma|$ incoming transitions, where $\Sigma$ is the alphabet of the automaton. A DFA is *circular* if there is a letter which acts on its set of states as a cyclic permutation.

## 3.3 Strongly Connected Automata

### 3.3.1 A Lower Bound for Ternary Automata

We start with a construction yielding a series of strongly connected ternary automata. We transform a synchronizing binary automaton $\mathcal{A}$ into a ternary automaton $\mathcal{A}'$ of a given rank $r > 1$ such that $\mathrm{mrt}(\mathcal{A}')$ is related to $\mathrm{rt}(\mathcal{A})$.

We start with a synchronizing binary automaton $\mathcal{A} = (Q, \{a, b\}, \delta)$ with $t$ states $q_1, \ldots, q_t$. We define a ternary automaton $\mathcal{A}' = (Q', \{a, b, c\}, \delta')$ of rank $r$ with the size $n = r \cdot t$ state set $Q' = \bigcup_{i=0}^{r-1} Q_i$ where each $Q_i$ contains $t$ states $q_{i,1}, \ldots, q_{i,t}$. The action of the transition function $\delta'$ on the set $Q_0$ repeats the action of $\delta$ on set $Q$ for the letters $a, b$: for $x = a$ and $x = b$ we have

$\delta'(q_{0,j}, x) = q_{0,k}$ if and only if $\delta(q_j, x) = q_k$. On the other sets $Q_1, \ldots, Q_{r-1}$ the transitions by the letters $a, b$ are self-loops: we set $\delta'(q_{i,k}, x) = q_{i,k}$ for $x = a$ and $x = b$, for all $i \neq 0$ and all $k$. Finally, the letter $c$ shifts states of $Q_i$ to the next set $Q_{i+1}$: we define $\delta'(q_{i,k}, c) = q_{i+1,k}$ where $i + 1$ is counted modulo $r$, that is, elements of $Q_{r-1}$ are shifted to the set $Q_0$. Note that the construction preserves the property of the automaton to be strongly connected or Eulerian.

Since $\mathcal{A}$ is synchronizing, we certainly obtain an automaton of rank $r$ as the result of this construction. No two states from different sets $Q_i, Q_j$ with $i \neq j$ can be merged for the obvious reason. Each of them though can be mapped using the letter $c$ to $Q_0$ which, in turn, can be mapped to a single state.

If $w$ is a shortest reset word for $\mathcal{A}$, a trivial way to compose a word of rank $r$ for $\mathcal{A}'$ is as follows. We use $w$ to merge the states of $Q_0$ to one particular state, then use the letter $c$ to shift the set at play and continue until every set $Q_i$ is merged into one state. The resulting word $w' = wcw \ldots cw$ thus has length $\mathrm{rt}(\mathcal{A}) \cdot r + r - 1$. Moreover, $w'$ is the shortest word of rank $r$. Indeed, since all the transitions in the sets $Q_1, Q_2, \ldots, Q_{r-1}$ are self-loops for $a, b$, the only place where merging of states takes place is inside $Q_0$. While states of some $Q_i$ are treated there, the states of all $Q_j, j \neq i$, remain invariant. Obviously, $c$ has to be applied at least $r - 1$ times. Hence, by the pigeonhole principle, the existence of a shorter word of minimum rank would imply that an automaton induced by the action of $\{a, b\}$ on $Q_0$ can be synchronized faster than in $\mathrm{rt}(\mathcal{A})$ steps.

If we apply the construction to the Černý automaton $\mathcal{C}_{\frac{n}{r}}$, we get the following.

**Proposition 9.** *For every $n$ and every $r > 1$ such that $r$ divides $n$, there exists a ternary strongly connected automaton with $n$ states and rank $r$ such that the length of its shortest word of minimum rank is $\frac{(n-r)^2}{r} + r - 1$.*

It is natural to ask for a lower bound on the minimum rank threshold for binary automata. There are some techniques known to decrease the alphabet size of an automaton while not changing the length of a shortest synchronizing word significantly. By carefully applying the construction encoding letters in states [11, 114] one can get a lower bound of $\frac{n^2}{3r} - \frac{7}{3}n + 5r$ for the binary case. Another technique decreasing alphabet size, namely by encoding binary representation of letters in states [11, Lemma 3], does not yield any better bounds. Below we present some different ideas providing stronger lower bounds on $\mathrm{mrt}(\mathcal{A})$ in the class of binary strongly connected automata.

### 3.3.2 Lower Bounds for Binary Automata

In the ternary construction above we may represent the actions of words $ac$ and $bc$ by two new letters, and afterwards remove the original letters $a, b, c$. This yields a binary automaton of rank $r$. More generally, we can do this on the analogous construction from an automaton with alphabet size $k$ to size $k + 1$, obtaining again an automaton with alphabet size $k$ and having rank $r$.

The detailed construction goes as follows. Given a strongly connected synchronizing automaton $\mathcal{A} = (Q, \Sigma, \delta)$ over any alphabet $\Sigma$ and with state set $Q = \{q_1, \ldots, q_t\}$, we define the automaton $\mathcal{A}' = (Q', \Sigma, \delta')$ over the same alphabet as follows. As in the ternary construction, the state set is $Q' = \bigcup_{i=0}^{r-1} Q_i$ where each $Q_i$ contains $t$ states $q_{i,1}, \ldots, q_{i,t}$. The transitions from $Q_0$ to $Q_1$ imitate the transitions of $\mathcal{A}$: for every letter $a \in \Sigma$ we set $\delta'(q_{0,j}, a) = q_{1,k}$ if and only if $\delta(q_j, a) = q_k$. For the states in $Q_i$ with $i \neq 0$ we define the transitions by just shifting a state to the state with the same index in the next set: for every $a \in \Sigma$ we set $\delta'(q_{i,j}, a) = q_{i+1,j}$, with the index $i + 1$ taken modulo $p$.

Observe that the action of the set of words $\Sigma^r$ on the set $Q_i$ in $\mathcal{A}'$ induces the automaton $\mathcal{A}$ (up to duplicating its letters). Moreover, the words of length $r - 1$ only shift the states of the set $Q_1$ to $Q_0$. Thus, any word synchronizing $Q_1$ is of length at least $\mathrm{rt}(\mathcal{A}) \cdot r$ over the initial alphabet.

Clearly, this automaton has rank $r$, and its period is also $r$ because $\mathcal{A}$ is synchronizing and thus primitive. We obtain the following result.

**Proposition 10.** *For every $t$-state strongly connected synchronizing automaton $\mathcal{A}$ and for every $r$ there exists a $tr$-state strongly connected automaton $\mathcal{A}'$ over the same alphabet, with period and rank equal to $r$, such that* $\mathrm{mrt}(\mathcal{A}') = \mathrm{rt}(\mathcal{A}) \cdot r$.

Observe that the construction described preserves the property of the automaton to be strongly connected, circular or Eulerian. Applied to the Černý automaton this construction yields the following result.

**Corollary 11.** *For every $n$ and every $r$ such that $r$ divides $n$, there exists an $n$-state circular binary automaton of period and rank $r$ with minimum rank threshold $\frac{(n-r)^2}{r}$.*

The *Wielandt digraph* $W_n$ has $n > 1$ vertices $0, \ldots, n-1$. From each vertex $i > 0$ there are two edges to the next vertex $i+1$ modulo $n$, and from vertex 0 there are single edges to vertices 1 and 2. Introduced in [118], and studied in connection to synchronizing automata in [5], these digraphs have the interesting property that they admit only one coloring, when automata obtained by renaming letters are considered identical. The reset threshold of this $n$-state Wielandt automaton was proved in [5] to be $n^2 - 3n + 3$.

The Hybrid Černý-Road Coloring problem (see [5], [27]) asks for the shortest length of a synchronizing word among all colorings of a fixed primitive digraph with $n$ vertices. Since $W_n$ has only one coloring, it provides the lower bound $n^2 - 3n + 3$ on this quantity. We can apply the binary construction of this section on the Wielandt automaton. The resulting automaton of rank $r$ also admits only one coloring. Hence we get the following result in the spirit of the Hybrid Černý-Road Coloring problem, generalizing it to cases $r > 1$.

**Corollary 12.** *For every $n > 1$ and every $r$ such that $r$ divides $n$, there exists an $n$-vertex strongly connected digraph $D$ of constant outdegree 2 such that all colorings of $D$ are circular, have the same period and rank $r$, and for every coloring the length of a word of minimum rank is $\frac{(n-r)^2}{r} - n + 2r$.*

It is interesting to note that the digraphs $D$ in Corollary 12 are the digraphs with the largest possible index, described in Theorem 4.3 of [51], after duplicating some edges to make all outdegrees equal to 2. Recall that the *index* of a strongly connected digraph with period $r$ is the smallest $k$ such that any pair of vertices are connected by a directed path of length $k$ if and only if they are connected by a path of length $k + r$. In fact, one can easily show the following relationship, which also appears in [46] for the primitive case $r = 1$.

**Proposition 13.** *For a strongly connected $n$-state automaton $\mathcal{A}$ of rank $r$ and period $r$ the following holds:*

$$\mathrm{mrt}(\mathcal{A}) \geq k(\mathcal{A}) - n + r,$$

*where $k(\mathcal{A})$ is the index of the underlying digraph of $\mathcal{A}$.*

Since the index of $D$ in Corollary 12 was proved in [51] to be $\frac{(n-r)^2}{r} + r$, we get from Proposition 13 the same lower bound as in Corollary 12.

We finish this section with a family of strongly connected binary automata that reach the same minimum rank threshold as the ternary automata in Proposition 9. Recall the $n$-state Černý automaton from Section 3.1. Let $r$ be a number that divides $n$. Change in the Černý automaton the transition from state 1 by letter $a$ to go into state $r + 1$ instead of state 2. After this change, for any states $i$ and $j$ such that $i \equiv j$ modulo $r$, also $\delta(i, x) \equiv \delta(j, x)$ modulo $r$ holds for both

$x = a$ and $x = b$. This means that states in different residue classes modulo $r$ cannot be merged, so that the rank of this automaton is at least $r$. Using the trick from [5], we introduce a new input letter $c$ that acts as the word $ab$ does. Now letters $c$ and $b$ define exactly the modified Wielandt automaton leading to Corollary 12 above, so there is a word of rank $r$ with letters $c$ and $b$. Hence our automaton has rank $r$ as well.

Since the action of word $aa$ is the same as the action of $a$, a shortest minimum rank word $w$ cannot contain factor $aa$. The word $wb$ has also minimum rank, and it can be factored into $ab$'s and $b$'s. Viewing this as a word over letters $c$ and $b$, we see that the number of $c$'s and $b$'s must be at least the minimum rank threshold $\frac{(n-r)^2}{r} - n + 2r$ from Corollary 12. Since $b$ is a permutation and since $c$ merges at most one pair of states, there must be at least $n - r$ letters $c$ used. Each $c$ counts as two letters over the alphabet $\{a, b\}$, so the length of word $wb$ is at least

$$\frac{(n-r)^2}{r} - n + 2r + (n-r) = \frac{(n-r)^2}{r} + r.$$

Removing the last $b$ from $wb$ we obtain the following lower bound. Observe that the bound is exactly the same as in the ternary case in Proposition 9.

**Proposition 14.** *For every $n$ and every $r > 1$ such that $r$ divides $n$, there exists a binary $n$-state circular automaton $\mathcal{A}$ of rank $r$ having* $\mathrm{mrt}(\mathcal{A}) = \frac{(n-r)^2}{r} + r - 1$.

### 3.3.3 Upper Bound in the Case When the Rank Equals the Period

Obviously, the period of an automaton is a lower bound on its rank. It is interesting to consider the special case of automata where these two values are equal. For lower bounds, observe that the rank $r$ automata reported in Corollaries 11 and 12 have the same period as the rank. In this section we obtain upper bounds on the minimum rank threshold from any known upper bounds on the reset threshold, in the case that the rank equals the period.

For every $n$, let $f(n)$ denote the maximum of reset thresholds of $n$-state synchronizing automata.

**Theorem 15.** *Let $\mathcal{A}$ be an automaton of rank $r$ and period $r$. Then* $\mathrm{mrt}(\mathcal{A}) \leq r^2 \cdot f(\frac{n}{r}) + (r-1)$.

*Proof.* Let $\mathcal{A} = (Q, \Sigma, \delta)$. By Lemma 8 there exists a partition of the set $Q$ into the sets $Q_0, \ldots, Q_{r-1}$ such that every transition maps a state in $Q_i$ to a state in $Q_{i+1}$ (with the index $i + 1$ taken modulo $r$). Since the rank of $\mathcal{A}$ equals its period, each of the sets $Q_0, \ldots, Q_{r-1}$ is synchronizable (a set is called *synchronizable* if there is a word mapping this set to a single state). Assume without loss of generality that $Q_0$ is the smallest set in the partition. Consider then the automaton $\mathcal{A}^r = (Q_0, \Sigma^r, \delta')$ induced by the actions of all the words of length $r$ on $Q_0$. This automaton is synchronizing, and by our assumption there is a word synchronizing it of length at most $f(|Q_0|) \leq f(\frac{n}{r})$ over the alphabet $\Sigma^r$. Over the alphabet $\Sigma$ this word has length at most $r \cdot f(\frac{n}{r})$. Then to find a word of minimum rank it is enough to subsequently map each set $Q_1, \ldots, Q_{r-1}$ to $Q_0$ and apply the described word. In total we get a word of minimum rank of length at most $r^2 \cdot f(\frac{n}{r}) + (r-1)$. □

For example, using the unconditional upper bound $f(n) \leq \frac{n^3-n}{6}$ on the reset threshold [84] we get that for every $n$-state automaton of rank $r$ and period $r$ we have $\mathrm{mrt}(\mathcal{A}) \leq \frac{n(n^2-r^2)}{6r} + (r-1)$, which is roughly $r$ times stronger than the best known upper bound for the general case [63]. The Černý Conjecture implies the upper bound of $(n-r)^2 + (r-1)$. Thus, in the case of automata with rank equal to period the Rank Conjecture is implied by the Černý Conjecture up to an additive factor of $(r-1)$. However we conjecture that in this case the upper bound can be improved to $\frac{(n-r)^2}{r} + O(n)$.

## 3.4 Eulerian Automata

### 3.4.1 The Rank Conjecture

We continue our discussion on the Rank Conjecture proving it for a particular class of automata, namely the Eulerian automata. Eulerian automata have been widely studied, in particular, Kari [60] showed that $\mathrm{rt}(\mathcal{A}) \leq (n-1)(n-2)+1$ for any synchronizing Eulerian $n$-state automaton, thus proving the Černý Conjecture for this class of automata. We extend the mentioned result to the case of arbitrary minimum rank.

**Theorem 16.** *Let $\mathcal{A}$ be an $n$-state Eulerian automaton of rank $r$. Then $\mathcal{A}$ has a word of rank $r$ of length at most $(n-r-1)(n-r)+1$.*

*Proof.* Let $\mathcal{A} = (Q, \Sigma, \delta)$. Following [60], we consider the set $Q$, $|Q| = n$, of states as an orthonormal basis of $\mathbb{R}^n$ with subsets of states corresponding to the sums of the basis vectors. Thus, a set $S \subseteq Q$ is viewed as a vector $\sum_{q \in S} q$.

Every word $w \in \Sigma^*$ defines a state transition function $f_w : Q \to Q$ on the set of states, with $f_w(q) = \delta(q, w)$. Furthermore, $f_w^{-1}(q) = \{v \mid f_w(v) = q\}$. Since we know the values of $f_w^{-1}$ on all the basis vectors, there is a unique way to extend it to a linear mapping $f_w^{-1} : \mathbb{R}^n \to \mathbb{R}^n$. Clearly, for a set $S \subseteq Q$ we have $f_w^{-1}(S) = \sum_{q \in S} f_w^{-1}(q)$. Moreover, for a vector $x = (x_1, \ldots, x_n)$ we define a linear *weight* function $|x|$ such that $|(x_1, \ldots, x_n)| = x_1 + \ldots + x_n$. The weight of a set $S \subseteq Q$ is just its cardinality.

Let $Z_1 \subseteq \mathbb{R}^n$ be the set of all *non-extendable* vectors, i.e. such vectors $x$ that there exists no word $w$ with $|f_w^{-1}(x)| \neq |x|$ (all the remaining vectors we call *extendable*). Observe that $\sum_{w \in \Sigma^k} |f_w^{-1}(S)| = |\Sigma|^k \cdot |S|$. Thus, if there exists a word $w$ of length $k$ such that $|f_w^{-1}(S)| \neq |S|$ then there is a word of the same length extending $S$ (a word $v$ is said to *extend* $S$ if $|f_v^{-1}(S)| > |S|$). We will refer to that as the *averaging argument*.

Note that $Z_1$ is a linear subspace of $\mathbb{R}^n$ of dimension at least $r$. First we prove that it is a linear subspace. Indeed, consider a linear combination $\lambda_1 v_1 + \ldots + \lambda_k v_k$ of vectors from $Z_1$. Since the weight function is linear, any image of this combination under $f_w^{-1}$ has the same weight, and thus the combination belongs to $Z_1$. To bound the dimension of $Z_1$ from below, consider a word $w$ of minimum rank such that there exists a partition of $Q$ into sets $S_1, \ldots, S_r$, such that each $S_i$ is a maximal synchronizable set. Such a word exists by Proposition 1 of [60]. The vectors corresponding to $S_1, \ldots, S_r$ are then non-extendable,and linearly independent since they have disjoint non-zero coefficients in the standard basis decomposition. We apply some linear algebra to obtain the following lemma.

**Lemma 17.** *For every extendable vector $x$ there exists a word $w$ of length at most $n-r$ such that $|f_w^{-1}(x)| \neq |x|$.*

*Proof.* Suppose the contrary: let $x$ be extendable such that that shortest word $w$ such that $|f_w^{-1}(x)| \neq |x|$ has length $m > n-r$. Note that for any words $u, v$ we have $f_{uv}^{-1}(x) = f_u^{-1}(f_v^{-1}(x))$. Take $Z_0$ to be the orthogonal complement of $Z_1$. Since the dimension of $Z_1$ is at least $r$, the dimension of $Z_0$ is at most $n-r$. For every $i \leq m$ we denote $x_i = f_{w_i}^{-1}(x)$ where $w_i$ is the suffix of $w$ of length $i$, and we write $x_i = x_i^{(0)} + x_i^{(1)}$ for $x_i^{(0)} \in Z_0$ and $x_i^{(1)} \in Z_1$. Since $m$ is greater than the dimension of $Z_0$, vectors $x_0^{(0)}, x_1^{(0)}, \ldots, x_{m-1}^{(0)}$ are linearly dependent. This means that for some $k < m$ the vector $x_k^{(0)}$ is a linear combination $\lambda_0 x_0^{(0)} + \cdots + \lambda_{k-1} x_{k-1}^{(0)}$ of vectors before it, with coefficients $\lambda_i \in \mathbb{R}$. The corresponding linear combination of vectors $x_i$ is $\lambda_0 x_0 + \cdots + \lambda_{k-1} x_{k-1} = x_k + x'$ for some $x' \in Z_1$. Let $w = uv$ where $v$ is the suffix of $w$ of length $k$. Then, $f_u^{-1}(x_k) = f_u^{-1}(f_v^{-1}(x)) = f_w^{-1}(x)$.

Moreover, for every $i < k$ we have $|f_u^{-1}(x_i)| = |x_i|$. Indeed, $f_u^{-1}(x_i) = f_u^{-1}(f_{w_i}^{-1}(x)) = f_{uw_i}^{-1}(x)$ has the same weight as $x$ because $uw_i$ is shorter than $w$, and of course $|x_i| = |x|$. Also, because $x'$ is non-extendable, we have $|f_u^{-1}(x')| = |x'|$. Putting all together, using linearity of $f_u^{-1}$ and the weight function, we obtain $|f_w^{-1}(x)| = |x|$, a contradiction. □

By the averaging argument we obtain from Lemma 17 that for any extendable set $S$ of states there is a word $w$ of length at most $n-r$ such that $|f_w^{-1}(S)| \geq |S|+1$. Now we apply this extension procedure as follows. Start with a one-state set. Extend it step by step to a maximal synchronizable set (having size $\frac{n}{r}$). Then add another state to this maximal synchronizable set and extend this new set to a union of two disjoint maximal synchronizable sets. Repeat this procedure of adding a new state and extending the set to a union of several maximal synchronizable sets until the whole set of states of the automaton is reached. The extension is possible, since at every step the set $S$ that we have to extend is a disjoint union of several maximal synchronizable subsets and a non-maximal synchronizable subset $S'$. Any word extending $S'$ extends $S$, since $f_w^{-1}$ preserves the weights of all the maximal synchronizable subsets for any word $w$ (since otherwise by the averaging argument such sets are extendable).

For each step of this algorithm, we have a word of length at most $n-r$ to extend a set by one element. Each maximal synchronizable set has size $\frac{n}{r}$, and we have to reach $r$ such sets, so the total length of the word is at most $(n-r)(\frac{n}{r}-1)r = (n-r)^2$. We can initially choose a one-state set extendable by a word of length 1, which improves the bound to $(n-r)(n-r-1)+1$. □

To obtain a lower bound on the minimum rank threshold of Eulerian automata, recall the construction used to prove the bound of Proposition 9. It was mentioned previously that applying it to an Eulerian automaton yields another Eulerian automaton. Thus, we repeat the same reasoning starting with a synchronizing $n$-state Eulerian automaton over alphabet of size 4 having reset threshold $\frac{n^2-3}{2}$, for any $n > 1$ such that $n \equiv 1 \pmod 4$, see [107].

**Proposition 18.** *For every $n$ and every $r < n$ such that $n = (4p+1)r$, there exists an $n$-state Eulerian automaton $\mathcal{A}$ of rank $r$ with* $\mathrm{mrt}(\mathcal{A}) = \frac{n^2-r^2}{2r} - 1$.

The standard binarization methods cannot be applied to provide the lower bounds for binary Eulerian automata. However, we can apply the argument of Proposition 10 to the $n$-state binary Eulerian automaton whose reset threshold is at least $\frac{n^2-3n+4}{2}$ for odd $n \geq 3$ [46]. (This was proved for all odd $n \geq 5$ in [46] but the same construction also covers the case $n = 3$.) The automaton we obtain is also Eulerian.

**Proposition 19.** *For every $n$ and every $r$ such that $r$ divides $n$ and $n/r \geq 3$ is odd, there exists an $n$-state binary Eulerian automaton $\mathcal{A}$ of rank $r$ having* $\mathrm{mrt}(\mathcal{A}) \geq \frac{(n-2r)^2+nr}{2r}$.

The multiplicative gap between the lower and the upper bounds consists intuitively of two parts. The factor of two comes from the gap between the known bounds on the reset threshold of Eulerian automata, while the factor $r$ comes from the gap on the minimum rank threshold in general strongly connected automata that we see in the results in Section 3.3.

### 3.4.2 A Corollary for Circular Automata

In this section we provide a simple trick, similar to the idea of [26], which allows to transfer the results on Eulerian automata to the class of circular automata. Recall that an automaton is called circular if there is a letter which acts on its set of states as a cyclic permutation. The Černý

Conjecture for this automata class was proved by Dubuc [36]. Note that the Černý automata are circular and possess the largest known reset thresholds.

Let us consider an $n$-state circular automaton $\mathcal{A} = (Q, \Sigma, \delta)$ such that some letter $b \in \Sigma$ acts as a cyclic permutation on $Q$. Let us replace each $a \in \Sigma$ by $n$ letters $a_0, \ldots, a_{n-1}$, where $a_i$ acts on $Q$ the same way as the word $ab^i$ does in the original automaton. Let $\Sigma'$ be the obtained new alphabet of size $n \cdot |\Sigma|$. It is not hard to prove that the obtained automaton is Eulerian.

Observe that this transformation preserves the synchronization properties of the initial automaton in the following sense. A word of rank $r$ over $\Sigma$ is clearly a word of rank $r$ over $\Sigma'$ because $\Sigma \subset \Sigma'$. The opposite holds as well since every word over $\Sigma'$ can be rewritten as a word over $\Sigma$. It follows that the rank of the resulting automaton is equal to the rank of the initial one.

**Theorem 20.** *Every $n$-state circular automaton of rank $r < n$ has a minimum rank word of length at most $(2n - r - 1)(n - r - 1) + 1$.*

*Proof.* Let $\mathcal{A} = (Q, \Sigma, \delta)$ be an $n$-state circular automaton with a cyclic permutation letter $b$. An Eulerian automaton $\mathcal{A}' = (Q, \Sigma', \delta')$ with $n$ states is constructed as above. Now we show how to use the procedure described in Theorem 16 to get the upper bound.

According to the proof of Theorem 16, for every $s \in \mathbb{R}^n$ there exists a unique representation $s = s_0 + s_1$ where $s_0 \in Z_0$ and $s_1 \in Z_1$. Furthermore, $|f_w^{-1}(s)| \neq |s|$ is equivalent to $|f_w^{-1}(s_0)| \neq |s_0|$.

Observe that any word in $\Sigma'$ can be written as a concatenation of words over $\Sigma$. Thus we can apply Lemma 3 of [60] on the linear transformations $f_a^{-1}$, for $a \in \Sigma$, in the automaton $\mathcal{A}'$, and get that the shortest word $w \in \Sigma^*$ such that $|f_w^{-1}(s_0)| \neq |s_0|$ has length at most $n - r$. Consequently, $w$ is the shortest word over $\Sigma$ such that $|f_w^{-1}(s)| \neq |s|$.

Let $w = cv$ where $c \in \Sigma$ and $v \in \Sigma^*$. Now consider all the words of the form $\sigma v$ with $\sigma \in \Sigma'$. Clearly, $w$ is one of them. Because $\mathcal{A}'$ is Eulerian, we have

$$\sum_{\sigma \in \Sigma'} |f_{\sigma v}^{-1}(x)| = \sum_{\sigma \in \Sigma'} |f_\sigma^{-1}(f_v^{-1}(x))| = |\Sigma'| \cdot |f_v^{-1}(x)| = |\Sigma'| \cdot |x|.$$

Since there exists a word $w = cv$ such that $|f_w^{-1}(x)| \neq |x|$, the above equality implies that there is $u = \sigma v$ such that $|f_u^{-1}(x)| > |x|$. Notice that $v$ is a word of length at most $n - r - 1$ over $\Sigma$, and hence $u$ is of length $|\sigma| + |v| \leq n + (n - r - 1) = 2n - r - 1$ over $\Sigma$.

Thus we showed that every extendable set of states in $\mathcal{A}'$ can be extended by a word of length at most $2n - r - 1$ (over the alphabet $\Sigma$). We can now use the extension procedure described in Theorem 16 (starting from a one-state set extendable by a word of length 1) and get the upper bound of $(2n - r - 1)(n - r - 1) + 1$ on the length of a shortest word of minimum rank in $\mathcal{A}$. □

### 3.4.3 A Road Coloring Algorithm

As proved by Kari [60], every primitive strongly connected Eulerian digraph such that all its vertices have equal outdegrees has a synchronizing coloring. If the primitiveness condition is omitted, the period of a digraph is the lower bound on the rank of any coloring. A coloring of rank equal to period always exists and can be found in quadratic time [8]. We show that for Eulerian digraphs it can be found in almost linear time. We use the approach described in Section 3 of [60] and show how to generalize it and turn into an algorithm.

First observe that a permutation coloring (a coloring of rank $n$) of an Eulerian digraph with $n$ vertices and constant outdegree $k$ corresponds to a partition of a regular bipartite graph with $n$ vertices and $kn$ edges into $k$ perfect matchings (Lemma 1 of [60]), and thus can be computed in $O(kn \log k)$ time [30].

The construction of a permutation coloring is used as a subroutine in order to construct a coloring with a stable pair of states. A pair of states $p, q$ of an automaton is called *stable* if application of any word to this pair results in a synchronizable pair. For a permutation coloring $\mathcal{A} = (Q, \Sigma, \delta)$ of a digraph take a state $x \in Q$ such that $y = \delta(x, a) \neq \delta(x, b) = z$ for some letters $a, b \in \Sigma$. Note that in a strongly connected digraph such state always exists, otherwise the digraph consists of one cycle and we have nothing to prove. We swap the letters coloring the edges $x \to y$ and $x \to z$. As proved in Theorem 1 of [60], the pair $y, z$ is then stable in the resulting automaton $\mathcal{A}'$ and thus defines a congruence relation (that is, an equivalence relation invariant under application of any word) $\equiv$ on its state set. The quotient automaton $\mathcal{A}'/\equiv$ is then obtained by merging all the states of each congruence class. If $\mathcal{A}'$ is Eulerian, so is $\mathcal{A}'/\equiv$ [60].

**Lemma 21.** *Let $\mathcal{A}'$ be the Eulerian automaton, and $y, z$ be the stable pair with corresponding congruence relation $\equiv$ obtained as described above. Then the quotient automaton $\mathcal{A}'/\equiv$ has at most half as many states as $\mathcal{A}'$.*

*Proof.* We compute $\mathcal{A}'/\equiv$ following the Merge procedure described in [8]. We start by merging the congruent pair $y, z$ and then propagate this equivalence to the images of $y, z$ under all the letters in $\Sigma$ until we get a deterministic automaton. Observe that since we start with a permutation coloring, each state that has not yet been merged with some other state has all incoming edges of different colors. Thus, if there is such a state in the pair to be merged, the second state in this pair is different from it, and thus further calls of merging their successor will be performed. Moreover, assume that some state is not merged with any state during this procedure. Then there is such a state $p$ having a transition going to it from some already merged state $q$, otherwise the digraph is not strongly connected. This means that during the first merging for $q$, merging for $p$ has to be called, which is a contradiction. Hence, each state is in a congruence class of cardinality at least 2, and after taking the quotient, the number of states of $\mathcal{A}'/\equiv$ is at most half of the number of states $\mathcal{A}'$. □

**Theorem 22.** *Given a strongly connected Eulerian digraph of period $r$ with $n$ vertices and outdegree $k$, a coloring of rank $r$ of this digraph can be found in $O((k \log k + \alpha(n)) \cdot n)$ time, where $\alpha(\cdot)$ is the inverse Ackermann function.*

*Proof.* The algorithm is recursive. At each iteration we start by finding a coloring with a stable pair as described above. Then we proceed by computing the quotient automaton as in Lemma 21. The automaton we obtain is Eulerian [60], moreover, it has the same period since no pair of states from different sets in a $p$-partition can be stable (since no such pair can be synchronized). If the automaton has rank $r$, we stop, otherwise we call the same algorithm for coloring it and then recover the final coloring by taking for every vertex the same permutation of the colors of outgoing edges as used for the equivalence class of this vertex (see Theorem 1 of [60]).

To analyze the time complexity, we estimate the complexity of one recursion step. Let $\ell$ be the size of the automaton at some iteration. As it was mentioned before, it takes $O(k\ell \log k)$ time to find a permutation coloring. The Merge procedure requires $O(k\ell)$ time for traversing and $O(\ell\alpha(\ell))$ time for merging the sets. Moreover, recovering the coloring from the smaller automaton can be done in $O(k\ell)$ time by storing the quotient automaton (together with the correspondence between the states and their equivalence classes) at each iteration. Hence, the time complexity of one iteration is $O(\ell(k \log k + \alpha(\ell)))$.

Now we can sum up the time complexity of all recursion steps. Lemma 21 implies that the number of states of each next automaton in the recursion call is decreased at least twice. Thus, the total time complexity is $O(n(k \log k + \alpha(n)))$, where $n$ is the number of vertices of the initial digraph. □

# Chapter 4

# Extremal Bounds for Partial DFAs and Prefix Codes

All automata in this chapter are deterministic, so we say "partial automaton" for "partial DFA", and "complete automaton" for "complete DFA".

## 4.1 Introduction

In this chapter, we transfer the Černý conjecture and a more general Rank Conjecture to the case of partial DFAs. This setting is motivated by the connection between strongly connected partial DFAs and recognizable prefix codes described in Section 2.4. The fact of the partial DFA being strongly connected is essential here, since checking if a partial DFA is synchronizing is PSPACE-complete in general [11].

Partial DFAs can be seen as a natural intermediate class between complete DFAs and unambiguous NFAs. The known upper bounds for unambiguous NFAs are quite high (see the next chapter), which motivates studying simpler models, and partial DFAs indeed allow to obtain much tighter bounds. Despite their similarity to complete DFAs, the case of partial DFAs requires a set of new techniques to tackle synchronization problems. In contrast to the complete case, in incomplete DFAs some states can have an undefined action of a synchronizing word and all the others are collected to one state. In other words, the process of synchronization becomes more complicated and includes two different ways of reducing the number of active states. Thus, it is not possible to transfer trivially the classical synchronization methods, both extending and compressing.

We develop a framework for deriving results for strongly connected partial DFAs and apply it to relate problems for strongly connected partial and complete DFAs, showing that they are equivalent, sometimes up to minor differences.

The chapter is structured incrementally: we start from basic properties and introduce more advanced techniques alternatingly with their applications. Our main results are as follows:

1. The rank conjecture is equivalent in both models. It shows a general way for transferring upper bounds from the case of complete DFAs to partial DFAs, e.g., that the rank conjecture holds for partial Eulerian automata. For this, we introduce the first basic tool called *fixing automaton*, which is a complete automaton obtained from a partial one and sharing some properties.

2. Synchronization of a strongly connected partial automaton can be effectively reduced to the same problem of a complete automaton. We get this through another construction called

*collecting automaton*, which extends the concept of the fixing automaton.

3. The Černý conjecture and all upper bounds on reset threshold, up to a subquadratic additive component, are equivalent in both models. This also follows through a particular usage of the collecting automaton.

4. The length of the shortest reset words of literal automata of prefix codes is at most $\mathcal{O}(n \log^3 n)$. This bound is the same as the best known for complete automata, but it requires more arguments, which constitute the most involved proof in this chapter. To show this, we prove that such automata have a linear word of logarithmic rank, and use another construction called *induced automaton*, which allows applying linear algebraic techniques for partial automata.

5. Undefined transitions do not help: an essential improvement of the upper bound on the length of the shortest reset words of a properly incomplete DFA implies an improvement on the equivalent bound for a complete DFA.

## 4.2 Preliminaries

By $\Sigma^i$ we denote the set of all words over $\Sigma$ of length exactly $i$ and by $\Sigma^{\leq i}$ the set of all words over $\Sigma$ of length at most $i$. For two sets of words $W_1, W_2 \in \Sigma^*$, by $W_1W_2$ we denote their product $\{w_1w_2 \in \Sigma^* \mid w_1 \in W_1, w_2 \in W_2\}$. The empty word is denoted by $\varepsilon$. Throughout the chapter, by $n$ we always denote the number of states $|Q|$.

We provide some definitions and notation special for partial DFAs. We use $\perp$ to say that a transition is undefined, that is, $\delta(q, w) = \perp$ means that the action of the word $w$ in undefined for $q$, and $\delta(q, w) \neq \perp$ means that is is defined. Given $S \subseteq Q$, the *image* of $S$ under the action of $w$ is $\delta(S, w) = \{\delta(q, w) \mid q \in S, \delta(q, w) \neq \perp\}$. The *preimage* of $S$ under the action of $w$ is $\delta^{-1}(S, w) = \{q \in Q \mid \delta(q, w) \in S\}$. Since $\mathcal{A}$ is deterministic, for disjoint subsets $S, T \subseteq Q$, their preimages under the action of every word $w \in \Sigma^*$ are also disjoint.

We say that a word $w$ *compresses* a subset $S \subseteq Q$, if $|\delta(S, w)| < |S|$ but $\delta(S, w) \neq \emptyset$. A subset that admits a compressing word is called *compressible*. There are two ways to compress a subset $S \subseteq Q$ with $|S| \geq 2$. One possibility is the *pair compression*, which is the same as in the case of a complete automaton, i.e., mapping at least two states $p, q \in S$ to the same state (but not to $\perp$). The other possibility is to map at least one state from $S$ to $\perp$, but not all states from $S$ to $\perp$. Sometimes, a subset can be compressed in both ways simultaneously.

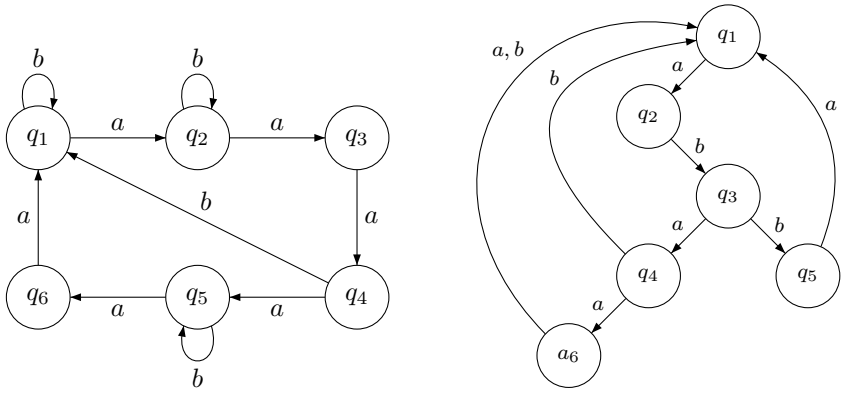


Figure 4.1: Left: a strongly connected partial 6-state binary automaton; right: the literal automaton of the prefix code $\{abaaa, abaab, abab, abba\}$.

An example of a strongly connected partial automaton is shown in Fig. 4.1(left). We have two undefined transitions: $\delta(q_3, b) = \delta(q_6, b) = \bot$. The unique shortest reset word for this automaton is $bab$: $\delta(Q, b) = \{q_1, q_2, q_5\}$, $\delta(Q, ba) = \{q_2, q_3, q_6\}$, and $\delta(Q, bab) = \{q_2\}$. However, in contrast with the case of a complete automaton, the preimage $\delta^{-1}(\{q_2\}, bab) = \{q_1, q_4\}$ is not $Q$.

## 4.3 Upper Bounds

### 4.3.1 Inseparability Equivalence

Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a partial automaton. We define the inseparability relation $\equiv$ on $Q$. Two states are *separable* if they can be separated by mapping exactly one of them to $\bot$, leaving the other one.

**Definition 23.** *The* inseparability equivalence $\equiv$ *on* $Q$ *is defined as follows:*

$$p \equiv q \quad \textit{if and only if} \quad \forall_{u \in \Sigma^*} \ (\delta(p, u) \in Q \Leftrightarrow \delta(q, u) \in Q)\,.$$

The same relation is considered in [14, Section 1.4] if all states of the partial automaton are final. Also, if we replace $\bot$ with a unique final state, then $\equiv$ is the well-known Myhill-Nerode congruence on words in a complete automaton. Under a different terminology, it also appears in the context of $\varepsilon$-machines, where non-equivalent states are called *topologically distinct* [109].

For a subset $S \subseteq Q$, let $\kappa(S)$ be the number of equivalence classes that have a non-empty intersection with $S$. In the automaton from Fig. 4.1(left) we have three equivalence classes, where $q_1 \equiv q_4$, $q_2 \equiv q_5$, and $q_3 \equiv q_6$.

Our first auxiliary lemma states that a subset $S \subseteq Q$ that intersects at least two equivalence classes can be compressed and the number of intersected classes can be decreased with a short word. This is done by mapping to $\bot$ all the states of $S$ from at least one equivalence class, but not the whole set $S$. A linear upper bound can be inferred from a standard analysis of the corresponding Myhill-Nerode congruence (e.g., [110]), but we will need a more precise bound in terms of $\kappa(S)$.

**Lemma 24.** *Let* $\mathcal{A} = (Q, \Sigma, \delta)$ *be a partial automaton, and let* $S \subseteq Q$ *be a subset such that* $\kappa(S) \geq 2$. *Then there is a word* $w \in \Sigma^*$ *of length at most* $\kappa(Q) - \kappa(S) + 1 \leq n - |S| + 1$ *and such that* $1 \leq \kappa(\delta(S, w)) < \kappa(S)$.

*Proof.* We define auxiliary relations on $Q$ that are restricted to words of certain lengths. For $k \geq 0$, we define:

$$p \equiv_k q \quad \text{if and only if} \quad \forall_{u \in \Sigma^{\leq k}} \ (\delta(p, u) \in Q \Leftrightarrow \delta(q, u) \in Q)\,.$$

Clearly, $\equiv_0$ has all states in one equivalence class, and there is some $m$ such that $\equiv_m$ is the same as $\equiv$ because the number of different actions of words is finite. Also, for every $p, q \in Q$ and $k \geq 0$, if $q \not\equiv_k p$, then $q \not\equiv_{k+1} p$.

We show that if for some $k$, $\equiv_k$ is the same as $\equiv_{k+1}$, then the chain of relations stabilizes at $\equiv$, i.e., all relations $\equiv_k, \equiv_{k+1}, \equiv_{k+2}, \ldots$ are the same as $\equiv$. Assume for a contradiction that $\equiv_k$ is the same as $\equiv_{k+1}$, but $\equiv_{k+2}$ is different from them. This means that there are two distinct states $p, q \in Q$ such that $p \equiv_k q$, $p \equiv_{k+1} q$, and $p \not\equiv_{k+2} q$. Hence, there exists a word $u$ of length $k+2$ such that, without loss of generality, $\delta(p, u) = r \in Q$ and $\delta(q, u) = \bot$. Write $u = av$, where $a \in \Sigma$ and $v \in \Sigma^*$. Then $\delta(\{p, q\}, a) = \{p', q'\}$ for some distinct states $p', q' \in Q$. Since $|v| = k + 1$, $\delta(p', v) = r$ and $\delta(q', v) = \bot$, we have $p' \not\equiv_{k+1} q'$, and from our assumption that $\equiv_k$ is the same as $\equiv_{k+1}$, we also have $p' \not\equiv_k q'$. This means that there exists a word $v'$ of length at most $k$ such that $\delta(p', v') = r' \in Q$ and $\delta(q', v') = \bot$, or vice versa. But then $\delta(p, av') = r'$, $\delta(q, av') = \bot$, and $|av'| \leq k + 1$, which yields a contradiction with that $p \equiv_{k+1} q$.

If for some $k$, $\equiv_{k+1}$ is different from $\equiv_k$, then the number of equivalence classes in $\equiv_{k+1}$ is larger by at least one than the number of equivalence classes in $\equiv_k$. Note that the number of equivalence classes is limited by $\kappa(Q)$, so $\equiv_{\kappa(Q)-1}$ (and every further relation) is the same as $\equiv$.

Observe that, for a $k \geq 0$, if $S$ is not contained in a single equivalence class of $\equiv_k$, then there exists a word $w \in \Sigma^{\leq k}$ such that, for some states $p, q \in S$, we have $\delta(p, w) \neq \bot$ and $\delta(q, w) = \bot$, thus $w$ satisfies $1 \leq \kappa(\delta(S, w)) < \kappa(S)$.

We consider $\equiv_{\kappa(Q)-\kappa(S)+1}$. It has at least $\kappa(Q) - \kappa(S) + 2$ equivalence classes. Therefore, since there are at most $\kappa(Q) - \kappa(S)$ equivalence classes not intersecting $S$, this relation must have at least two classes that intersect $S$, so $S$ is not contained in a single class. It follows that there is a word $w$ of length at most $\kappa(Q) - \kappa(S) + 1$ satisfying the lemma.

The second upper bound follows from the fact that there are at most $n - |S|$ equivalence classes in $Q \setminus S$, so $\kappa(Q) \leq n - |S| + \kappa(S)$. □

By an iterative application of Lemma 24, we can easily compress any subset of states to a subset of a single equivalence class.

**Corollary 25.** *Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a partial automaton, and let $S \subseteq Q$ be a non-empty subset. There is a word $w$ of length at most $(\kappa(S) - 1)(\kappa(Q) - \kappa(S)/2)$ such that $\delta(S, w)$ is non-empty and is contained in one inseparability class.*

*Proof.* In the worst case, we apply at most $\kappa(S) - 1$ times Lemma 24 for subsets intersecting $\kappa(S), \kappa(S) - 1, \ldots, 2$ equivalence classes. □

### 4.3.2 Fixing Automaton

The other possibility of compressing a subset in a partial automaton is the classical pair compression. This is the only way for compressing a subset with all states in one equivalence class, which is always the case in a complete automaton.

Our next tool to deal with this way of compression is the *fixing automaton*. This is a complete automaton obtained from partial one, defined as follows.

**Definition 26** (Fixing automaton)**.** *For a partial automaton $\mathcal{A}(Q, \Sigma, \delta)$, the* fixing automaton *is the complete automaton $\mathcal{A}^{\mathrm{F}} = (Q, \Sigma, \delta^{\mathrm{F}})$ such that the states are fixed instead of having an undefined transition: for every $q \in Q$ and $a \in \Sigma$, we have $\delta^{\mathrm{F}}(q, a) = q$ if $\delta(q, a) = \bot$, and $\delta^{\mathrm{F}}(q, a) = \delta(q, a)$ otherwise.*

We list some useful properties of the fixing automaton.

**Lemma 27.** *Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a partial automaton, let $S \subseteq Q$, and let $w \in \Sigma^*$. We have $\delta(S, w) \subseteq \delta^{\mathrm{F}}(S, w)$. Moreover, if for every state $q \in S$ we have $\delta(q, w) \neq \bot$, then $\delta(S, w) = \delta^{\mathrm{F}}(S, w)$.*

**Lemma 28.** *Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a partial automaton. Let $S \subseteq Q$ be a non-empty subset and $w \in \Sigma^*$ be a word. There exists a word $w' \in \Sigma^*$ of length $|w'| \leq |w|$ such that $\emptyset \neq \delta(S, w') \subseteq \delta^{\mathrm{F}}(S, w)$. In particular, if $w$ has rank $r$ in $\mathcal{A}^{\mathrm{F}}$, then $w'$ has rank $1 \leq r' \leq r$ in $\mathcal{A}$.*

*Proof.* For a letter $a \in \Sigma$, let $\delta^{-1}(\bot, a) = \{q \in Q \mid \delta(q, a) = \bot\}$, which is the set of states that are mapped to $\bot$ under the action of $a$ in $\mathcal{A}$.

We prove the statement by induction on the length $|w|$. Obviously, it holds for $|w| = 0$. Consider $w = ua$ for some $u \in \Sigma^*$ and $a \in \Sigma$, and let $u'$ be the word obtained from the inductive assumption for $u$. Recall that all states from $\delta^{-1}(\bot, a)$ are fixed in $\mathcal{A}^{\mathrm{F}}$ under the action of $a$. Let $T = \delta(S, u')$; thus $\emptyset \neq T \subseteq \delta^{\mathrm{F}}(S, u)$ by the inductive assumption. We have two cases.

1. If $T \subseteq \delta^{-1}(\perp, a)$, then we let $w' = u'$. Hence, we still have $\delta(S, w') = \delta(S, u') = T \subseteq \delta^{\mathrm{F}}(S, ua)$, because $T$ is fixed under the action of $a$ by $\delta^{\mathrm{F}}$.

2. If $T \not\subseteq \delta^{-1}(\perp, a)$, then we let $w' = u'a$. Since there is a state $q \in T \setminus \delta^{-1}(\perp, a)$, we know that $\delta(T, a)$ is non-empty. We also have $\delta(T, a) \subseteq \delta^{\mathrm{F}}(T, a)$, since the states in $T \setminus \delta^{-1}(\perp, a)$ are mapped in the same way under the action of $a$ by both $\delta$ and $\delta^{\mathrm{F}}$. We get that $\delta(S, u'a) = \delta(T, a) \subseteq \delta^{\mathrm{F}}(T, a) \subseteq \delta^{\mathrm{F}}(S, ua)$.

□

**Corollary 29.** *The minimal non-zero rank of a partial automaton $\mathcal{A}$ is at most the minimal rank of $\mathcal{A}^{\mathrm{F}}$.*

In the general case of a partial automaton, it can happen that we cannot compress some subset $S$ even if there exists a word of non-zero rank smaller than $|S|$. This is the reason why the shortest words of the minimal non-zero rank can be exponentially long and why deciding if there is a word of a given rank is PSPACE-complete [11].

However, in the case of a strongly connected partial automaton, as well as for a complete automaton, every non-mortal word can be extended to a word of the minimal non-zero rank. This is a fundamental difference that allows constructing compressing words iteratively. Note that the fixing automaton of a strongly connected partial one is also strongly connected.

**Lemma 30.** *Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a strongly connected partial automaton, and let $r$ be the minimal non-zero rank over all words. For every non-empty subset $S \subseteq Q$, there exists a non-mortal word $w$ such that $|\delta(S, w)| \leq r$.*

*Proof.* Let $u$ be a word of rank $r$. Then there exists a state $q \in Q$ such that $\delta(q, u)$ is defined ($\neq \perp$). Let $p \in S$ be any state and let $v_{p,q}$ be a word mapping $p$ to $q$ (from strong connectivity). Then $\delta(S, v_{p,q}u)$ has a non-zero rank $\leq r$. □

### 4.3.3 Rank Conjecture

The *rank conjecture* (sometimes called *Černý-Pin conjecture*) is a well-known generalization of the Černý conjecture to non-synchronizing automata (e.g., [84]). See Chapter 3 for the discussion of the rank conjecture for the case of complete DFAs.

**Conjecture 31** (The rank conjecture)**.** *For an $n$-state complete automaton where $r$ is the minimal rank over all words, there exists a word of rank $r$ and of length at most $(n - r)^2$.*

For partial automata, the rank conjecture is analogous with the exception that $r$ is the minimal non-zero rank.

**Theorem 32.** *Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a strongly connected partial automaton. If the rank conjecture holds for the fixing automaton $\mathcal{A}^{\mathrm{F}}$, then it also holds for $\mathcal{A}$.*

*Proof.* Let $r$ be the minimal rank in $\mathcal{A}^{\mathrm{F}}$ over all words. From the conjecture and by Lemma 28, there exists a word $w'$ of length at most $(n - r)^2$ and such that $\emptyset \neq \delta(Q, w') \subseteq \delta^{\mathrm{F}}(Q, w)$.

Let $r' \leq r$ be the minimal rank of $\mathcal{A}$. For every $s = r, r-1, \ldots, r'+1$, we inductively construct a word of non-zero rank at most $s-1$, of length at most $(n-(s-1))^2$, and such that $w'$ is its prefix. Let $w'v$ be a word of rank at most $s$ and of length at most $(n-s)^2$, and let $S = \delta(Q, w'v)$. Suppose that $\kappa(S) = 1$. Since $s$ is not the minimal rank of $\mathcal{A}$, by Lemma 30, $S$ must be compressible. Since its states are inseparable, there must be two distinct states $p, q \in S$ and a word $u$ such

that $\delta(q,u) = \delta(p,u) \neq \bot$. But (from Lemma 27) $\{p,q\} \subseteq \delta(Q,w'v) \subseteq \delta^{\mathrm{F}}(Q,w'v) \subseteq \delta^{\mathrm{F}}(Q,wv)$, thus $\delta^{\mathrm{F}}(Q,wv)$ is compressible in $\mathcal{A}^{\mathrm{F}}$, which contradicts that $w$ has the minimal rank in $\mathcal{A}^{\mathrm{F}}$. Hence $\kappa(S) \geq 2$, and by Lemma 24, $\delta(Q,w'v)$ can be compressed with a word $u$ of length at most $n-s+1$. We have $|w'vu| \leq (n-s)^2 + n - s + 1 \leq (n-(s-1))^2$, which proves the induction step. □

The theorem implies that, in the strongly connected case, the rank conjecture is true for complete automata if and only if it is true for partial automata. For instance, we immediately get the result for the class of Eulerian automata. A partial automaton is *Eulerian* if the numbers of outgoing and incoming transitions are the same at every state, i.e., for every $q \in Q$, we have $|\{a \in \Sigma \mid \delta(q,a) \in Q\}| = |\{(p,a) \in Q \times \Sigma \mid \delta(p,a) = q\}|$. The following corollary follows from the facts that the rank conjecture holds for complete Eulerian automata [61] and that the fixing automaton of a partial Eulerian automaton is also Eulerian.

**Corollary 33.** *The rank conjecture is true for partial Eulerian automata.*

### 4.3.4 Collecting Automaton

The fixing automaton allows relating the behavior of words in a partial and a complete automaton, but its main disadvantage is that it is not necessarily synchronizing. Therefore, we will need one more tool, called *collecting automaton*. It is an extension of the fixing automaton by an additional letter that allows a quick synchronization into one inseparability class, while it does not affect the length of a shortest synchronizing word for any particular inseparability class.

By $\mathcal{A}/_{\equiv} = (Q_{\mathcal{A}/_{\equiv}}, \Sigma, \delta_{\mathcal{A}/_{\equiv}})$, we denote the quotient automaton by the inseparability relation. $\mathcal{A}/_{\equiv}$ is also a partial automaton, and if $\mathcal{A}$ is strongly connected, then so is $\mathcal{A}/_{\equiv}$. By $[p] \in Q_{\mathcal{A}/_{\equiv}}$, we denote the class of a state $p \in Q$ of the original automaton $\mathcal{A}$.

Let $T$ be a directed tree on $Q_{\mathcal{A}/_{\equiv}}$ such that each edge from a class $[p] \in Q_{\mathcal{A}/_{\equiv}}$ to a class $[p'] \in Q_{\mathcal{A}/_{\equiv}}$ corresponds to a transition of a letter $a \in \Sigma$ and is labeled by this letter, and there is an arbitrary root $[r] \in Q_{\mathcal{A}/_{\equiv}}$ such that all edges are directed toward it. We call such a directed labeled tree a *collecting tree* of $\mathcal{A}$. (See Fig. 4.2 for an example.) An automaton can have many collecting trees, even for the same $[r]$, and every strongly connected automaton has a collecting tree for every class $[r]$.

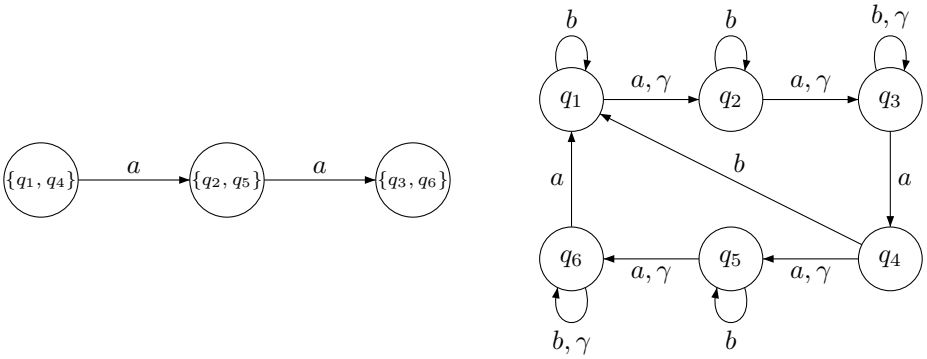


Figure 4.2: Left: a collecting tree with root $[q3] = \{q_3, q_6\}$; right: the corresponding collecting automaton (right) of the example from Fig. 4.1(left).

**Definition 34** (Collecting automaton)**.** *Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a strongly connected partial automaton and let $T$ be one of its collecting trees with a root $[r]$. The* collecting automaton $\mathcal{A}^{\mathrm{C}(T)} = (Q, \Sigma \cup \{\gamma\}, \delta^{\mathrm{C}(T)})$ *is defined as follows:*

- $\gamma \notin \Sigma$ *is a fresh letter.*
- *The transition function* $\delta^{\mathrm{C}(T)}$ *on* $\Sigma$ *is defined as in the fixing automaton* $\mathcal{A}^{\mathrm{F}}$*.*
- *The action of the new letter* $\gamma$ *is defined according to the edges in* $T$*: Let* $q_1 \in Q \setminus [r]$ *be a state. Since* $T$ *is a tree directed to* $[r]$*, there is exactly one edge* $([q_1], [q_2], a) \in T$*, for some* $[q_2] \in Q_{\mathcal{A}/_{\equiv}}$ *and* $a \in \Sigma$*. We set* $\delta^{\mathrm{C}(T)}(q_1, \gamma) = \delta(q_1, a)$*. Finally, the transition of* $\gamma$ *on each state in* $[r]$ *is the identity.*

A collecting automaton is always strongly connected, as it contains all transitions of the fixing automaton. We prove several properties connecting partial automata and their collecting complete automata. They are preliminary steps toward relating the Černý conjecture for strongly connected partial and complete automata.

**Lemma 35.** *Let* $\mathcal{A} = (Q, \Sigma, \delta)$ *be a strongly connected partial automaton, and let* $T$ *be one of its collecting trees with a root* $[r]$*. If there is a word over* $\Sigma \cup \{\gamma\}$ *synchronizing* $[r]$ *in* $\mathcal{A}^{\mathrm{C}(T)} = (Q, \Sigma \cup \{\gamma\}, \delta^{\mathrm{C}(T)})$*, then there is such a word over* $\Sigma$ *that is not longer.*

*Proof.* Let $w' \in (\Sigma \cup \{\gamma\})^*$ be a word synchronizing $[r]$ in $\mathcal{A}^{\mathrm{C}(T)}$. We construct $w \in \Sigma^*$ from $w'$ such that $w$ also synchronizes $[r]$ and $|w| \leq |w'|$. This is done by replacing every occurrence of $\gamma$ in $w'$ with a suitable substituting letter from $\Sigma$ or removing it.

Consider an occurrence of $\gamma$ in $w'$, so let $w' = u\gamma v$, for some $u, v \in (\Sigma \cup \{\gamma\})^*$. Let $[p]$ be the class such that $\delta^{\mathrm{C}(T)}([r], u) \subseteq [p]$. If $[p] \neq [r]$, then let $a \in \Sigma$ be the letter labeling the outgoing edge $([p], [p'], a)$ in the collecting tree. The action of $a$ on $[p]$ is thus the same as the action of $\gamma$. Hence, $\delta^{\mathrm{C}(T)}([r], u\gamma v) = \delta^{\mathrm{C}(T)}([r], uav)$. If $[p] = [r]$, then $\gamma$ acts as identity on $[p]$, thus $\delta^{\mathrm{C}(T)}([r], u\gamma v) = \delta^{\mathrm{C}(T)}([r], uv)$.

We can consider the $\gamma$ occurrences in any order and the obtained word $w$ have the property that $\delta^{\mathrm{C}(T)}([r], w') = \delta^{\mathrm{C}(T)}([r], w)$, thus synchronizes $[r]$ as $w'$ does. □

**Lemma 36.** *Let* $\mathcal{A} = (Q, \Sigma, \delta)$ *be a strongly connected partial automaton and let* $T$ *be one of its collecting trees. Then* $\mathcal{A}$ *is synchronizing if and only if the collecting automaton* $\mathcal{A}^{\mathrm{C}(T)} = (Q, \Sigma \cup \{\gamma\}, \delta^{\mathrm{C}(T)})$ *is synchronizing.*

This also means that the choice of $T$ does not matter: $\mathcal{A} = (Q, \Sigma, \delta)$ is synchronizing if and only if all $\mathcal{A}^{\mathrm{C}(T)}$ are synchronizing.

*Proof.* Let $[r]$ be the root class of $T$. Let $w \in \Sigma^*$ be a reset word for $\mathcal{A}$. Then there is some class $[p]$ such that $w$ synchronizes it. By Lemma 27, $\delta([p], w) = \delta^{\mathrm{F}}([p], w)$ is a singleton. Since $\mathcal{A}$ is strongly connected, there is a word $u_{r \to p}$ whose action maps all states from $[r]$ into $[p]$. Then $\gamma^{n-1} u_{r \to p} w$ is a synchronizing word for $\mathcal{A}^{\mathrm{C}(T)}$.

Conversely, if $\mathcal{A}^{\mathrm{C}(T)}$ is synchronizing, in particular there exists a word synchronizing $[r]$. By Lemma 35, there is also a word $w$ synchronizing $[r]$ in $\mathcal{A}^{\mathrm{C}(T)}$. By Corollary 25, we can find a word $v$ whose action maps $Q$ into a single equivalence class $[p]$. By strong connectivity, we can find a word $u_{p \to r}$ that maps all states from $[p]$ into $[r]$. Thus, $v u_{p \to r} w$ is a reset word for $\mathcal{A}$. □

### 4.3.5 Algorithmic Issues

Checking if a strongly connected partial automaton is synchronizing and finding a minimum-rank word can be done similarly as for a complete automaton, by a suitable generalization of the well-known Eppstein algorithm [37, Algorithm 1]. The same algorithm for checking synchronizability, under different terminology, was described in the context of $\varepsilon$-machines [109].

**Proposition 37.** *Checking if a given strongly connected partial automaton with $n$ states over an alphabet $\Sigma$ is synchronizing can be done in $\mathcal{O}(|\Sigma|n^2)$ time and $\mathcal{O}(n^2+|\Sigma|n)$ space. Finding a word of the minimum rank can be done in $\mathcal{O}(|\Sigma|n^3)$ time and $\mathcal{O}(n^2+|\Sigma|n)$ space.*

*Proof.* We modify the well-known Eppstein algorithm [37, Algorithm 1] by generalizing it to partial and not necessarily synchronizing automata. We mark all pairs of states $\{p,q\}$ such that $|\delta(\{p,q\},a)|=1$ for some $a\in\Sigma$ (note that this includes two possibilities of compression) and propagate the compressibility backwards. If all pairs are compressible, then the automaton is synchronizing. Then starting from $Q$, we repetitively apply a shortest word compressing a pair of states in the current subset. The algorithm stops when the subset is no longer compressible, and by Lemma 30, its size is equal to the minimal rank. □

Furthermore, the problem of checking the synchronizability of a strongly connected partial automaton can be reduced in smaller time to the case of a complete automaton.

**Theorem 38.** *Given a strongly connected partial automaton, we can construct in $\mathcal{O}(|\Sigma|n\log n)$ time a complete automaton that is synchronizing if and only if the given partial automaton is synchronizing.*

*Proof.* We can compute all inseparability classes in $\mathcal{O}(|\Sigma|n\log n)$ time. This is done by the Hopcroft minimization algorithm [53], if we interpret the partial DFA as a language-accepting DFA with an arbitrary initial state and the sink state $\perp$ that is its only final state.

Having computed the classes, we can construct a collecting automaton for an arbitrary collecting tree. Note that it can be done in $\mathcal{O}(|\Sigma|n)$ time by a breadth-first search from a class $[r]$. The desired property follows from Lemma 36. □

### 4.3.6 Černý Conjecture

The famous Černý conjecture is the rank conjecture for $r=1$. Let $\mathfrak{C}(n)$ be the maximum length of the shortest reset words of all $n$-state synchronizing complete automata. It is well known that $\mathfrak{C}(n)\geq(n-1)^2$ [28]. The Černý conjecture states that $\mathfrak{C}(n)=(n-1)^2$, but the best proved upper bound is cubic [103, 106].

Let $\mathfrak{C}_{\mathrm{P}}(n)$ be the maximal length of the shortest reset words of all $n$-state synchronizing strongly connected partial automata. We show that if the Černý conjecture is true (or another upper bound holds), then a slightly weaker upper bound holds for synchronizing strongly connected partial automata.

To prove the following theorem, we combine several techniques, in particular, the inseparability equivalence, the collecting automaton, and an algebraic upper bound on the reset threshold of a complete automaton with a word of small rank [12].

**Theorem 39.**

$$\mathfrak{C}_{\mathrm{P}}(n)\leq\mathfrak{C}(n)+\mathcal{O}(n^{4/3}).$$

*Proof.* Let $\mathcal{A}=(Q,\Sigma,\delta)$ be a synchronizing partial automaton with $n$ states. Let $T$ be a collecting tree of $\mathcal{A}$ with a root class $[r]$ containing the smallest number of states. We consider the collecting automaton $\mathcal{A}^{\mathrm{C}(T)}=(Q,\Sigma\cup\{\gamma\},\delta^{\mathrm{C}(T)})$. By Lemma 36, $\mathcal{A}^{\mathrm{C}(T)}$ is synchronizing. We consider two cases, depending on the number $\kappa(Q)$ of inseparability classes of $\mathcal{A}$.

First, suppose that $\kappa(Q)\leq n^{2/3}$. Then, by Corollary 25 (for $S=Q$), there is a word $v$ of length at most $(\kappa(Q)-1)\kappa(Q)/2<n^{4/3}$ such that $\delta(Q,v)$ is non-empty and is contained in one equivalence class, say $[p]$. Since $\mathcal{A}$ is strongly connected, there is a word $u_{p\to r}$ of length at most

$n-1$ whose action maps $[p]$ into $[r]$. Let $w'$ be a reset word for $\mathcal{A}^{\mathrm{C}(T)}$ of length at most $\mathfrak{C}(n)$. In particular, $w'$ synchronizes $[r]$, so by Lemma 35, we get a word $w$ of length at most $\mathfrak{C}(n)$ that synchronizes $[r]$ in $\mathcal{A}$. Then, $vu_{p\to r}w$ is a reset word for $\mathcal{A}$ of length at most $n^{4/3}+n-1+\mathfrak{C}(n)$.

In the second case, we have $\kappa(Q) > n^{2/3}$. Then the size of $[r]$, which has been chosen to have the smallest size, has at most $n^{1/3}$ states. Note that $\gamma^{n-1}$ is a word of rank at most $n^{1/3}$. Then we can apply [12, Theorem 2] ($d_1 = d_2 = n-1$) or Corollary 42 for $\mathcal{A}^{\mathrm{C}(T)}$ with this word, obtaining that the reset threshold of $\mathcal{A}^{\mathrm{C}(T)}$ is upper bounded by $(n-1)+2(n-1)\mathfrak{C}(n^{1/3})$. Using the well-known general cubic upper bound $n^3/6$ on the reset threshold of a complete automaton (e.g., [84]), we get that there is a reset word $w'$ for $\mathcal{A}^{\mathrm{C}(T)}$ of length at most $(n-1)+2(n-1)n/6$. Now, we return to $\mathcal{A}$. By Corollary 25 (for $S = Q$), we get a word $v$ of length at most $(n-1)n/2$ such that $\delta(Q,u)$ is non-empty and is contained in one equivalence class $[p]$. As in the first case, there is a word $u_{p\to r}$ of length at most $n-1$ whose action maps $[p]$ into $[r]$. By Lemma 35, from $w'$ we obtain a word $w$ that synchronizes $[r]$ and is no longer than $(n-1)+2(n-1)n/6$. Finally, $vu_{p\to r}w$ is a reset word for $\mathcal{A}$ of length at most $(n-1)n/2+(n-1)+(n-1)+2(n-1)n/6 \leq (n-1)^2$ for $n \geq 18$.

From both cases, we conclude that $\mathrm{rt}(\mathcal{A}) \leq \mathfrak{C}(n)+\mathcal{O}(n^{4/3})$. □

From Theorem 39, it follows that all upper bounds on the reset threshold of a complete automaton transfer to upper bounds for partial automata, up to a subquadratic component. Thus $\mathfrak{C}_{\mathrm{P}}(n) \leq 0.1654n^3+\mathcal{O}(n^2)$ [103]. It can be also seen from the proof that if we have a better general upper bound on $\mathfrak{C}(n)$, we get a smaller additional term, e.g., if $\mathfrak{C}(n) \in \mathcal{O}(n^2)$, then $\mathfrak{C}_{\mathrm{P}}(n) = \mathfrak{C}(n)+\mathcal{O}(n)$, so the additional term is at most linear. The additional component is likely not needed, but it is difficult to completely get rid of it in general, as for that we could not lengthen by any means the reset word assumed for a complete automaton. However, it is easy to omit it when reproving known bounds for complete automata, both combinatorial [84] and based on avoiding words [103, 106]. We conjecture that $\mathfrak{C}_{\mathrm{P}}(n) = \mathfrak{C}(n)$ for all $n$.

### 4.3.7 Induced Automaton

We develop an algebraic technique applied to partial automata. It will allow us deriving upper bounds on reset thresholds, in particular, in the cases when there exists a short word of a small rank, which is the case of the literal automaton of a prefix code. We base on the results from [12] for complete automata and generalize them to be applied to partial automata. The existing linear algebraic proofs for complete automata do not work for partial ones, because the matrices of transitions may not have a constant sum of the entries in each row.

We need to introduce a few definitions from linear algebra for automata (see, e.g., [12, 60, 82, 106]). Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a partial DFA. Without loss of generality we assume that $Q = \{1, \ldots, n\}$. By $\mathbb{R}^n$, we denote the real $n$-dimensional linear space of row vectors. For a vector $v \in \mathbb{R}^n$ and an $i \in Q$, we denote the vector's value at the $i$-th position by $v(i)$. Similarly, for a matrix $M$, we denote its value in an $i$-th row and a $j$-th column by $M(i,j)$. A vector is *non-negative* if $g(i) \geq 0$ for all $i$, and it is *stochastic* if $\sum_{i=1}^{n} g(i) = 1$. For a word $w \in \Sigma^*$, by $\mathrm{M}(w)$ we denote the $n \times n$ matrix of the transformation of $w$ in $\delta$: $\mathrm{M}(w)(p,q) = 1$ if $\delta(p,w) = q$, and $\mathrm{M}(w)(p,q) = 0$ otherwise. Note that if $\delta(p,w) = \bot$, then we have $\mathrm{M}(w)(p,q) = 0$ for all $q \in Q$. The usual scalar product of two vectors $u, v$ is denoted by $u \odot v$. The linear subspace *spanned* by a set of vectors $V$ is denoted by $\mathrm{span}(V)$.

Given a transition function $\delta$ (which defines matrices $\mathrm{M}(w)$), call a set of words $W \subseteq \Sigma^*$ *complete for a subspace* $V \subseteq \mathbb{R}^n$ *with respect to a vector* $g \in V$, if $\mathrm{span}(\{g\mathrm{M}(w) \mid w \in W\}) = \mathrm{span}(\{g\mathrm{M}(w) \mid w \in \Sigma^*\}) = V$. Notice that due to the linear extending argument, $\Sigma^{\leq n-1}$ is complete for every $g \in V$. A set of words $W \subseteq \Sigma^*$ is *complete for a subspace* $V \subseteq \mathbb{R}^n$ if for every non-negative

stochastic vector $g \in V$, $W$ is complete for $V$ with respect to $g$. Let $\chi(p)$ denote the characteristic (unitary) vector of $\{p\}$. For a subset $S \subseteq Q$, we define $\mathbb{V}(S) = \mathrm{span}(\{\chi(p) \mid p \in S\}) \subseteq \mathbb{R}^n$.

For example, consider the automaton from Fig. 4.1(left). Let $V = \mathbb{V}(\{q_1, q_2, q_4\})$ and let $W = \{\varepsilon, b, bab\}a^*$. Let $g \in V$ be a non-negative stochastic vector. Using that the transition of $b$ is undefined for $q_3$, depending on $g$, we can choose $u_1 \in \{\varepsilon, b, bab\}$ so that $g\mathrm{M}(u_1)$ has exactly one non-zero component, say $i \in \{1, 2, 3\}$; e.g., if $g$ has two non-zero entries of $q_1$ and $q_3$, then $u_1 = b$. Then, for each $j \in \{1, 2, 3\}$, the vector $g\mathrm{M}(u_1 a^{6-i+j})$ has a unique non-zero component corresponding to $q_j$, thus $W$ is complete for $V$ with respect to $g$.

The induced automaton of a partial one is another automaton acting on a subset of states $R \subseteq Q$. The alphabet consists of words, whose actions always keep the states mapped back into $R$. We can analyze an induced automaton as a separate one, and synchronize the whole automaton using it, which is particularly profitable when $R$ is small.

**Definition 40** (Induced automaton)**.** *Let $W_1, W_2 \subseteq \Sigma^*$ be non-empty and $R = \{\delta(q, w) \mid q \in Q,\ w \in W_1, \delta(q, w) \neq \bot\}$. If $R$ is non-empty, we define the* induced automaton $\mathcal{A}^{\mathrm{I}(W_1,W_2)} = (R, W_2W_1, \delta_{\mathcal{A}^{\mathrm{I}(W_1,W_2)}})$*, where the transition function is defined in compliance with the actions of words in $\mathcal{A}$, i.e., $\delta_{\mathcal{A}^{\mathrm{I}(W_1,W_2)}}(q, w) = \delta(q, w)$ for all $q \in R$ and $w \in W_2W_1$.*

The induced automaton $\mathcal{A}^{\mathrm{I}(W_1,W_2)}$ is $\mathcal{A}$ restricted to $R$. Note that its transition function is well defined, which is ensured by the fact that every word of the form $w_2w_1 \in W_2W_1$ has the action mapping every state $q \in Q$ into $R$ or to $\bot$.

The following lemma states that the completeness property of a set of words in the whole automaton transfers to the induced automaton. This is straightforward in the case of a complete automaton [12], because the whole subspace $\mathbb{V}(Q)$ transformed by the matrices of $W_1$ yields $\mathbb{V}(R)$, but it requires a new argument in the case of a partial automaton.

**Lemma 41.** *Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a strongly connected synchronizing partial automaton and let $W_1$ and $W_2$ be two non-empty sets of words over $\Sigma$. Let $\mathcal{B} = \mathcal{A}^{\mathrm{I}(W_1,W_2)} = (R, W_2W_1, \delta_{\mathcal{A}^{\mathrm{I}(W_1,W_2)}})$ be the induced automaton. If $W_2$ is complete for $\mathbb{V}(Q) = \mathbb{R}^n$ with respect to a stochastic non-negative vector $\alpha \in \mathbb{V}(R)$ (e.g., $W_2 = \Sigma^{\leq n-1}$), then $W_2W_1$ is complete for $\mathbb{V}(R)$ with respect to $\alpha$, and $\mathcal{B}$ is synchronizing and strongly connected.*

*Proof.* First, let us prove the completeness of $W_2W_1$ with respect to an arbitrary non-negative stochastic vector $\alpha \in \mathbb{V}(R)$. By the definition of $R$, for each state $r \in R$, there is a state $q_1$ and a word $u_1 \in W_1$ such that $\delta(q_1, u_1) = r$. Since $\mathcal{A} = (Q, \Sigma, \delta)$ is synchronizing and strongly connected, there is a synchronizing word $u_2$ such that $\delta(Q, u_2) = \{q_1\}$. Since the image is non-empty, there is a state $q_2 \in Q$ such that $\delta(q_2, u_2) = q_1$. Let $q_3 \in R$ be a state such that $\chi(q_3) \odot \alpha \neq 0$ (since $\alpha \in \mathbb{V}(R)$ is stochastic, there must be a non-zero entry). Again by strong connectivity of $\mathcal{A}$, there is a word $u_3$ whose action maps $q_3$ to $q_2$. Hence, $u_3u_2u_1$ is a synchronizing word with the action mapping $q_3$ to $r$. Thus $\alpha\mathrm{M}(u_3u_2u_1) \in \mathrm{span}(\{\chi(r)\})$, and since $\alpha$ and $\mathrm{M}(u_3u_2u_1)$ are also non-negative, $(\alpha\mathrm{M}(u_3u_2u_1))(r) \neq 0$. Since this holds for every $r \in R$, that is, for every $r$ we can find a word $u_3u_2u_1$ that yields a vector with non-zero $r$-th entry and all the other entries containing zero. Moreover, these words are from $\Sigma^*W_1$, thus we get $\mathrm{span}(\{\alpha\mathrm{M}(w) \mid w \in \Sigma^*W_1\}) = \mathbb{V}(R)$. As $W_2$ is complete for $\mathbb{V}(Q)$ with respect to $\alpha$, we have $\mathrm{span}(\{\alpha\mathrm{M}(w) \mid w \in \Sigma^*\}) = \mathbb{V}(Q) = \mathrm{span}(\{\alpha\mathrm{M}(w) \mid w \in W_2\})$, thus $\mathrm{span}(\{\alpha\mathrm{M}(w) \mid w \in W_2W_1\}) = \mathbb{V}(R)$.

It remains to prove that $\mathcal{B}$ is synchronizing. If $|R| = 1$, we are done. If $R$ is compressible in $\mathcal{B}$ by a word $w$, we can use induction proving that the induced automaton for $(W_1\{w\}, W_2)$ is synchronizing. Otherwise, for each word $v \in W_2W_1$, we have $\delta(R, v) \in \{R, \emptyset\}$. This contradicts with $W_2W_1$ being complete for $\mathbb{V}(R)$ since a vector $\alpha\mathrm{M}(v)$ either has sum of the entries equal 1 or it is the zero vector. Hence $R$ is compressible in $\mathcal{B}$ and the statement follows by induction. □

The following corollary directly follows from Lemma 41. It is useful for deriving upper bounds for automata with a word of small rank; we show its application in the next subsection.

**Corollary 42.** *Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a strongly connected synchronizing partial automaton with $n$ states, and let $w \in \Sigma^*$ be a word such that $R = \delta(Q, w) \neq \emptyset$. Let $W_1 = \{w\}$, $W_2 = \Sigma^{\leq n-1}$, and $\mathcal{B} = \mathcal{A}^{\mathrm{I}(W_1,W_2)} = (R, \Sigma^{\leq n-1}\{w\}, \delta_{\mathcal{A}^{\mathrm{I}(W_1,W_2)}})$ be the induced automaton. Then $\mathrm{rt}(\mathcal{A}) \leq |w| + (|w| + n - 1) \cdot \mathrm{rt}(\mathcal{B})$.*

### 4.3.8 The Literal Automaton of a Finite Prefix Code

Let $X$ be a finite prefix code over an alphabet $\Sigma$, and $\mathcal{A}_X = (Q, \Sigma, \delta)$ be its literal automaton. The state corresponding to the empty prefix $\varepsilon$ is called the *root state*. The *height* of a literal automaton is the length of a longest path of its transitions without repetition of states; equivalently, this is the length of the longest word in $X$ minus one. Note that the number of states of $\mathcal{A}_X$ is at most the total length of all codewords of $X$. An example of a literal automaton is shown in Fig. 4.1(right).

It is proved that every complete literal automaton over an alphabet of size $k$ has a word of rank and length at most $\lceil \log_k n \rceil$ ( [12, Lemma 16], cf. [17, Lemma 14]). This is no longer true for non-mortal words in partial literal automata [98]. We prove that there exist $\mathcal{O}(\log_2 n)$-rank non-mortal words of length $\mathcal{O}(n)$ in such automata, excluding the case of a code with only one word. Then we use this result to provide an $\mathcal{O}(n \log_2^3 n)$ upper bound on the reset threshold of $n$-state synchronizing partial literal automata, matching the known upper bound for complete literal automata [12].

We start with a special case of one-word codes. A non-empty word $w$ is called *primitive* if it is not a power of another word, i.e., $w \neq u^k$ for every word $u$ and $k \geq 2$. The upper bound on $\mathrm{rt}(\mathcal{A}_X)$ follows from a result of Weinbaum ( [49, 117]).

**Proposition 43.** *Let $X = \{x\}$ be a one-word prefix code, and suppose that $x = y^k$, where $y$ is a non-empty primitive word and $k \geq 1$. Then $\mathcal{A}_X$ has rank $k$. If $\mathcal{A}_X$ is synchronizing, then $\mathrm{rt}(\mathcal{A}_X) \leq \frac{|x|}{2}$, and this bound is tight.*

*Proof.* Observe that the literal automaton of a one-word code forms as a single cycle labeled by the letters of this word. No two states can be mapped to a single state by the action of any letter, which excludes pair compression.

Observe that $\mathcal{A}_X/_{\equiv}$ is the literal automaton $\mathcal{A}_Y$ of $Y = \{y\}$. In $\mathcal{A}_X$, there are $n/k$ inseparability classes, each of size $k$. Hence, the rank of $\mathcal{A}_X$ equals $k$ times the rank of $\mathcal{A}_Y$. Since $y$ is primitive, its action is defined only for the root state (and maps the root to itself). Thus $\mathcal{A}_Y$ is synchronizing, so the rank of $\mathcal{A}_X$ equals $k$.

Now, we will bound the length of the shortest reset words. A word $t$ is called a *conjugate* of $t'$ if $t = uv$ and $t' = vu$ for some words $u, v$. By a result of Weinbaum, stated in terms of automata, every primitive word $x$ has a conjugate $x' = uv$ such that both $u$ and $v$ have the action defined exactly for one state of $\mathcal{A}_X$ [49]. Thus, both $u$ and $v$ are reset for $\mathcal{A}_X$. The shorter of them has length at most $\frac{|x|}{2}$.

To see that the bound can be met, consider the code $Z = \{a^k b a^{k+1} b\}$. The shortest reset word for $\mathcal{A}_Z$ is $a^{k+1}$ of length $\lfloor \frac{2k+3}{2} \rfloor = k + 1$. □

In the remaining cases, there always exists a word of linear length and logarithmic rank.

**Theorem 44.** *Let $X$ be a prefix code with at least two words. Let $\mathcal{A}_X = (Q, \Sigma, \delta)$ be its partial literal automaton with $n$ states and height $h$. Then there exists a word of length at most $2h$ and of rank at most $\lceil \log_2 hn \rceil + \lceil \log_2 h \rceil$ for $\mathcal{A}_X$. Moreover, such a word can be found in polynomial time in $n = |Q|$.*

*Proof.* Since $X$ consists of at least two words, there exists a state $p$ such that at least two letters have defined the transition from it. We choose $p$ to be a state with this property that is at the minimal distance from the root state $r$, i.e., such that the length of the shortest word $w$ such that $\delta(r, w) = p$, is the smallest possible. Let $a, b \in \Sigma$ be two letters with defined transitions going from $p$. Since each state on the path from the root to $p$ has only one letter with a defined outgoing transition, such a state $p$ is unique. For the literal automaton in Fig. 4.1(right), the chosen state $p$ is $q_3$.

**Filtering algorithm.** Consider the following auxiliary algorithm having a word $w \in \Sigma^*$ as the input. We perform steps for $i = 1, 2, \ldots$. In each step, we keep a subset of *active* states $S_i \subseteq Q$, and two words $u_i$, $w_i$. At the beginning, $S_1 = Q$, $u_1 = \varepsilon$, and $w_1 = w$. In an $i$-th step, we have two cases.

*(Case 1)* If $p$ is active, then let $a$ be the first letter of $w_i$, assuming $w_i$ is non-empty. We apply the transition of $a$ to $S_i$, and let $S_{i+1} = \delta(S_i, a)$ for the next iteration. Also, we move the first letter from $w_i$ to the end of $u_i$, thus we set $u_{i+1} = u_i a$ and $w_{i+1}$ to be such that $w_i = a w_{i+1}$.

*(Case 2)* If $p$ is not active, then let $y$ be the smallest letter, under some fixed order on $\Sigma$ (the same for every step), that is non-mortal for $S_i$, i.e., $\delta(S_i, y) \neq \bot$. Such a letter always exists in a literal automaton, as for every state we can find a non-mortal letter. We apply $y$ to $S_i$, so $S_{i+1} = \delta(S_i, y)$, and set $u_{i+1} = u_i y$ and $w_{i+1} = w_i$.

*(Termination)* If $w_{i+1}$ is empty or $u_{i+1}$ has length at least $h$, we stop. The word $u_{i+1}$ is the output of the algorithm.

To see that the algorithm terminates, observe that at every step the length of $u_i$ is increased by 1, thus there are at most $h$ iterations.

For a word $z$, denote by $\alpha(z)$ the word $u_{i+1}$ obtained in the last step of the algorithm. Note that $\delta(Q, u_i) = S_i$ holds in every step. Since every time we apply a letter that is non-mortal for the set of active states, this word is non-mortal for $\mathcal{A}_X$. It has also length at most $h$. Observe that for two different words $v_1, v_2$, either the words $\alpha(v_1)$, $\alpha(v_2)$ are different or they have length $h$. Indeed, if they have a length smaller than $h$, then the algorithm terminates when $w_{i+1}$ is empty. Consider the longest common prefix $v$ of $v_1$ and $v_2$, so $v_1 = v v_1'$ and $v_2 = v v_2'$. The algorithm executes in the same way up to the $|v|$-th time when Case 1 is applied. Then it performs some iterations with Case 2, which do not decrease the length of $w_i$. Finally, Case 1 must hold, and the algorithm applies the first letters of $v_1'$ and $v_2'$. At this moment, the two words constructed become different.

**All-through-root word construction.** Now, we show that there exists a word $w$ of length $\lceil \log_2 hn \rceil$ such that, for every state $q \in Q$, either the action of some prefix of $\alpha(w)$ maps $q$ to the root $r$ or the action of $\alpha(w)$ is undefined for $q$, and the word $\alpha(w)$ is non-mortal. Recall two letters $a, b \in \Sigma$ such that state $p$ has a defined outgoing transition under both of them. Consider the set of the resulting words $W = \{\alpha(w) \mid w \in \{a, b\}^{\lceil \log_2 hn \rceil}\}$. If there is at least one word of length $h$ in $W$, we are done. Otherwise, all words in $W$ are of a length less than $h$, and there are at least $hn$ pairwise distinct words. Then, by the pigeonhole principle, there is a subset $W' \subseteq W$ consisting of at least $n + 1$ words of equal length. Suppose for a contradiction that for every word $\alpha(w) \in W'$, there is a state $q_{\alpha(w)}$ such that $\delta(q_{\alpha(w)}, \alpha(w)) \neq \bot$ and that is not mapped to $r$ by the action of any prefix of $\alpha(w)$. Note that in a literal automaton of a prefix code, every state except the root state has an ingoing transition for exactly one letter. Thus for a pair $q$, $q'$ of states and two words $t$, $t'$ of the same length, we can have $\delta(q, t) = \delta(q', t) \neq \bot$ only if some prefix of $t$ (or $t'$) has the action mapping $q$ (or $q'$, respectively) to the root state. Therefore, all states $q_{\alpha(w)}$ for $\alpha(w) \in W'$ are pairwise distinct, and we have at least $n + 1$ such states. This means that the number of states of $\mathcal{A}_X$ is larger than $n$, which is a contradiction. Thus, we get a non-mortal word $\alpha(w)$ of length at most $h$ with the property that every state $q \in Q$ with $\delta(q, \alpha(w)) \neq \bot$ is mapped to the root state by some prefix of $\alpha(w)$.

**Splitting the image.** We consider the set $\delta(Q, \alpha(w))$ and split it into two sets, one with specific states and the other one with logarithmic size. Let $P \subseteq Q$ be the set of all states in the unique shortest path from $r$ to $p$ excluding $p$ itself. For the literal automaton in Fig. 4.1(right), $P = \{q_1, q_2\}$. The size of $P$ is at most $h$. Note that, by the selection of $p$, for each state in $P$, exactly one letter has a defined transition. Thus, the other set $Q \setminus P$ has the property that the unique shortest path from $r$ to each state from $Q \setminus P$ contains $p$. Note that $\delta(Q, \alpha(w)) \cap (Q \setminus P)$ has up to $\lceil \log_2 hn \rceil$ states, since by the construction of $\alpha(w)$, at most $|w|$ prefixes of $\alpha(w)$ have the action mapping some state to $p$ (when Case 1 holds).

**Compressing the part in $P$.** Denote by $R$ the image $\delta(Q, \alpha(w))$. We are going to construct a word $v$ of length at most $h$ such that $1 \leq |\delta(P \cap R, v)| \leq \lceil \log_2 h \rceil$. However, if $P \cap R = \emptyset$, then we skip this phase and let $v = \varepsilon$.

Consider the following procedure constructing an auxiliary word $v$. We perform steps $i = 1, 2, \ldots$, and in an $i$-th step keep a word $v_i$ and a set $S_i$ of *active* states. At the beginning, we set $v_1 = \varepsilon$ and $S_1 = R \cap P$. In an $i$-th step, we do the following. Let $\ell_i$ be the shortest word mapping some active state from $P$ to $p$. We apply $\ell_i$ to the set $S$ of active states and let $S' = \delta(S, \ell_i)$. Let $\ell'_i \in \{a, b\}$ be a letter whose transition is undefined for at least half of the states in $S' \cap P$ (unless $S' \cap P = \emptyset$). At least one of these two letters have this property since all states in $P$ have only one defined transition. We set $v_{i+1} = v_i \ell_i \ell'_i$ and $S_{i+1} = \delta(S' \setminus \{p\}, \ell'_i)$ for the next step; note that we additionally remove $p$ from the set of active states. If the length of $v_{i+1}$ is at least $h$ or $S_{i+1} = \emptyset$, we stop. Otherwise, we proceed to the $(i+1)$-th step for $S_{i+1} \subset P$ of size at most $\lfloor |S_i|/2 \rfloor$.

Denote the word $v_{i+1}$ constructed at the last step by $v$. By the construction, the word $\alpha(w)v$ is non-mortal, because after each step, $p \in \delta(P \cap R, v_{i+1})$. The number of active states decreases at least twice in every step, thus the number of performed steps is at most $\lceil \log_2 h \rceil$. Furthermore, in every step, exactly one active state from $P$ is mapped to a state in $Q \setminus P$, since we use the shortest word such $\ell_i$. This state is always $p$, which we remove from the active states. The number of these removed states is at most $\lceil \log_2 h \rceil$ and we know that the other states from $R \cap P$ are mapped to $\perp$ in some step, thus the cardinality of $\delta(P \cap R, v)$ is at most $\lceil \log_2 h \rceil$.

To bound the length of $v$, we observe that the sum of the lengths of words $\ell_i \ell'_i$ from every iteration cannot exceed $h$. It is because every active state, if it is not mapped to $\perp$, is mapped by the action of $\ell_i \ell'_i$ to a state farther from the root by $|\ell \ell'_i|$. Thus, the sum of the lengths cannot exceed $h$.

**Summary.** It follows that the rank of $\alpha(w)v$ is at most $\lceil \log_2 hn \rceil + \lceil \log_2 h \rceil$: Indeed, the states in $Q$ that are mapped into the set $P$ by the action of $\alpha(w)$ are then mapped by the action of $v$ to a set of cardinality at most $\lceil \log_2 h \rceil$. All the other states are mapped by the action of $\alpha(w)$ to a set of size at most $\lceil \log_2 hn \rceil$.

Finally, note that the word $\alpha(w)v$ can be found in polynomial time. In particular, to find $\alpha(w)$, we check $2^{\lceil \log_2 hn \rceil} = \mathcal{O}(hn)$ different words, and every word is processed in polynomial time by the auxiliary algorithm computing $\alpha$. □

**Corollary 45.** *Let $\mathcal{A}_X$ be a partial literal automaton with $n$ states. If it is synchronizing, its reset threshold is at most $\mathcal{O}(n \log_2^3 n)$.*

*Proof.* If $|X| = 1$ then the bound follows from Proposition 43. If $|X| \geq 2$, from Theorem 44, we get a word $w$ of length $\mathcal{O}(n)$ and rank $\mathcal{O}(\log n)$. Then we use Corollary 42 with $w$, which yields the upper bound $\mathcal{O}(n) + \mathcal{O}(n) \cdot \mathrm{rt}(\mathcal{B})$, where $\mathcal{B}$ is an induced automaton with $\mathcal{O}(\log n)$ states. Then we use a cubic upper bound on the reset threshold of a complete DFA ( [103, 106]) transferred to strongly connected partial DFAs by Theorem 39. □

## 4.4 Lower Bounds for Properly Incomplete Automata

Finally, we note about transferring lower bounds from the complete case to the partial case. Of course, in general, this is trivial, since a complete automaton is a special case of a partial one. On the other hand, letters with totally undefined transitions cannot be used for synchronization thus are useless. Hence we need to add a restriction to exclude these cases and see the effect of usable incomplete transitions. A partial automaton is *properly incomplete* if it is not complete and there is at least one letter whose transition is defined for some state and is undefined for some other state.

We will show that bounding the rank/reset threshold of a strongly connected properly incomplete automaton is related to bounding the corresponding threshold of a complete automaton.

**Definition 46** (Duplicating automaton)**.** *For a complete automaton $\mathcal{A} = (Q, \Sigma, \delta_\mathcal{A})$, we construct the* duplicating automaton $\mathcal{A}^\mathrm{D} = (Q \cup Q', \Sigma \cup \{\gamma\}, \delta_{\mathcal{A}^\mathrm{D}})$ *as follows. Assume that $Q = \{q_1, \ldots, q_n\}$. Then $Q' = \{q'_1, \ldots, q'_n\}$ is a set of fresh states disjoint with $Q$ and $\gamma \notin \Sigma$ is a fresh letter. For all $1 \leq i \leq n$ and $a \in \Sigma$, we define: $\delta_{\mathcal{A}^\mathrm{D}}(q_i, a) = q_i$, $\delta_{\mathcal{A}^\mathrm{D}}(q_i, \gamma) = q'_i$, $\delta_{\mathcal{A}^\mathrm{D}}(q'_i, a) = \delta_\mathcal{A}(q_i, a)$, and $\delta_{\mathcal{A}^\mathrm{D}}(q'_i, \gamma) = \bot$.*

The duplicating automaton $\mathcal{A}^\mathrm{D}$ has twice the number of states of $\mathcal{A}$ and is properly incomplete. Also, it is strongly connected if $\mathcal{A}$ is.

**Proposition 47.** *Let $\mathcal{A} = (Q, \Sigma, \delta_\mathcal{A})$ be a strongly connected complete automaton. For all $1 \leq r < n$, we have $\mathrm{rt}(\mathcal{A}^\mathrm{D}, r) = 2\,\mathrm{rt}(\mathcal{A}, r)$.*

*Proof.* It is easy to observe that if $\mathcal{A}$ is strongly connected, then so is $\mathcal{A}^\mathrm{D}$.

Let $1 \leq r < n$, and let $w$ be a shortest word of rank at most $r$ in $\mathcal{A}^\mathrm{D}$. We observe that $w = \gamma a_1 \gamma a_2 \gamma a_3 \ldots \gamma a_k$, for some letters $a_i \in \Sigma$. Indeed, the action of $a_i a_j$ is the same as the action of $a_i$, thus two consecutive letters from $\Sigma$ cannot occur in a shortest word. Also, $\gamma^2$ cannot occur, as it is a mortal word. There is no letter $a_i$ at the beginning of $w$, because $\delta_{\mathcal{A}^\mathrm{D}}(Q, a_i\gamma) = \delta_{\mathcal{A}^\mathrm{D}}(Q, \gamma) = Q'$. Finally, $w$ does not contain $\gamma$ at the end, because otherwise the rank would be the same, i.e., for all $S \subseteq Q'$, we have $|\delta_{\mathcal{A}^\mathrm{D}}(S, a_i\gamma)| = |\delta_{\mathcal{A}^\mathrm{D}}(S, a_i)|$.

We observe that the word $w' = a_1 a_2 \ldots a_k$ has rank $r$ in $\mathcal{A}$. Indeed, $\delta_{\mathcal{A}^\mathrm{D}}(Q \cup Q', w) \subset Q$, and $q_i \in \delta_{\mathcal{A}^\mathrm{D}}(Q \cup Q', w)$ if and only if $q_i \in \delta_\mathcal{A}(Q, w')$. Since $w$ is a word of rank $r$ in $\mathcal{A}^\mathrm{D}$, so is $w'$ in $\mathcal{A}$. Thus $\mathrm{rt}(\mathcal{A}^\mathrm{D}, r) = |w| = 2|w'| \geq 2\,\mathrm{rt}(\mathcal{A}, r)$.

Conversely, having a shortest word $w' = a_1 a_2 \ldots a_k$ of rank $r$ in $\mathcal{A}$, we can construct $w = \gamma a_1 \gamma a_2 \gamma a_3 \ldots \gamma a_k$ or rank $r$ in $\mathcal{A}^\mathrm{D}$, thus $\mathrm{rt}(\mathcal{A}^\mathrm{D}, r) \leq 2\,\mathrm{rt}(\mathcal{A}, r)$. □

From Proposition 47, it follows that we cannot expect a better upper bound on the reset threshold of a properly incomplete strongly connected automaton than $0.04135n^3 + \mathcal{O}(n^2)$, unless we can improve the best general upper bound on the reset threshold of a complete automaton, which currently is roughly $0.1654n^3 + \mathcal{O}(n^2)$ [103, 106]. We can also show a lower bound on the largest possible reset threshold. There is a connection with a particular class of automata with long shortest mortal words [91].

**Proposition 48.** *For every $n$, there exists a strongly connected properly incomplete $n$-state automaton with reset threshold $\frac{n^2-n}{2}$.*

*Proof.* Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a strongly connected partial automaton with only one state $r$ having undefined outgoing transitions of some letter, and let $w$ be its shortest mortal word. Let $w = w'a$, where $a \in \Sigma$. Then $\delta(Q, w') = \{r\}$. Moreover, since the automaton is strongly connected, every prefix $w''$ of $w'$ of length smaller than $|w| - (|Q| - 1)$ has the action mapping $Q$ to a set of size at least two; otherwise there is a shorter word mapping $Q$ to $\{r\}$. Thus we get that $\mathcal{A}$ is synchronizing,

and the length of its shortest reset word is at least $|w| - (|Q| - 1) - 1 = |w| - |Q|$. For every $n$, Rystsov [91] constructed a strongly connected properly incomplete $n$-state automaton with the only one state having undefined outgoing transitions. The length of its shortest mortal word is equal to $\frac{n^2+n}{2}$. Thus we get the lower bound of $\frac{n^2+n}{2} - n$ on the length of its shortest reset word. It can be seen directly from the construction of the automaton that this bound is tight. □

# Chapter 5

# Extremal Bounds for Unambiguous NFAs and General Codes

Everywhere in the paper we assume the automata to be strongly connected.

## 5.1 Introduction

As described in Section 2.4, there is a natural correspondence between recognizable codes and strongly connected unambiguous NFAs. This correspondence allows to generalize many synchronization problems from deterministic to unambiguous automata. In this chapter we study the length of shortest words of minimum rank and minimum non-zero rank in strongly connected unambiguous NFAs, thus further generalizing the Rank Conjecture. In contrast to the case of strongly connected partial DFAs, a polynomial upper bound is not trivial in this case, since it is much harder to control the image of a subset of states. In particular, it is not possible anymore to simply decrease the size of the set of active states step by step. We show that a polynomial upper bound is still achievable, by extending the technique of Carpi [25] and Kiefer and Mascle [62]. Thus we partially answer a question stated in the list of research problems of [14], and obtain a generalization of the Černý conjecture to the case of strongly connected unambiguous NFAs. Our proof also implies that it can be checked in polynomial time whether a strongly connected unambiguous automaton is synchronizing. It also allows to get a better bounds for the case of finite codes.

The chapter is organized as follows. In Section 5.2 we present the idea of the proof of Kiefer and Mascle on bounding the lengths of mortal words in strongly connected unambiguous NFAs. In Section 5.3 we show how to extend this proof to obtain an $n^5$ upper bound on the length of shortest words of minimum non-zero rank. In Section 5.4 we refine the bounds on the length of shortest words of minimum rank for prefix automata of finite codes. In Section 5.5 we provide lower bounds on the length of shortest mortal words in several classes of deterministic automata. In Section 5.6 we prove a sufficient condition for recognizable codes to be synchronizing.

## 5.2 The result of Kiefer and Mascle

Let $\mathcal{A} = (Q, \Sigma, \Delta)$ be an $n$-state strongly connected unambiguous automaton. Fix an ordering $q_1, \ldots, q_n$ of the set $Q$. For a subset $S$ of states denote by $\mathrm{vect}(S)$ the $n$-dimensional *characteristic (row) vector* of $S$, that is, a vector $v \in \{0,1\}^n$ with $i$th component equal to 1 if $q_i \in S$ and equal to 0 otherwise.

A set $C \subseteq Q$ is called *mergeable* if there exists a word $w$ and a state $q'$ such that for every state $q \in R$ we have $q' \in q.w$. A set $R \subseteq Q$ is called *coreachable* if there exists a word $w$ and a state $q'$ such that for every state $q \in C$ we have $q \in q'.w$. Characteristic vectors of mergeable and coreachable sets are exactly non-zero columns and rows of matrices in the monoid $\mathcal{M}(\mathcal{A})$.

Let $y, z$ be a pair of words. We denote by $\mathcal{P}(y, z)$ the set of all states $q$ which are reached by $y$ and survive $z$, that is such that $y.q$ and $q.z$ are non-empty. Studying $\mathcal{P}(y, z)$ is the central idea of the discussed result. Assume without loss of generality that $\mathcal{P}(y, z) = \{q_1, \ldots, q_k\}$. Then there exist two families of non-empty sets: a family $\mathcal{C} = \{C_1, \ldots, C_k\}$ of mergeable and a family $\mathcal{R} = \{R_1, \ldots, R_k\}$ of coreachable sets such that $C_i = y.q_i$, $R_i = q_i.z$ for $1 \leq i \leq k$.

The rank of the word $yz$ is at most $k$. Indeed, we can represent the matrix $M(yz)$ as a product of the matrix with columns $\mathrm{vect}(C_1)^T, \ldots, \mathrm{vect}(C_k)^T$ and the matrix with rows $\mathrm{vect}(R_1), \ldots, \mathrm{vect}(R_k)$ (by $v^T$ we denote the column which is the transposition of the row $v$):

$$M(yz) = \mathrm{vect}(C_1)^T \cdot \mathrm{vect}(R_1) + \ldots + \mathrm{vect}(C_k)^T \cdot \mathrm{vect}(R_k).$$

Without additional restrictions on the words $y$ and $z$, the sets from $\mathcal{C}$ and $\mathcal{R}$ can intersect. However, the relations between the sets $C_i$ and $R_j$ are not arbitrary. First, for a mergeable set $C$ and a coreachable set $R$ the intersection $C \cap R$ has size at most one. Second, if $C_i$ intersects $C_j$ then $R_i$ and $R_j$ are disjoint and vice versa. Some further results in this direction are obtained in [85].

In [62] Kiefer and Mascle show how to construct a pair $y, z$ of words of length $O(n^4)$ such that no pair of states in $\mathcal{P}(y, z)$ is coreachable or mergeable. In particular this means that no pair of sets in $\mathcal{C}$ and no pair of sets in $\mathcal{R}$ intersects. The algorithm decreases the number of states in $\mathcal{P}(y', z')$ by iteratively constructing the words $y', z'$. This is done by concatenating words $z_q$ for different states $q$ with the property that no state coreachable with $q$ survives $z_q$. Thus we obtain the word $z$ forcing that no pair of states in $\mathcal{P}(\epsilon, z)$ is coreachable. The same is done symmetrically for mergeable states by the word $y$. Such a pair $y, z$ allows to manipulate unambiguous automata much easier. In particular, one can kill a state in $\mathcal{P}(y, z)$ by a word of linear length (Lemma 14 of [62]). The final mortal word is then obtained by subsequently killing all the remaining states in $\mathcal{P}(y, z)$. The rank of $yz$ for such $y, z$ is exactly the cardinality of $\mathcal{P}(y, z)$.

The proof of Carpi [25] (with some small modifications) can be seen as a similar algorithm. The main difference is that the structure of complete automata (provided in particular by Proposition 3.4 of [25]) makes it possible to get a better upper bound. In particular, it is enough to perform only $r$ steps of the algorithm producing the pair $y, z$, and there is no need then to kill the states in $\mathcal{P}(y, z)$ afterwards.

## 5.3 Words of minimum non-zero rank

In this section we extend the result of Kiefer and Mascle described in Section 5.2 to words of minimum non-zero rank. First we observe that the word $yz$ constructed in Lemma 11 of [62] is not mortal. Indeed, every word $w_q$ produced in Lemma 10 of [62] does not kill the state $q$. Then the algorithm in Lemma 11 consists of taking a state $p.w$ such that $p$ survives $w$, and applying $w_{p.w}$ to it, thus $p$ is not killed by $ww_{p.w}$. After getting rid of coreachable states the symmetric version for mergeable states which are not yet killed is performed, and by the same reason it is still maintained that some state survives.

Let now $w$ be a word of minimum non-zero rank $r$ such that at least one state $q_i \in \mathcal{P}(y, z)$ is not killed by $zwy$. Such a word always exists, since we can take $w = w_1 w' w_2$ where $w'$ is a word of minimum non-zero rank, and the words $w_1$ and $w_2$ are such that $w_1$ maps a state $q_i.z$ to a state

which is not killed by $w'$, and $w_2$ maps a state in $q_i.zw_1w'$ to a state surviving $y$. The pair $yzwy, z$ has then the same properties as $y, z$ but the cardinality of the set $\mathcal{P}(y, zwyz)$ is exactly $r$. This means there exist exactly $r$ states in $\mathcal{P}(y, z)$ which are mapped by $zw$ to $y.\mathcal{P}(y, z)$, and all the other states are mapped to the complement of this set. Let $q_i, q_j$ be a pair of states such that $zw$ sends exactly one of them to $y.\mathcal{P}(y, z)$, and the other one to the complement of this set. The next lemma is an analogue of Lemma 14 from [62] for the case of words of minimum non-zero rank instead of mortal words.

**Lemma 49.** *One can compute in polynomial time a word $x \in \Sigma^*$ with $|x| \leq n$ such that exactly one of the sets $q_i.zxyz$ and $q_j.zxyz$ is empty.*

*Proof.* It is enough to compute a word $x$ such that exactly one of the sets $q_i.zx$ and $q_j.zx$ intersects $y.\mathcal{P}(y, z)$. Define $e_i, e_j$ and $f$ to be the characteristic vectors of $q_i.z$, $q_j.z$ and $y.\mathcal{P}(y, z)$ respectively. As shown in the proof of Lemma 14 of [62], we have that $e_iM(x)f^T \leq 1$ and $e_jM(x)f^T \leq 1$. Since all the entries are non-negative and integer, the only two possible values are thus 0 and 1.

Let $V$ be the subspace of $\mathbb{R}^n$ spanned by the vectors $(e_i - e_j)M(x)$ for $x \in \Sigma^*$. This subspace can be seen as the smallest subspace containing the vector $e_i - e_j$ and closed under multiplying by $M(a)$ for all $a \in \Sigma$. Thus we can compute a basis $B$ of $V$ by subsequently finding for an already added word $u$ a letter $a \in \Sigma$ such that $(e_i - e_j)M(ua)$ is not in the already generated subspace and adding this vector to the basis. Since the dimension of $V$ is at most $n$, every vector in this basis corresponds to a word of length at most $n$.

As shown before, the possible values of $(e_i - e_j)M(x)f^T$ belong to the set $\{-1, 0, 1\}$. Since there exists a word $w$ (defined above) with $(e_i - e_j)M(w)f^T \neq 0$, we have that $V$ is not orthogonal to $f$. Thus there exists a word $x$ corresponding to a vector in $B$ such that $(e_i - e_j)M(x)f^T \in \{-1, 1\}$, and $|x| \leq n$. The word $x$ has then the required properties. □

Now we can follow the proof of [62] up to using Lemma 14, and then use Lemma 49 of this paper instead. This lemma allows to kill one state of the set $\mathcal{P}(y, z)$ while maintaining that the constructed word is not mortal. Thus we can proceed to Lemma 15 of [62] and construct iteratively a word of minimum non-zero rank having the same upper bound of $n^5$ on its length. We summarize these results in the following theorem.

**Theorem 50.** *Every strongly connected unambiguous automaton $\mathcal{A}$ with $n$ states has a word of minimum non-zero rank of length at most $n^5$. Such a word can be found in polynomial time.*

In particular, this means that it can be checked in polynomial time whether a finite code is synchronizing (for example, by checking that its prefix automaton is synchronizing, see Section 5.4 for the definition). It is natural to ask whether a faster algorithm to check synchronizability of a finite code (or a strongly connected unambiguous automaton in general) exists.

The main contribution of Theorem 50 is the first known polynomial upper bound on the length of shortest words of minimum non-zero rank for the non-complete case. Additionally it can be seen as a way to make the proof of Kiefer and Mascle (in the general case of words of minimum rank) more uniform, since it still works for the complete case and does not require the proof of Carpi for the special case of complete automata.

If a non-complete strongly connected deterministic automaton has a synchronizing word of length $\ell$, then it has a mortal word of length at most $\ell + n$, where $n$ is the number of states. To find this word it is enough to kill the only state in the image of the synchronizing word, which can be done by a word of linear length. A similar relation for strongly connected unambiguous automata is provided by the following result.

**Proposition 51.** *Let $\mathcal{A}$ be an $n$-state strongly connected non-complete synchronizing unambiguous automaton. If $\mathcal{A}$ has a synchronizing word of length $\ell$, then it has a mortal word of length $3\ell + n$.*

*Proof.* Let $w$ be a synchronizing word for $\mathcal{A} = (Q, \Sigma, \Delta)$. If $ww$ is mortal, we are done. Otherwise there exists exactly one state $q$ which survives $w$ and is reached from some state by $w$. Indeed, by definition every word of rank 1 has sets $C, R$ of states such that $Q.w = R$, $w.Q = C$ and $w$ maps every state in $C$ to every state in $R$. Thus, if there are two different states in $R \cap C$, the automaton is not unambiguous. Hence we can apply Lemma 14 of [62] for $y = w$ and $z = w$ and get a word $x$ of length at most $n$ such that the word $wxww$ of length at most $3\ell + n$ is mortal. □

Obviously, no relation in the other direction exists in the general case: one can consider an arbitrary synchronizing complete deterministic automaton and add a nowhere defined new letter to it. Then the length of a shortest synchronizing word is not bounded by any non-trivial function of the length of a shortest mortal word which is equal to 1.

## 5.4 Prefix automata

The next lemma generalizes the log-log lemma (Lemma 16 of [12], see also [17]) from complete prefix codes to general complete codes.

**Lemma 52** (Log-log lemma for general complete codes)**.** *Let $\mathcal{A}$ be the prefix automaton of a finite code $X$ over an alphabet of size $m$. Then there exists a word of length $\lceil \log_m n \rceil$ and rank at most $\lceil \log_m n \rceil$ for this automaton, where $n$ is the number of its states.*

*Proof.* Denote $r = \lceil \log_m n \rceil$ and let $i$ be the state corresponding to $\epsilon$. Similar to the prefix case, we first show that there exists a word $w$ of length $r$ such that for every state $q$ every path from $q$ labeled by $w$ contains the state $i$. Suppose that this is not true and hence for every word $w$ of this length there is a state $q_w$ having a path labeled by $w$ which does not contain the state $i$. Denote by $q'_w$ a state in $q_w.w$ such that the path from $q_w$ to $q'_w$ labeled by $w$ does not contain $i$. Observe that for different words $w_1, w_2$ the states $q'_{w_1}$, $q'_{w_2}$ are different, and none of these states is the state $i$. Thus the total number of states in $\mathcal{A}$ is at least $m^r + 1 = m^{\lceil \log_m n \rceil} + 1 > n$, which is a contradiction.

Now we will show that $w$ has rank at most $r$. Denote by $W$ the set of prefixes of $w$. Since for every state $q$ all the paths going from $q$ and labeled by $w$ contain the state $i$, every row of $M(w)$ is a linear combination of vectors $\mathrm{vect}(i.w')$ for $w' \in W$ with coefficients 0 and 1. Hence the rank of $M(w)$ is at most $r$. □

Provided a short word of small rank, we can perform the general algorithm of [62] described in Section 5.2 starting with the pair $w, w$ and thus improve the upper bound. Let $w$ be a word of rank $k$ for a strongly connected unambiguous automaton $\mathcal{A}$. Then the set $\mathcal{P}(w, w)$ contains at most $k^2$ states. Indeed, by definition of rank the matrix $M(w)$ can be represented as $\sum_{i=1}^{k} \mathrm{vect}(C_i)^T \cdot \mathrm{vect}(R_i)$ for some families $C_1, \ldots, C_k$ and $R_1, \ldots, R_k$ of states, where $w$ maps every state of $C_i$ to every state of $R_i$. Every state in $\mathcal{P}(w, w)$ is then contained in some $C_i$ and in some $R_j$. If there are two different states contained in the same pair $C_i$ and $R_j$, the automaton $\mathcal{A}$ is not unambiguous. Hence the cardinality of $\mathcal{P}(w, w)$ is at most $k^2$. If the rank of $ww$ is zero, we have constructed a mortal word for $\mathcal{A}$. Otherwise we can start with the pair $w, w$ and apply the iterative algorithm described in Section 5.2 to get a pair $y', z'$ of words such that no pair of states in $\mathcal{P}(y'w, wz')$ is coreachable or mergeable. Since $\mathcal{P}(w, w)$ contains at most $k^2$ states, it remains to perform at most $k^2$ steps to get rid of all mergeable and coreachable states by $y'$ and $z'$. Then we can kill the states

in $\mathcal{P}(y'w, wz')$ one by one as described in Lemma 15 of [62]. The length of a mortal word thus constructed is at most $k^2n+(k^2+1)(\frac{1}{4}k^2(n+2)^2(n-1)+2|w|)$, see the bound in Lemma 15 of [62].

By Lemma 52 we can find in polynomial time a word $w$ of logarithmic length and logarithmic rank (by enumerating all words of that length). By starting with the pair $w, w$ we get the following.

**Theorem 53.** *Let $\mathcal{A}$ be the prefix automaton of a finite non-complete code $X$ over an alphabet of size $m$. Then there exists a mortal word of length $O(n^3(\log_m n)^4)$, where $n$ is the number of states in $\mathcal{A}$.*

For the case of complete automata we can get a slightly better bound by using the main result of [25], since by Lemma 52 the minimum rank of a word in a complete prefix automaton of a finite code is at most logarithmic.

**Corollary 54.** *Let $\mathcal{A}$ be the prefix automaton of a finite complete code $X$ over an alphabet of size $m$. Then the length of a word of minimum rank for $\mathcal{A}$ is at most $\frac{1}{2}n^3 \cdot \lceil \log_m n \rceil$, where $n$ is the number of states in $\mathcal{A}$.*

Now we proceed to a much stronger bound for the special case of non-complete finite prefix codes. If $k$ is the length of the longest codeword of a finite non-complete prefix code, an upper bound of $2k^2$ on the length of a shortest mortal word can be easily obtained (see, e.g., [79]). We improve this bound.

**Theorem 55.** *Let $X$ be a non-complete finite prefix code, and let $k$ be the length of a longest word in $X$. Then there exists a mortal word of length at most $\frac{3}{2}(k^2+k)$.*

*Proof.* Consider the prefix automaton $\mathcal{A} = (Q, \Sigma, \delta)$ of $X$ (since $X$ is a prefix code, this automaton is deterministic). Observe that $Q' = Q.a^k$ is a subset of $Q$ of size at most $k$. For any word $w \in \Sigma^*$ the set $Q'.w$ does not contain two different states corresponding to prefixes of the same length.

Now we start with the set $Q'$ of active states and consequently perform the following algorithm: while there is at least one active state, take the active state corresponding to the longest prefix and kill it by sending it to a state $q$ such that $q.a$ is empty for some $a \in \Sigma$ and then applying $a$. Since the set of active states consists of states corresponding to prefixes of pairwise different lengths, it takes a word of length at most $(1+k)+(2+k)+\ldots+(k+k) = \frac{k(k+1)}{2}+k^2 = \frac{3k^2+k}{2}$ to map each state first to the state corresponding to the empty prefix and then to kill it. Together with the word $a^k$ we get the required. □

As proved in [86], the code $\Sigma^k \setminus \{u\}$ for any unbordered word $u$ (that is, a word having no prefix equal to its suffix) has the length of a shortest mortal word equal to $k^2+k-1$. We conjecture that the order of this bound is tight, that is, the optimal upper bound is $k^2+O(k)$.

## 5.5 Mortality lower bounds for deterministic automata

In this section we provide lower bounds on the length of shortest mortal words in several classes of deterministic automata. It is known that every $n$-state non-complete deterministic automaton admits a mortal word of length at most $\frac{n(n+1)}{2}$, and the bound is tight [91] (the lower bound is the best known even for unambiguous automata). For strongly connected binary automata the best known lower bound is of order $\frac{n^2}{4}+O(n)$, see, e.g., [70]. We concentrate on several subclasses of strongly connected automata and show similar lower bounds for them. We start with a short proof of a lower bound of Pribavkina for Huffman decoders [86], basing directly on the structure of the automaton. Then we use a similar technique to provide quadratic upper bounds for two other

classes of automata. For all figures of this section dashed lines represent transitions for $a$, dotted lines for $b$, and solid lines for both $a$ and $b$.

Recall that a strongly connected deterministic automaton is called a *Huffman decoder* if it has a state which is contained in every cycle.

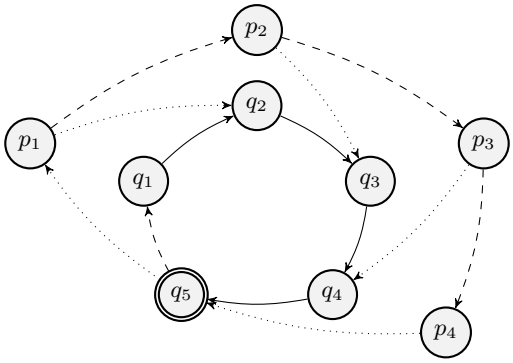


Figure 5.1: A Huffman decoder with 9 states.

**Proposition 56.** *For every odd $n \geq 3$ there exists an $n$-state binary non-complete Huffman decoder $\mathcal{A}$ with the length of a shortest mortal word equal to $\frac{n^2+4n-1}{4}$.*

*Proof.* Consider the following construction of a non-complete deterministic automaton $A = (Q, \{a, b\}, \delta)$. The automaton has a cycle $q_1, \ldots, q_m \in Q$ such that it is possible to leave this cycle only at the state $q_m$. More formally, $\delta(q_i, x) = q_{i+1}$ for $1 \leq i \leq m-1$ and $x \in \{a, b\}$. The state $q_m$ is mapped to $q_1$ by $a$.

Besides the cycle $q_1, \ldots, q_m$, $\mathcal{A}$ has a chain $p_1, \ldots, p_{m-1}$ of states leading to the only state with an undefined transition, which is $p_{m-1}$. More precisely, the letter $a$ moves the states of the chain (except $p_{m-1}$) one step forward: $\delta(p_i, a) = p_{i+1}$ for $1 \leq i \leq m-2$. The letter $b$ returns the states of the chain to the cycle (to the same position as like they did not leave it): $\delta(p_i, b) = q_{i+1}$ for $1 \leq i \leq m-1$. The only undefined transition is by the letter $a$ at the state $p_{m-1}$. Finally, the state $q_m$ is mapped to $p_1$ by $b$, which is the only way to reach the chain $p_1, \ldots, p_{m-1}$ from the cycle $q_1, \ldots, q_m$. See Figure 5.1 for the example with $m = 5$.

In the automaton $\mathcal{A}$ thus constructed all the cycles pass through the state $q_m$, hence it is a Huffman decoder.

Observe that the word $ba^{m-1}(bba^{m-1})^{m-1}$ of length $m^2 + m - 1$ is mortal for $\mathcal{A}$. We are going to show that it is in fact a shortest mortal word.

Indeed, $\mathcal{A}$ has period $m$, which means that $Q$ can be partitioned into $m$ equivalence classes such that no pair of states from different classes can be mapped to the same state. More precisely, the classes are $\{q_i, p_i\}$ for $1 \leq i \leq m-1$ and $\{q_m\}$. The only way to decrease the number of active classes is to map a state of some active class to the state $p_{m-1}$ and then apply $a$. We call a class active if at least one state from it is active. Let us start with the set $\{q_1, \ldots, q_m\}$ of active states, one from each class. To kill a state from this set (by sending it to $p_{m-1}$ first) we have to apply the word $ba^{m-1}$, and during the application of this word no other active state leaves the set $\{q_1, \ldots, q_m\}$. Further, after any application of $ba^{m-1}$ the state $q_m$ cannot be active, thus we have to apply $b$ to make $q_m$ active so that we can kill another active state. We have to perform that for each of $m$ classes, thus the length of a shortest mortal word is at least $m + (m+1)(m-1) = m^2 + m - 1$. □

If we take the state $q_m$ to be the only initial and accepting state, the prefix code decoded by the automaton constructed in the proof is the code $\{a, b\}^m \setminus \{ba^{m-1}\}$. The same construction and

arguments show that for any alphabet $\Sigma$ and an unbordered word $u$ of length $m$ the code $\Sigma^m \setminus \{u\}$ has the same length of a shortest mortal word [86] (recall that a word is called *unbordered* is none of its prefix equals to its suffix).

We proceed with a lower bound for circular automata. Recall that an automaton is called *circular* if it has a letter acting as a cyclic permutation on the whole set of its states. Circular automata play a crucial role in the theory of synchronizing automata since the automata with the longest known length of shortest synchronizing words are circular [113].

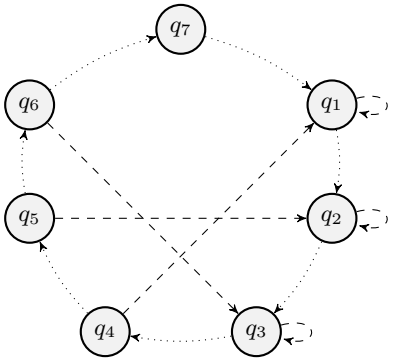


Figure 5.2: A circular automaton with 7 states.

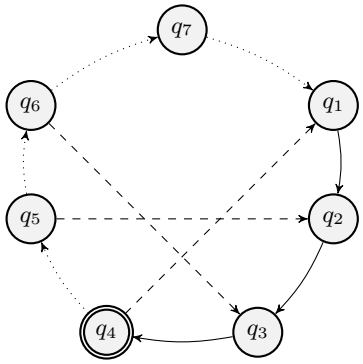


Figure 5.3: A circular Huffman decoder with 7 states.

**Proposition 57.** *For every odd $n \geq 3$ there exists an $n$-state non-complete deterministic circular automaton $\mathcal{A}$ with the length of a shortest mortal word equal to $\frac{n^2+2n+1}{4}$.*

*Proof.* Consider the following construction of a non-complete deterministic automaton $\mathcal{A} = (Q, \{a, b\}, \delta)$. The automaton has a cycle $q_1, \ldots, q_n \in Q$ which is induced by the letter $b$. This means that $\delta(q_i, b) = q_{i+1}$ for $1 \leq i \leq n-1$, and $\delta(q_n, b) = q_1$. The letter $a$ is undefined for $q_n$. Let $n = 2m+1$. For $q_i$, $1 \leq i \leq m$, $a$ acts as a self-loop. Finally, for $q_i$, $m+1 \leq i \leq 2m$, $a$ sends the state $m$ steps back on the cycle: $\delta(q_i, a) = q_{i-m}$. See Figure 5.2 for an example.

Observe that $a(b^{m+1}a)^m$ is a mortal word of length $m^2 + 2m + 1 = \frac{n^2+2n+1}{4}$. We will show that this is in fact a shortest mortal word. Let us start with a set $\{q_n, q_1, \ldots, q_m\}$ of active states, and let $w$ be a shortest word killing all these states. There are two options: either $w$ first maps a pair of active states to the same state, or it first kills one state. In both cases after that the set of active states is exactly $\{q_1, \ldots, q_m\}$. No pair of active states can be mapped to the same state after that, so the only way to decrease the number of active states is to map a state to $q_n$ and then to apply $a$. After each application of $a$ the set of active states is a subset of $\{q_1, \ldots, q_m\}$, and the number of active states is decreased by at most 1. Thus, every mortal word has length at least $1 + (m+2)m$. □

**Proposition 58.** *For every odd $n \geq 3$ there exists an $n$-state non-complete circular Huffman decoder with the length of a shortest mortal word at least $\frac{n^2}{8}$.*

*Proof.* We modify the construction of Proposition 57. Consider the following construction of a non-complete deterministic automaton $\mathcal{A} = (Q, \{a, b\}, \delta)$. The automaton has a cycle $q_1, \ldots, q_n \in Q$ which is induced by the letter $b$. This means that $\delta(q_i, b) = q_{i+1}$ for $1 \leq i \leq n-1$, and $\delta(q_n, b) = q_1$. The letter $a$ is undefined for $q_n$. Let $n = 2m+1$. For $q_i$, $1 \leq i \leq m$, $a$ sends the state $q_i$ one step forward on the cycle: $\delta(q_i, a) = q_{i+1}$. Finally, for $q_i$, $m+1 \leq i \leq 2m$, $a$ sends the state $m$ steps back on the cycle: $\delta(q_i, a) = q_{i-m}$. See Figure 5.3 for an example. The proof of the lower bound is similar to the previous two proofs, so we omit it because of the space constraints. □

It is an interesting question whether there is a connection of the length of shortest mortal words with properties of graphs and matrices, similar to the synchronization case [5].

## 5.6 Codes of full combinatorial rank

In this section we use unambiguous automata to prove a result about synchronizing codes. A *bi-infinite word* over an alphabet $\Sigma$ is a bi-infinite sequence of symbols from $\Sigma$, i.e. a mapping $w : \mathbb{Z} \to \Sigma$. A bi-infinite word $w$ is *periodic* if there exists an integer $k$ such that $w(i) = w(i + k)$ for every $i \in \mathbb{Z}$. A finite word which is not a power of a shorter word is called *primitive*.

**Proposition 59.** *Let $X$ be a recognizable (by a finite automaton) code consisting of at least two words. If $X$ is not synchronizing, then there exists an infinite number of bi-infinite periodic words having two different factorizations over $X$.*

*Proof.* Construct a strongly connected unambiguous automaton $\mathcal{A} = (Q, \Sigma, \Delta)$ such that $X$ is the set of first return words of some of its state $i$ (such an automaton always exists, see Section 4 of [14]). If $\mathcal{A}$ is not synchronizing, the minimum non-zero rank $r$ of words in $\mathcal{A}$ is at least 2. Then there exists an idempotent $e \in \Sigma^*$ of rank $r$ stabilizing $i$ (Theorem 9.3.10 of [14]), that is, $i.e = i$ and for every state $q \in Q$ we have $q.e^2 = q.e$. Consider the set $W$ of all non-mortal words $eue$ with $u \in \Sigma^*$. Let $S$ be the set of fixed points of $e$, that is, states $q$ with $q.e = q$. The set $W$ acts on $S$ as a transitive permutation group of degree $r$, with $r \geq 2$ (Theorem 9.3.10 of [14]). Thus, there exists a word $w = eme \in W$ having $i.w = q$, where $q \in S$, $q \neq i$. Let $\ell$ be the smallest number such that $i.w^\ell = i$. The states $i, i.w, i.w^2, \ldots, i.w^{\ell-1}$ are pairwise different. Thus, the word $w^\ell \in X^*$ is not a power of a shorter word from $X^*$.

Let $x_1, x_2$ be two words in $X$. At least one of them, say $x_1$, does not commute with $w' = eem$, that is, $x_1 w' \neq w' x_1$. Then $x_1$ and $w'$ are not the powers of the same word. We are going to show that the words $w' x_1^k$ are primitive for large enough $k$. Indeed, assume that $u^t = w' x_1^k$ for $t > 1$. If $|x_1^k| \geq |x_1| + |u|$, then by the periodicity lemma [40] we get that $u$ and $x_1$ are powers of the same word, and thus $w'$ is also a power of this word, which is not possible. If $|x_1^k| < |x_1| + |u|$, then $t = 2$ and $|w'| + |x_1| \geq |x_1^{k-1}|$, which is false for $k$ large enough.

Denote $w_k = emx_1^k e$. Since the word $w' x_1^k = eemx_1^k$ is primitive for large enough $k$, so is $w_k$. The word $w_k$ is not mortal for every $k$ since $q'.w_k = i$, where $q' \in S$ is a state such that $q'.w = i$ (the word $x_1$ stabilizes $i$). Moreover, $w_k$ has rank $r$, and we have $q'.w_k = i$, so $i.w_k \neq i$. Thus in the same way as for $w$, we get for every large enough $k$ a word $(w_k)^{\ell_k} \in X^*$ which is not a power of a shorter word in $X^*$. To every word $w_k$ we put in correspondence a bi-infinite word obtained by repeating $w_k$. Since the lengths of the words $w_k$ increase and the words are primitive for large enough $k$, all these bi-infinite words are pairwise different. Each of them has two different factorizations over $X$. □

The converse statement is not true. To see that, it is enough to consider a non-synchronizing code which is contained in a synchronizing code. For example, the code

$$\{aaaa, aabb, bbaa, bbbb\}$$

is contained in the code

$$\{aaaa, aabb, bbaa, bbbb, abab, baba\}.$$

The *combinatorial rank* of a set $X$ of words over an alphabet $\Sigma$ is the minimum cardinality of a set $Y \subseteq \Sigma^*$ such that $X \subseteq Y^*$. That is, it is the minimum cardinality of a set of words such that every word of $X$ can be written as a concatenation of words from $Y$. For example, the

combinatorial rank of every binary code is exactly two. By the result of [58] every finite code $X$ with the combinatorial rank equal to the cardinality of $X$ has only finite number of bi-infinite words with two different factorizations over $X$. By Proposition 59 this means that this code is synchronizing. Thus we get the following.

**Theorem 60.** *Every finite code such that its cardinality equals its combinatorial rank is synchronizing.*

Since every two-word code has combinatorial rank 2, we get in particular that every two-word code is synchronizing. Consider the following examples. For the two-word code $\{x, y\}$ with $x = abab$, $y = baba$ none of the codewords is synchronizing, but $xy$ and $yx$ are. For the two-word code $\{x, y\}$ with $x = b$, $y = abab$ none of the codewords is synchronizing, $xy$ is not synchronizing, but $yx$ is. For the two-word code $\{x, y\}$ with $x = ababa$, $y = bab$ none of the codewords is synchronizing, $xy$ and $yx$ are not synchronizing, but $xx$ and $yy$ are. We conjecture that for every two-word code $X$ there always exists a synchronizing word in $X^2$.

# Computational Complexity and Algorithms

# Chapter 6

# Weakly Acyclic Automata

Since all automata in this chapter are deterministic, we write "automaton" for a "complete DFA", and "partial automaton" for a "partial DFA".

## 6.1 Introduction

In this chapter we provide various results concerning computational complexity and approximability of the problems related to subset synchronization in weakly acyclic automata. Such automata serve as an example of a small class of automata where most of the synchronization problems are still hard. More precisely, switching from general automata to weakly acyclic usually results in changing a PSPACE-complete problem to a NP-complete one.

In Section 6.2 we prove some lower and upper bounds on the length of a shortest word synchronizing a weakly acyclic automaton or, more generally, a subset of its states. We also show that the problem of finding the length of a shortest word synchronizing a subset is hard to approximate. In Section 6.3 we study inapproximability of the problem of finding a subset of states of maximum size. In Section 6.4 we give strong inapproximability results for computing the rank of a subset of states in binary weakly acyclic automata. In Section 6.5 we show that several other problems related to recognizing a synchronizable set in a weakly acyclic automaton are hard.

## 6.2 Bounds on the Length of Shortest Synchronizing Words

Each synchronizing weakly acyclic automaton is a 0-automaton (i.e., an automaton with exactly one sink state), which gives an upper bound $\frac{n(n-1)}{2}$ on the length of a shortest synchronizing word [91]. The same bound can be deduced from the fact that each weakly acyclic automaton is aperiodic [108]. However, for weakly acyclic automata a more accurate result can be obtained, showing that weakly acyclic automata of rank $r$ behave in a way similar to monotonic automata of rank $r$ (see [3]). The following result is a special case of Theorem 2.6 of [2], but here we provide its direct proof.

**Proposition 61.** *Let $\mathcal{A} = (Q, \Sigma, \delta)$ be an $n$-state weakly acyclic automaton, such that there exists a word of rank $r$ with respect to $\mathcal{A}$. Then there exists a word of length at most $n - r$ and rank at most $r$ with respect to $\mathcal{A}$.*

*Proof.* Observe that the rank of a weakly acyclic automaton is equal to the number of sink states in it. The conditions of the theorem imply that $\mathcal{A}$ has at most $r$ sink states.

Consider the sets $S_1, \ldots, S_t$ constructed in the following way. Let $p_i$ be the state in $S_{i-1}$ with the smallest index in the topological sort such that $p_i$ is not a sink state. Let $x_i, 1 \leq i \leq t$, be a letter mapping the state $p_i$ to some other state, where $S_i = \{\delta(q, x_i) \mid q \in S_{i-1}\}, 1 \leq i \leq t$, and $S_0 = Q$. Since $\mathcal{A}$ has at most $r$ sink states, we can thus construct step by step a word $w = x_1 \ldots x_t$ with $t \leq n - r$ that has rank at most $r$ with respect to $\mathcal{A}$. □

The following simple example shows that the bound is tight even for the case of alphabet of size one. Consider an automaton $\mathcal{A} = (Q, \{x\}, \delta)$ with states $q_1, \ldots, q_n$. For the letter $x$ define the transition function $\delta(q_i, x) = q_{i+1}$ for $1 \leq i \leq n - r$ and $\delta(q_i, x) = q_i$ for $n - r + 1 \leq i \leq n$. Obviously, $\mathcal{A}$ has rank $r$ and shortest words of rank $r$ with respect to $\mathcal{A}$ have length $n - r$. For an example with a larger alphabet one can duplicate the letter $x$.

**Proposition 62.** *Let $S$ be a synchronizable set of states of size $k$ in a weakly acyclic $n$-state automaton $\mathcal{A} = (Q, \Sigma, \delta)$. Then the length of a shortest word synchronizing $S$ is at most $\frac{k(2n-k-1)}{2}$.*

*Proof.* Consider a topological sort $q_1, \ldots, q_n$ of the set $Q$. Let $q_s$ be a state such that all states in $S$ can be mapped to it by a shortest word $w = x_1 \ldots x_t$. We can assume that the images of all words $x_1 \ldots x_j$, $j \leq t$, are pairwise distinct, otherwise some letter in this word can be removed. Then a letter $x_j$ maps at least one state of the set $\{\delta(q, x_1 \ldots x_{j-1}) \mid q \in S\}$ to some other state. Thus the maximum total number of letters in $w$ sending all states in $S$ to $q_s$ is at most $(n-k) + (n-k+1) + \ldots + (n-1) = \frac{k(2n-k-1)}{2}$, since application of each letter of $w$ increases the sum of the indices of reached states by at least one. □

Consider a binary automaton $\mathcal{A} = (Q, \{0, 1\}, \delta)$ with $n$ states $q_1, \ldots, q_{k-1}$, $s_1, \ldots, s_\ell$, $t$, where $\ell = n - k$. Define $\delta(q_i, 0) = q_i$, $\delta(q_i, 1) = q_{i+1}$ for $1 \leq i \leq k-2$, $\delta(q_{k-1}, 1) = s_1$. Define also $\delta(s_i, 0) = s_{i+1}$ for $1 \leq i \leq \ell - 1$, $\delta(s_i, 1) = t$ for $1 \leq i \leq \ell - 1$. Define both transitions for $s_\ell$ and $t$ as self-loops. Set $S = \{q_1, \ldots, q_{k-1}, s_\ell\}$. The shortest word synchronizing $S$ is $(10^{l-1})^{k-1}$ of length $(k-1)(n-k)$. The automaton in this example is binary weakly acyclic, and even has rank 2. Figure 6.1 gives the idea of the described construction.

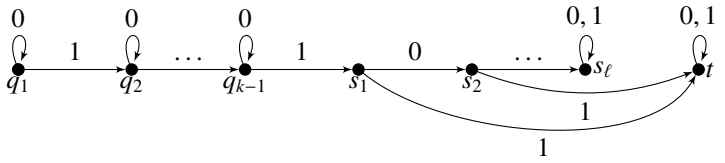


Figure 6.1: The automaton providing the lower bound for subset synchronization

As was noted by an anonymous reviewer of the paper [92], for alphabet of size $n-2$, a better lower bound of $\frac{(k-1)(2n-k-2)}{2}$ can be shown as follows. Let $Q = \{-1, 0, 1, \ldots, n-2\}$, $\Sigma = \{a_1, \ldots, a_{n-2}\}$,

$$\delta(k, a_i) = \begin{cases} k & \text{if } k > i, \\ k-1 & \text{if } k = i, \\ -1 & \text{if } 0 < k < i, \\ k & \text{if } k \in \{-1, 0\} \end{cases}$$

If $k < n$ and $S = \{0, n-2, n-3, \ldots, n-k\}$, then it is easy to see that the shortest word synchronizing $S$ has length $(n-k) + (n-k+1) + \cdots + (n-2) = \frac{(k-1)(2n-k-2)}{2}$. For each $n$ and $k$, this is less than the upper bound of Proposition 62 by $n-1$ only.

Consider now the following problem, which is a generalization of the Short Sync Word problem.

SHORTEST SET SYNC WORD
*Input*: An automaton $\mathcal{A}$ and a synchronizable subset $S$ of its states;
*Output*: The length of a shortest word synchronizing $S$.

It follows from Proposition 62 that the decision version of this problem (asking whether there exists a word of length at most $k$ synchronizing $S$) is in NP for weakly acyclic automata, so it is reasonable to investigate its approximability.

**Theorem 63.** *The* SHORTEST SET SYNC WORD *problem for $n$-state binary weakly acyclic automata cannot be approximated in polynomial time within a factor of $O(n^{\frac{1}{2}-\epsilon})$ for any $\epsilon > 0$ unless $P = NP$.*

*Proof.* To prove this theorem, we construct a gap-preserving reduction from the SHORTEST SYNC WORD problem in $p$-state binary automata, which cannot be approximated in polynomial time within a factor of $O(p^{1-\epsilon})$ for any $\epsilon > 0$ unless P = NP [43]. Let a binary automaton $\mathcal{A} = (Q, \{0,1\}, \delta)$ be the input of SHORTEST SYNC WORD. Let $Q = \{q_1, \ldots, q_p\}$. Construct a binary automaton $\mathcal{A}' = (Q', \{0,1\}, \delta')$ with the set of states $Q' = \{q_i^{(j)} \mid 1 \leq i \leq p, 1 \leq j \leq p+1\}$. Define $\delta'(q_i^{(j)}, x) = q_k^{(j+1)}$ for $1 \leq i \leq p$, $1 \leq j \leq p$, $x \in \{0,1\}$, where $k$ is such that $q_k = \delta(q_i, x)$. Define $\delta'(q_i^{(p+1)}, x) = q_i^{(p+1)}$ for $1 \leq i \leq p$ and $x \in \{0,1\}$. Take $S' = \{q_i^{(1)} \mid 1 \leq i \leq p\}$.

Observe that any word synchronizing $S'$ in $\mathcal{A}'$ is a synchronizing word for $\mathcal{A}$ because of the definition of $\delta'$. In the other direction, we note that a shortest synchronizing word for a $p$-state automaton in the construction of Gawrychowski and Straszak [43] has length at most $p$. Hence, a shortest synchronizing word for $\mathcal{A}$ also synchronizes $S'$ in $\mathcal{A}'$. Thus, the length of a shortest synchronizing word for $\mathcal{A}$ is equal to the length of a shortest word synchronizing $S'$ in $\mathcal{A}'$, and we get a gap-preserving reduction with gap $O(p^{1-\epsilon}) = O(n^{\frac{1}{2}-\epsilon})$, as $\mathcal{A}'$ has $O(p^2)$ states. Finally, it is easy to see that $\mathcal{A}'$ is binary weakly acyclic. □

## 6.3 Finding a Synchronizable Set of Maximum Size

One possible approach to measure and reduce initial state uncertainty in an automaton is to find a subset of states of maximum size where the uncertainty can be resolved, i.e., to find a synchronizable set of maximum size. This is captured by the following problem.

MAX SYNC SET
*Input*: An automaton $\mathcal{A}$;
*Output*: The maximum cardinality of a synchronizable set of states in $\mathcal{A}$.

Türker and Yenigün [111] study a variation of this problem, which is to find a set of states of maximum size that can be mapped by some word to a subset of a given set of states in a given monotonic automaton. They reduce the N-QUEENS PUZZLE problem [9] to this problem to prove its NP-hardness. However, their proof is unclear, since the input has size $O(\log N)$, and the output size is polynomial in $N$. Also, the N-QUEENS PUZZLE problem is solvable in polynomial time [9].

First we investigate the PSPACE-completeness of the decision version of the MAX SYNC SET problem, which we will denote as MAX SYNC SET-D. Its formulation is the following: given an automaton $\mathcal{A}$ and a number $c$, decide whether there is a synchronizable set of states of cardinality at least $c$ in $\mathcal{A}$.

**Proposition 64.** *The* MAX SYNC SET-D *problem is PSPACE-complete for binary automata.*

*Proof.* The SYNC SET problem is in PSPACE [100]. Thus, the MAX SYNC SET-D problem is also in PSPACE, as we can sequentially check whether each subset of states is synchronizable and compare the size of a maximum synchronizable set to $c$.

To prove that the Max Sync Set-D problem is PSPACE-hard for binary automata, we will reduce a PSPACE-complete Sync Set problem for binary automata [114] to it. Let an automaton $\mathcal{A}$ and a subset $S$ of its states be an input to Sync Set. Let $n$ be the number of states of $\mathcal{A}$. Construct a new automaton $\mathcal{A}'$ by initially taking a copy of $\mathcal{A}$. For each state $s \in S$, add $n+1$ *new* states to $\mathcal{A}'$ and define all the transitions from these new states to map to $s$, regardless of the input letter. Define the set $S'$ to be a union of all new states and take $c = |S'| = (n+1)|S|$.

Let $S_1$ be a maximum synchronizable set in $\mathcal{A}$ not containing at least one new state $q$. As $S_1$ is maximum, it does not contain other $n$ new states that can be mapped to the same state as $q$. Thus, the size of $S_1$ is at most $n+(n+1)|S|-(n+1) < (n+1)|S| = c$. Hence, each synchronizable set of size at least $c$ in $\mathcal{A}'$ contains $S'$. The set $S$ is synchronizable in $\mathcal{A}$ if and only if $S'$ is synchronizable in $\mathcal{A}'$, as each word $w$ synchronizing $S$ in $\mathcal{A}$ corresponds to a word $xw$ synchronizing $S'$ in $\mathcal{A}'$, where $x$ is an arbitrary letter. Thus, $\mathcal{A}'$ has a synchronizable set of size at least $c$ if and only if $S$ is synchronizable in $\mathcal{A}$. □

Now we proceed to inapproximability results for the Max Sync Set problem in several classes of automata. We will need some results from graph theory. An *independent set* $I$ in a graph $G$ is a set of its vertices such that no two vertices in $I$ share an edge. The size of a maximum independent set in $G$ is denoted $\alpha(G)$. The Independent Set problem is defined as follows.

Independent Set
*Input*: A graph $G$;
*Output*: An independent set of maximum size in $G$.

Zuckerman [121] has proved that, unless P = NP, there is no polynomial $p^{1-\varepsilon}$-approximation algorithm for the Independent Set problem for any $\varepsilon > 0$, where $p$ is the number of vertices in $G$.

**Proposition 65.** *The problem* Max Sync Set *for weakly acyclic $n$-state automata over an alphabet of cardinality $O(n)$ cannot be approximated in polynomial time within a factor of $O(n^{1-\varepsilon})$ for any $\varepsilon > 0$ unless $P = NP$.*

*Proof.* We will prove this theorem by constructing a gap-preserving reduction from the Independent Set problem. Given a graph $G = (V, E)$, $V = \{v_1, v_2, \ldots, v_p\}$, we construct an automaton $\mathcal{A} = (Q, \Sigma, \delta)$ as follows. For each $v_i \in V$, we add two states $s_i, t_i$ in $Q$. We also add a state $f$ to $Q$. Thus, $|Q| = 2p+1$. The alphabet $\Sigma$ consists of letters $\tilde{v}_1, \ldots, \tilde{v}_p$ corresponding to the vertices of $G$.

The transition function $\delta$ is defined in the following way. For each $1 \leq i \leq p$, the state $s_i$ is mapped to $f$ by the letter $\tilde{v}_i$. For each $v_i v_j \in E$ the state $s_i$ is mapped to $t_i$ by the letter $\tilde{v}_j$, and the state $s_j$ is mapped to $t_j$ by the letter $\tilde{v}_i$. All yet undefined transitions map a state to itself.

**Lemma 66.** *Let $I$ be a maximum independent set in $G$. Then the set $S = \{s_i \mid v_i \in I\} \cup \{f\}$ is a synchronizable set of maximum cardinality (of size $\alpha(G)+1$) in the automaton $\mathcal{A} = (Q, \Sigma, \delta)$.*

*Proof.* Let $w$ be a word obtained by concatenating the letters corresponding to $I$ in arbitrary order. Then $w$ synchronizes the set $S = \{s_i \mid v_i \in I\} \cup \{f\}$ of states of cardinality $|I|+1$. Thus, $\mathcal{A}$ has a synchronizable set of size at least $\alpha(G)+1$.

In other direction, let $w$ be a word synchronizing a set of states $S'$ of maximum size in $\mathcal{A}$. We can assume that after reading $w$ all the states in $S'$ are mapped to $f$, as all the sets of states that are mapped to any other state have cardinality at most two. Then there are no edges in $G$ between any pair of vertices in $I' = \{v_i \mid s_i \in S'\}$, since each time some state $s_i$ is mapped to $f$, each state $s_j$ such that $v_j$ and $v_i$ share an edge in $G$, is mapped to $t_j$. Hence $I'$ is an independent set of size $|S'|-1$ in $G$. Thus the maximum size of a synchronizable set in $\mathcal{A}$ is equal to $\alpha(G)+1$. □

Thus we have a gap-preserving reduction from the Independent Set problem to the Max Sync Set problem with a gap $\Theta(p^{1-\varepsilon})$ for any $\varepsilon > 0$. It is easy to see that $n = \Theta(p)$ and $\mathcal{A}$ is weakly acyclic, which concludes the proof of the theorem. □

Next we move to a weaker inapproximability result for binary automata.

**Proposition 67.** *The problem* Max Sync Set *for binary $n$-state automata cannot be approximated in polynomial time within a factor of $O(n^{\frac{1}{2}-\varepsilon})$ for any $\varepsilon > 0$ unless $P = NP$.*

*Proof.* Again, we construct a gap-preserving reduction from the Independent Set problem extending the proof of Proposition 65. Given a graph $G = (V, E), V = \{v_1, v_2, \ldots, v_p\}$, we construct an automaton $\mathcal{A} = (Q, \Sigma, \delta)$ in the following way. Let $\Sigma = \{0, 1\}$. First we construct the main gadget $\mathcal{A}_{main}$ having a synchronizable set of states of size $\alpha(G)$. For each vertex $v_i \in V, 1 \le i \le p$, we construct a set of new states $L_i = V_i \cup U_i$ in $Q$, where $V_i = \{v_j^{(i)} : 1 \le j \le p\}, U_i = \{u_j^{(i)} : 1 \le j \le p\}$. We call $L_i$ the $i$th *layer* of $\mathcal{A}_{main}$. We also add a state $f$ to $Q$. For each $i$, $1 \le i \le p$, the transition function $\delta$ imitates taking the vertices $v_1, v_2, \ldots, v_p$ into an independent set one by one and is defined as:

$$\delta(v_j^{(i)}, 0) = \begin{cases} u_j^{(i)} & \text{if } i = j, \\ v_j^{(i+1)} & \text{otherwise} \end{cases}$$

$$\delta(v_j^{(i)}, 1) = \begin{cases} u_j^{(i)} & \text{if there is an edge } v_i v_j \in E, \\ v_j^{(i+1)} & \text{otherwise} \end{cases}$$

Here all $v_j^{(n+1)}, 1 \le j \le p$, coincide with $f$. For each state $u_j^{(i)}$, the transitions for both letters 0 and 1 lead to the originating state (i.e. they are self-loops).

We also add a $p$-state cycle $\mathcal{A}_{cycle}$ attached to $f$. It is a set of $p$ states $c_1, \ldots, c_p$, mapping $c_i$ to $c_{i+1}$ and $c_p$ to $c_1$ regardless of the input symbol. Finally, we set $c_1$ to coincide with $f$. Thus we get the automaton $\mathcal{A}_1$. Figure 6.2 presents an example of $\mathcal{A}_1$ for a graph with three vertices $v_1, v_2, v_3$ and one edge $v_2 v_3$.

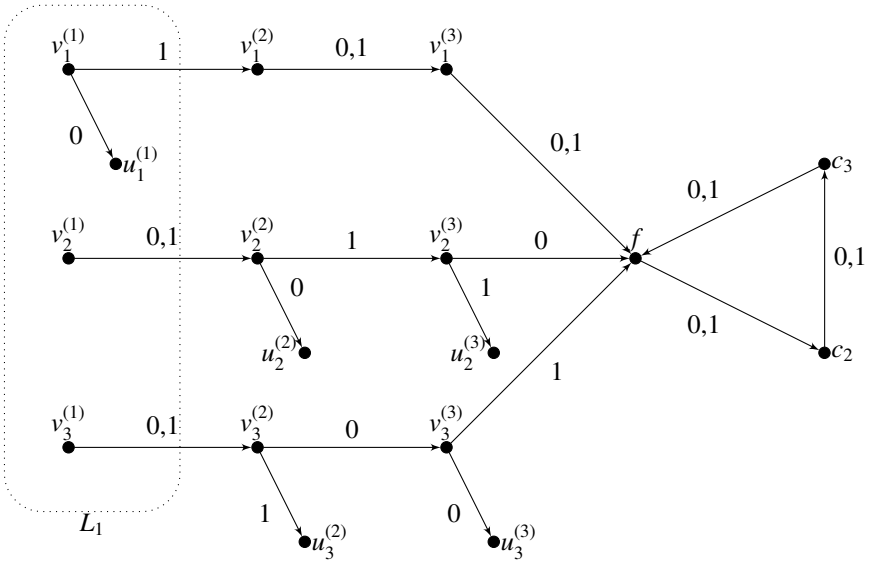


Figure 6.2: An example of $\mathcal{A}_1$. Unachievable states and self-loops are omitted.

The main property of $\mathcal{A}_1$ is claimed by the following lemma.

**Lemma 68.** *The size of a maximum synchronizable set of states from the first layer in $\mathcal{A}_1$ equals $\alpha(G)$.*

*Proof.* Let $I$ be a maximum independent set in $G$. Consider a word $w$ of length $p$ such that its $i$th letter is equal to 0 if $v_i \notin I$ and to 1 if $v_i \in I$. By the construction of $\mathcal{A}_1$, this word synchronizes the set $\{v_j^{(1)} \mid v_j \in I\}$.

Conversely, a synchronizable set of at least three states from the first layer can be mapped only to some vertex of $\mathcal{A}_{cycle}$, and the corresponding set of vertices in $G$ is an independent set, since application of each letter corresponds to taking or not taking a vertex into an independent set, and after taking some vertex all the states corresponding to the neighbors of this vertex are sent to separate sink states. □

Some layer in the described construction can contain a synchronizable subset of size larger than the maximum synchronizable subset of the first layer. To avoid that, we modify $\mathcal{A}_1$ by repeating each state (with all transitions) of the first layer $p$ times. More formally, we replace each pair of states $v_j^{(1)}$, $u_j^{(1)}$ with $p$ different pairs of states such that in each pair all the transitions repeat the transitions between $v_j^{(1)}$, $u_j^{(1)}$, and all the other states of the automaton. We denote the automaton thus constructed as $\mathcal{A}$.

The following lemma claims that the described procedure of constructing $\mathcal{A}$ from $G$ is a gap-preserving reduction from the Independent Set problem in graphs to the Max Sync Set problem in binary automata.

**Lemma 69.** *If $\alpha(G) > 1$, then the maximum size of a synchronizable set in $\mathcal{A}$ is equal to $p\alpha(G) + 1$.*

*Proof.* Note that since the length of the cycle in $\mathcal{A}_{cycle}$ equals $p$, each synchronizable set of $\mathcal{A}$ is either a subset of a single layer of $\mathcal{A}$ together with a state in $\mathcal{A}_{cycle}$ (if this set is first mapped to $f$) or a subset of a set $\{v_j^{(i)} \mid 2 \leq i \leq \ell\} \cup \{u_j^{(\ell)}\}$ for some $\ell$ and $j$, together with $p$ new states that replaced $v_j^{(1)}$ (if this set is mapped to $u_j^{(\ell)}$). Consider the first case. If some maximum synchronizable set $S$ contains a state from the $i$th layer of $\mathcal{A}$ and $i > 1$, then its size is at most $p + 1$. A maximum synchronizable set containing some states from the first layer of $\mathcal{A}$ consists of $p\alpha(G)$ states from this layer (according to Lemma 68) and some state of $\mathcal{A}_{cycle}$, so this set has size $p\alpha(G) + 1 \geq 2p + 1$. In the second case, the maximum size of a synchronizable set is at most $p + (p - 1) + 1 = 2p < p\alpha(G) + 1$. □

It is easy to see that the constructed reduction is gap-preserving with a gap $\Theta(p^{1-\varepsilon}) = \Theta(n^{\frac{1}{2}-\varepsilon})$, where $n$ is the number of states in $\mathcal{A}$, as $n = \Theta(p^2)$. Thus the Max Sync Set for $n$-state binary automata cannot be approximated in polynomial time within a factor of $O(n^{\frac{1}{2}-\varepsilon})$ for any $\varepsilon > 0$ unless P = NP, which concludes the proof of the theorem. □

Proposition 67 can also be proved by using Proposition 65 and a slight modification of the technique used in [114] for decreasing the size of the alphabet. However, in this case the resulting automaton is far from being weakly acyclic, while the automaton in the proof of Proposition 65 has only one cycle. The next theorem shows how to modify our technique to prove an inapproximability bound for Max Sync Set in binary weakly acyclic automata.

**Theorem 70.** *The* Max Sync Set *problem for binary weakly acyclic $n$-state automata cannot be approximated in polynomial time within a factor of $O(n^{\frac{1}{3}-\varepsilon})$ for any $\varepsilon > 0$ unless $P = NP$.*

*Proof.* We modify the construction of the automaton $\mathcal{A}_{main}$ from Proposition 67 in the following way. We repeat each state (with all transitions) of the first layer $p^2$ times in the same way as it is done in the proof of Proposition 67. Thus we get a weakly acyclic automaton $\mathcal{A}_{wa}$ with $n = \Theta(p^3)$ states, where $p$ is the number of vertices in the graph $G$. Furthermore, similar to Lemma 69, the size of the maximum synchronizable set of states in $\mathcal{A}_{wa}$ is between $p^2\alpha(G)$ and $p^2\alpha(G) + p(p-1) + 1$, because some of the states from the layers other than the first can be also mapped to $f$. Both of the values are of order $\Theta(p^2\alpha(G))$, thus we have an gap-preserving reduction providing the inapproximability within a factor of $O(p^{1-\varepsilon}) = O(n^{\frac{1}{3}-\varepsilon})$ for any $\varepsilon > 0$, where $n$ is the number of states in $\mathcal{A}_{wa}$. □

We finish by noting that for two classes of automata the Max Sync Set problem is solvable in polynomial time.

**Proposition 71.** *The problem* Max Sync Set *can be solved in polynomial time for unary automata.*

*Proof.* Consider the digraph $G$ induced by states and transitions of an unary automaton $\mathcal{A}$. By definition, each vertex of $G$ has outdegree 1. Thus, the set of the vertices of $G$ can be partitioned into directed cycles and a set of vertices not belonging to any cycle, but lying on a directed path leading to some cycle. Let $n$ be the number of states in $\mathcal{A}$. It is easy to see that after performing $n$ transitions, each state of $\mathcal{A}$ is mapped into a state in some cycle, and all further transitions will not map any two different states to the same state. Thus, it is enough to perform $n$ transitions and select such state $s$ that the maximum number of states are mapped to $s$. □

A more interesting case is covered by the following proposition. Recall that an automaton $\mathcal{A} = (Q, \Sigma, \delta)$ is called *Eulerian* if there exists $k$ such that for each state $q \in Q$ there are exactly $k$ pairs $(q', a)$, $q' \in Q$, $a \in \Sigma$, such that $\delta(q', a) = q$.

**Proposition 72.** *The problem* Max Sync Set *can be solved in polynomial time for Eulerian automata.*

*Proof.* According to Theorem 2.1 in [42] (see also [60] for the discussion of the Eulerian case), each word of minimum rank with respect to an Eulerian automaton synchronizes the sets $S_1, S_2, \ldots, S_r$ forming a partition of the states of the automaton into inclusion-maximal synchronizable sets. Moreover, according to this theorem all inclusion-maximal synchronizable sets in an Eulerian automaton are of the same size, thus each inclusion-maximal synchronizable set has maximum cardinality. A word of minimum rank with respect to an automaton can be found in polynomial time [90], which concludes the proof. □

## 6.4 Computing the Rank of a Subset of States

Assume that we know that the current state of the automaton $\mathcal{A}$ belongs to a subset $S$ of its states. Even if it is not possible to synchronize $S$, it can be reasonable to minimize the size of the set of possible states of $\mathcal{A}$, reducing the uncertainty of the current state as much as possible. One way to do it is to map $S$ to a set $S'$ of smaller size by applying some word to $\mathcal{A}$. Recall that the size of the smallest such set $S'$ is called the rank of $S$. Consider the following problem of finding the rank of a subset of states in a given automaton.

Set Rank
*Input*: An automaton $\mathcal{A}$ and a set $S$ of its states;
*Output*: The rank of $S$ in $\mathcal{A}$.

The rank of an automaton, that is, the rank of the set of its states, can be computed in polynomial time [90]. However, since the automaton in the proof of PSPACE-completeness of Sync Set in [89] has rank 2 (and thus each subset of states in this automaton has rank either 1 or 2), it follows immediately that there is no polynomial $c$-approximation algorithm for the Set Rank problem for any $c < 2$ unless P = PSPACE. It also follows that checking whether the rank of a subset of states equals the rank of the whole automaton is PSPACE-complete.

We will need the Chromatic Number problem. A *proper coloring* of a graph $G = (V, E)$ is a coloring of the set $V$ in such a way that no two adjacent vertices have the same color. The chromatic number of $G$, denoted $\chi(G)$, is the minimum number of colors in a proper coloring of $G$. Recall that a set of vertices in a graph is called *independent* if no two vertices in this set are adjacent. A proper coloring of a graph can be also considered as a partition of the set of its vertices into independent sets.

Chromatic Number
*Input*: A graph $G$;
*Output*: The chromatic number of $G$.

This problem cannot be approximated within a factor of $O(p^{1-\epsilon})$ for any $\epsilon > 0$ unless P = NP, where $p$ is the number of vertices in $G$ [121].

**Proposition 73.** *The* Set Rank *problem for $n$-state weakly acyclic automata with alphabet of size $O(\sqrt{n})$ cannot be approximated within a factor of $O(n^{\frac{1}{2}-\epsilon})$ for any $\epsilon > 0$ unless $P = NP$.*

*Proof.* We will prove this theorem by constructing a gap-preserving reduction from the Chromatic Number problem, extending the technique in the proof of Proposition 65. Given a graph $G = (V, E)$, $V = \{v_1, v_2, \ldots, v_p\}$, we construct an automaton $\mathcal{A} = (Q, \Sigma, \delta)$ as follows. The alphabet $\Sigma$ consists of letters $\tilde{v}_1, \ldots, \tilde{v}_p$ corresponding to the vertices of $G$, together with a *switching* letter $\nu$. We use $p$ identical *synchronizing* gadgets $T^{(k)}, 1 \leq k \leq p$, such that each gadget synchronizes a subset of states corresponding to an independent set in $G$. Gadget $T^{(k)}$ consists of a set $\{s_i^{(k)}, t_i^{(k)} \mid 1 \leq i \leq p\} \cup \{f^{(k)}\}$ of states.

The transition function $\delta$ is defined as follows. For each gadget $T^{(k)}$, for each $1 \leq i \leq p$, the state $s_i^{(k)}$ is mapped to $f^{(k)}$ by the letter $\tilde{v}_i$. For each $v_iv_j \in E$ the state $s_i^{(k)}$ is mapped to $t_i^{(k)}$ by the letter $\tilde{v}_j$, and the state $s_j^{(k)}$ is mapped to $t_j^{(k)}$ by the letter $\tilde{v}_i$. All yet undefined transitions corresponding to the letters $\tilde{v}_1, \ldots, \tilde{v}_p$ map a state to itself.

It remains to define the transitions corresponding to $\nu$. For each $1 \leq k \leq p-1$, $\nu$ maps $t_i^{(k)}$ and $s_i^{(k)}$ to $s_i^{(k+1)}$, and $f^{(k)}$ to itself. Finally, $\nu$ acts on all states in $T^{(p)}$ as a self-loop.

Define $S = \{s_i^{(1)} \mid 1 \leq i \leq p\}$. We will prove that the rank of $S$ is equal to the chromatic number of $G$. Consider a proper coloring of $G$ with the minimum number of colors and let $I_1 \cup \ldots \cup I_{\chi(G)}$ be the partition of $G$ into independent sets defined by this coloring. For each $I_j$, consider a word $w_j$ obtained by concatenating the letters corresponding to the vertices in $I_j$ in some order. Consider now the word $w_1\nu w_2\nu \ldots \nu w_{\chi(G)}$. This word maps the set $S$ to the set $\{f^{(i)} \mid 1 \leq i \leq \chi(G)\}$, which proves that the rank of $S$ is at most $\chi(G)$.

In the other direction, note that after each reading of $\nu$ all states except $f^{(k)}, 1 \leq k \leq p-1$, are mapped to the next synchronizing gadget (except the last gadget $T^{(p)}$ which is mapped to itself). Note that each time a state $s_i^{(k)}$ is mapped to $f^{(k)}$, each state $s_j^{(k)}$ such that $v_j$ and $v_i$ share an edge in $G$, is mapped to $t_j^{(k)}$. Thus, only a subset of states corresponding to an independent set of vertices can be mapped to some particular $f^{(k)}$, and the image of $S$ after reading any word is a subset of the states in some gadget together with some of the states $f^{(k)}, 1 \leq k \leq p$. Hence, the rank of $S$ is at least $\chi(G)$.

Thus we have a gap-preserving reduction from the Chromatic Number problem to the Set Rank problem with gap $\Theta(p^{1-\varepsilon})$ for any $\varepsilon > 0$. It is easy to see that $n = \Theta(p^2)$, $\mathcal{A}$ is weakly acyclic and its alphabet has size $O(\sqrt{n})$, which finishes the proof of the theorem. □

Using the classical technique of reducing the alphabet size (see [114]), $O(n^{\frac{1}{3}-\epsilon})$ inapproximability can be proved for binary automata. To prove the same bound for binary weakly acyclic automata, we have to refine the technique of the proof of the previous theorem.

**Theorem 74.** *The* Set Rank *problem for $n$-state binary weakly acyclic automata cannot be approximated within a factor of $O(n^{\frac{1}{3}-\epsilon})$ for any $\epsilon > 0$ unless $P = NP$.*

*Proof.* To prove this theorem we construct a gap-preserving reduction from the Chromatic Number problem, extending the proof of the previous theorem.

Given a graph $G = (V, E), V = \{v_1, v_2, \ldots, v_p\}$, we construct an automaton $\mathcal{A} = (Q, \{0, 1\}, \delta)$. In our reduction we use two kinds of gadgets: $p$ *synchronizing gadgets* $T^{(k)}$, $1 \leq k \leq p$, and $p$ *waiting gadgets* $R^{(k)}$, $1 \leq k \leq p$. Gadget $T^{(k)}$ consists of a set $\{v_{i,j}^{(k)} \mid 1 \leq i, j \leq p\}$ of states, together with a state $f^{(k)}$, and $R^{(k)}$, $1 \leq k \leq p$, consists of the set $\{u_{i,j}^{(k)} \mid 1 \leq i, j \leq p\}$.

For each $i, j, k$, $1 \leq i, j, k \leq p$, the transition function $\delta$ is defined as:

$$\delta(v_{i,j}^{(k)}, 0) = \begin{cases} u_{i,j}^{(k)} & \text{if } i = j, \\ v_{i+1,j}^{(k)} & \text{otherwise} \end{cases}$$

$$\delta(v_{i,j}^{(k)}, 1) = \begin{cases} u_{i,j}^{(k)} & \text{if there is an edge } v_i v_j \in E, \\ v_{i+1,j}^{(k)} & \text{otherwise} \end{cases}$$

Here all $v_{p+1,j}^{(k)}, 1 \leq j \leq p$, coincide with $f^{(k)}$. We set $\delta(u_{i,j}^{(k)}, x) = u_{i+1,j}^{(k)}$ for $x \in \{0, 1\}$, $1 \leq i, k \leq p-1$, $1 \leq j \leq p$, and $\delta(u_{p,j}^{(k)}, x) = v_{1,j}^{(k+1)}$ for $1 \leq j \leq p$, $1 \leq k \leq p-1$, $x \in \{0, 1\}$. The states $u_{i,j}^{(p)}$ are sink states: both letters 0 and 1 act on them as self-loops. Finally, we set $S = \{v_{1,j}^{(1)} \mid 1 \leq j \leq p\}$. Figure 6.3 gives an idea of the described construction.

The idea of the presented construction is essentially a combination of the ideas in the proofs of Proposition 67 and Proposition 73, so we provide only a sketch of the proof. A synchronizing gadget $T^{(k)}$ synchronizes a set $S^{(k)} \subseteq S$ of states corresponding to some independent set in $G$. All the states corresponding to the vertices adjacent to vertices corresponding to $S^{(k)}$ are mapped to the corresponding waiting gadget $R^{(k)}$, and get to the next synchronizing gadget $T^{(k+1)}$ only after the states of $S^{(k)}$ are synchronized (and thus mapped to $f^{(k)}$). Hence, the minimum size of a partition of $V$ into independent sets is equal to the rank of $S$. The number of states in $\mathcal{A}$ is $O(p^3)$. Thus, we get $\Theta(n^{\frac{1}{3}-\epsilon})$ inapproximability. □

A natural open problem is to study the Max Sync Set and Set Rank problems complexity in strongly connected automata. The technique used by Vorel for proving PSPACE-completeness of the Sync Set problem in strongly connected automata [114] seems to fail here.

## 6.5 Subset Synchronization

In this section, we obtain complexity results for several problems related to subset synchronization in weakly acyclic automata. We adapt Eppstein's construction from [37], which is a powerful and flexible tool for such proofs. We will need the following NP-complete SAT problem [104].

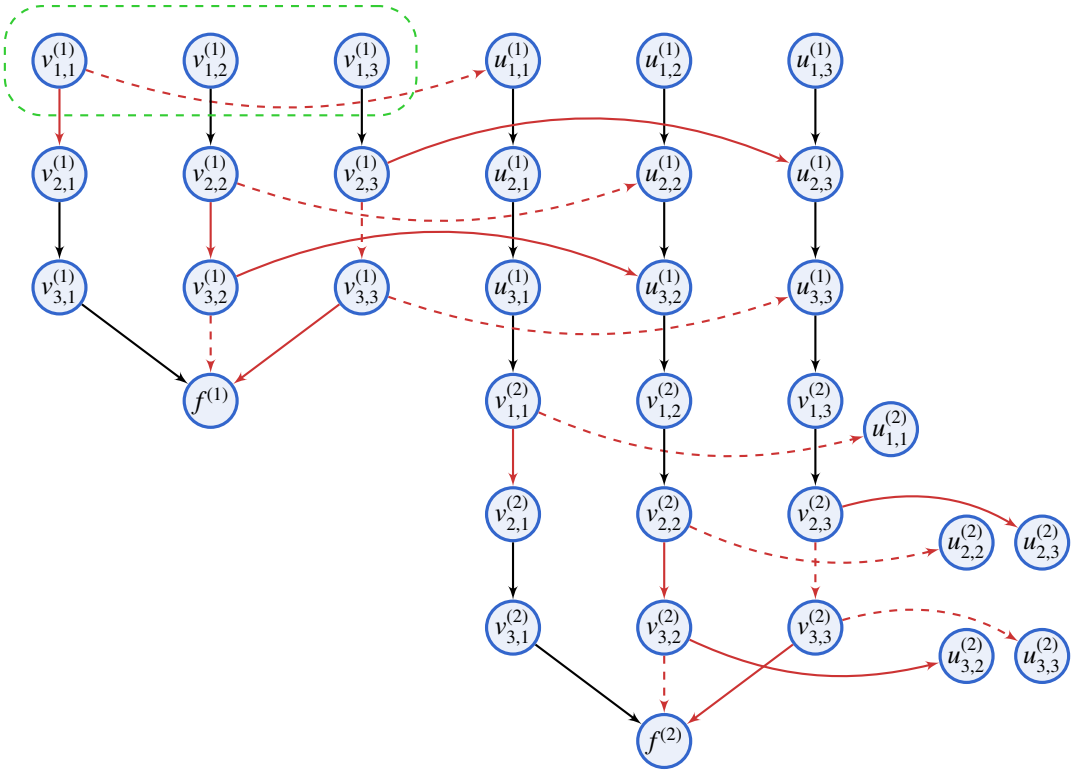


Figure 6.3: A part of the construction in the reduction for Set Rank. Red dashed arrows represent transitions for letter 0, red solid arrows – for letter 1, black arrows – for both letters. Self-loops are omitted.

SAT
*Input*: A set $X$ of $n$ boolean variables and a set $C$ of $m$ clauses;
*Output*: Yes if there exists an assignment of values to the variables in $X$ such that all clauses in $C$ are satisfied, No otherwise.

**Theorem 75.** *The* Sync Set *problem in binary weakly acyclic automata is NP-complete.*

*Proof.* Because of the polynomial upper bound on the length of a shortest word synchronizing a subset of states proved in Proposition 62, we can use such word as a certificate. Thus, the problem is in NP.

We reduce the SAT problem. Given $X$ and $C$, we construct an automaton $\mathcal{A} = (Q, \{0,1\}, \delta)$. For each clause $c_j$, we add $n+1$ states $y_i^{(j)}, 1 \leq i \leq n+1$, to $Q$. We introduce also a state $f \in Q$. The transitions from $y_i^{(j)}$ correspond to the occurrence of $x_i$ in $c_j$ in the following way: for $1 \leq i \leq n$, $1 \leq j \leq m$, $\delta(y_i^{(j)}, a) = f$ if the assignment $x_i = a$, $a \in \{0,1\}$, satisfies $c_j$, and $\delta(y_i^{(j)}, a) = y_{i+1}^{(j)}$ otherwise. The transition function $\delta$ also maps $y_{n+1}^{(j)}$ to itself for all $1 \leq j \leq m$ and both letters 0 and 1.

Let $S = \{y_1^{(j)} \mid 1 \leq j \leq m\}$. The word $w = a_1 a_2 \dots a_n$ synchronizes $S$ if $a_i$ is the value of $x_i$ in an assignment satisfying $C$, and vice versa. Thus, the set is synchronizable if and only if all clauses in $C$ can be satisfied by some assignment of binary values to the variables in $X$. □

By identifying the states $y_{n+1}^{(j)}$ for $1 \leq j \leq m$ and adding $f$ to $S$ it is also possible to prove that the problem of checking whether the rank of a subset of states equals the rank of an automaton

is coNP-complete for binary weakly acyclic automata (cf. the remarks in the beginning of Section 6.4).

The proof of Theorem 75 can be used to prove the hardness of a special case of the following problem, which is PSPACE-complete [66].

FINITE AUTOMATA INTERSECTION
*Input*: Automata $\mathcal{A}_1, \ldots, \mathcal{A}_k$ (with initial and accepting states);
*Output*: Yes if there is a word which is accepted by all automata, No otherwise.

**Proposition 76.** FINITE AUTOMATA INTERSECTION *is NP-complete when all automata in the input are binary weakly acyclic.*

*Proof.* Observe first that if there exists a word which is accepted by all automata, then a shortest such word $w$ has length at most linear in the total number of states in all automata. Indeed, for each automaton consider a topological sort of the set of its states. Each letter of $w$ maps at least one state in some automaton to some other state, which has larger index in the topological sort of the set of states of this automaton. Thus, the considered problem is in NP.

For the hardness proof, we use the same construction as in Theorem 75. Provided $X$ and $C$, define $\mathcal{A}$ in the same way as in Theorem 75. Define $\mathcal{A}_j = (Q_j, \{0,1\}, \delta_j)$ as follows. Take $Q_j = \{y_i^{(j)}, 1 \leq i \leq n+1\} \cup \{f\}$ and $\delta_j$ to be the restriction of $\delta$ to the set $Q_j$. Set $y_1^{(j)}$ to be the input state and $f$ to be the only accepting state of $\mathcal{A}_j$. Then there exists a word accepted by automata $\mathcal{A}_1, \ldots, \mathcal{A}_m$ if and only if all clauses in $C$ are satisfiable by some assignment. □

To obtain the next results, we will need a modified construction of the automaton from the proof of Theorem 75, as well as some new definitions. Given an instance of the SAT problem, construct a partial automaton $\mathcal{A}_{base} = (Q, \{0,1\}, \delta)$ as follows. We introduce a state $f \in Q$. For each clause $c_j$, we add $n+1$ states $y_i^{(j)}, 1 \leq i \leq n+1$, to $Q$. For each $c_j$, add also states $z_i^{(j)}$ for $h_i + 1 \leq i \leq n+1$, where $h_i$ is the smallest index of a variable occurring in $c_j$. The transitions from $y_i^{(j)}$ correspond to the occurrence of $x_i$ in $c_j$ in the following way: for $1 \leq i \leq n$, $\delta(y_i^{(j)}, a) = z_{i+1}^{(j)}$ if the assignment $x_i = a$, $a \in \{0,1\}$, satisfies $c_j$, and $\delta(y_i^{(j)}, a) = y_{i+1}^{(j)}$ otherwise. For $a \in \{0,1\}$, we set $\delta(z_i^{(j)}, a) = z_{i+1}^{(j)}$ for $h_i + 1 \leq i \leq n$, $1 \leq j \leq m$. The transition function $\delta$ also maps $z_{n+1}^{(j)}$, $1 \leq j \leq m$, and $f$ to $f$ for both letters 0 and 1. We use $\mathcal{A}_{base}$ to prove the hardness of the CAREFUL SYNCHRONIZATION problem.

We call a partial automaton *aperiodic* if for any word $w \in \Sigma^*$ and any state $q \in Q$ there exists $k$ such that either $\delta(q, w^k)$ is undefined, or $\delta(q, w^k) = \delta(q, w^{k+1})$.

**Proposition 77.** CAREFUL SYNCHRONIZATION *is NP-hard for aperiodic partial automata over a three-letter alphabet.*

*Proof.* We reduce the SAT problem. Given $X$ and $C$, we first construct $\mathcal{A}_{base}$. Then we add an additional letter $r$ to the alphabet of $\mathcal{A}_{base}$ and introduce $m$ new states $s^{(j)}$, $1 \leq j \leq m$. For $1 \leq i \leq n$, $1 \leq j \leq m$, we define $\delta(s^{(j)}, r) = y_1^{(j)}$, $\delta(y_i^{(j)}, r) = y_1^{(j)}$, $\delta(z_i^{(j)}, r) = y_1^{(j)}$, $\delta(f, r) = f$. All other transitions are left undefined. Let us call the constructed automaton $\mathcal{A}$.

The automaton $\mathcal{A}$ is carefully synchronizing if and only if all clauses in $C$ can be satisfied by some assignment of binary values to the variables in $X$. Moreover, the word $w = rw_1w_2 \ldots w_n0$, is carefully synchronizing if $w_i$ is the value of $x_i$ in such an assignment.

Indeed, note that the first letter of $w$ is necessarily $r$, as it is the only letter defined for all the states. Moreover, each word starting with $r$ maps $Q$ to a subset of $\{y_i^{(j)}, z_i^{(j)} \mid 1 \leq j \leq m+1\} \cup \{f\}$. The only way for a word to map all states to $f$ is to map them first to the set $\{z_{n+1}^{(j)} \mid 1 \leq j \leq$

$m\} \cup \{f\}$, because there are no transitions defined from any $y_{n+1}^{(j)}$, except the transitions defined by $r$. If only the states of the set $\{z_{n+1}^{(j)} \mid 1 \leq j \leq m\} \cup \{f\}$ are reached by a word $ra_1 \dots a_n$, then for every clause in $C$ there exists a variable $x_i$ such that taking $x_i = a_i$ satisfies this clause. Thus there exists an assignment satisfying $C$.

The constructed automaton is aperiodic, because each cycle which is not a self-loop contains exactly one letter $r$. □

The complexity of the following problem can be obtained from Proposition 77.

Positive Matrix
*Input*: A set $M_1, \dots, M_k$ of $n \times n$ binary matrices;
*Output*: Yes if there exists a sequence $M_{i_1} \times \dots \times M_{i_k}$ of multiplications (possibly with repetitions) providing a matrix with all elements equal to 1, No otherwise.

**Corollary 78.** Positive Matrix *is NP-hard for two upper-triangular and two lower-triangular matrices.*

*Proof.* The proof uses the idea from [45]. Consider three transition matrices corresponding to the letters of the automaton constructed in the proof of Proposition 77. Add the matrix corresponding to the letter mapping the state $f$ to all states and undefined for all other states. Any sequence of matrices resulting in a matrix with only positive elements must contain the new matrix, and before that there must be a sequence of matrices corresponding to a word carefully synchronizing the automaton from the proof of Proposition 77. Thus we get a reduction from Careful Synchronization for aperiodic partial automata over a three-letter alphabet to Positive Matrix. It is easy to see that the reduction uses two upper-triangular and two lower-triangular matrices. □

Finally, we show the hardness of the following problem (PSPACE-complete in general [19]).

Subset Reachability
*Input*: An automaton $\mathcal{A} = (Q, \Sigma, \delta)$ and a subset $S$ of its states;
*Output*: Yes if there exists a word $w$ such that $\{\delta(q, w) \mid q \in Q\} = S$, No otherwise.

**Theorem 79.** Subset Reachability *is NP-complete for weakly acyclic automata.*

*Proof.* Consider a topological sort of $Q$. Let $w$ be a shortest word mapping $Q$ to some reachable set of states. Then each letter of $w$ maps at least one state to a state with a larger index in the topological sort. Thus $w$ has length $O(|Q|^2)$, since the maximum total number of such mappings is $(|Q| - 1) + (|Q| - 2) + \dots + 1 + 0$. Thus, the considered problem is in NP.

For the NP-hardness proof, we again reduce the SAT problem. Given an instance of SAT, construct $\mathcal{A}_{base}$ first. Next, add a transition $\delta(y_{n+1}^{(j)}, a) = f$ for $1 \leq j \leq m$, $a \in \{0, 1\}$, resulting in a deterministic automaton $\mathcal{A}$.

Similar to the proof of Proposition 77, $C$ is satisfiable if and only if the set $S = \{z_j^{(n+1)} \mid 1 \leq j \leq m\} \cup \{f\}$ is reachable in $\mathcal{A}$. Indeed, consider the word $w = a_1 a_2 \dots a_n$ where $a_i$ is the value of $x_i$ in an assignment satisfying $C$. Then the image of $Q$ under the mapping defined by $w$ equals $S$. On the other hand, any word $w$ with $\{\delta(q, w) \mid q \in Q\} = S$ has length exactly $n$, since the application of each new letter increases by one the minimal index $i$ such that a state $y_i^{(j)}$ or $z_i^{(j)}$ is reached. Let $w = a_1 a_2 \dots a_n$. If the assignment $x_i = a_i$ does not satisfy a clause $c_j$, then $\{\delta(q, w) \mid q \in Q\}$ contains $y_{n+1}^{(j)}$, and thus cannot be equal to $S$. □

# Chapter 7

# Monotonic Automata

Since all automata in this chapter are deterministic, we write "automaton" for a "complete DFA", and "partial automaton" for a "partial DFA".

## 7.1 Introduction

Monotonic automata provide form a special class of automata with a lot of additional structure. A number of NP-complete and PSPACE-complete synchronization problems, such as finding a shortest synchronizing word or subset synchronization are solvable in polynomial time for such automata. Moreover, this class admits upper bounds that are better than in the general case.

In this chapter, we study both extremal and algorithmic questions of subset synchronization in monotonic automata. In Section 7.2, we provide structural results about synchronizable sets of states in monotonic automata and give algorithmic consequences of this results. In Section 7.3, we provide lower and upper bounds on the maximum length of shortest words synchronizing a subset of states in monotonic automata, and show some lower bounds for related problems. In Section 7.4, we provide NP-hardness and inapproximability of several problems related to subset synchronization and careful synchronization of monotonic automata.

## 7.2 Structure of Synchronizing Sets

Let $\mathcal{A}$ be an automaton, and $S$ be a subset of its states. In general, if any two states in $S$ can be synchronized (i.e., form a synchronizable set), $S$ does not necessarily have to be synchronizable, as it is shown by the following theorem.

**Theorem 80.** *For any positive integer $k_0$, there exists a binary weakly acyclic automaton $\mathcal{A}$ and a subset $S$ of its states such that $|S| \geq k_0$, each pair of states in $S$ in synchronizable, but the rank of $S$ equals $|S| - 1$.*

*Proof.* Consider the following automaton $\mathcal{A} = (Q, \{0, 1\}, \delta)$. Let $S = \{s_0, \ldots, s_{k-1}\}$. Let $k = 2^\ell$ for an integer number $\ell$, and let $\mathrm{bin}(i)$ be a word which is equal to the binary representation of $i$ of length $\ell$ (possibly with zeros at the beginning). We introduce new states $t_i, p_i$ for $0 \leq i \leq k-1$, a state $f$, and new intermediate states in $Q$ as follows. For each $s_i$, $0 \leq i \leq k-1$, consider a construction sending $s_i$ to $f$ for a word $\mathrm{bin}(i)$, and to $t_i$ by any other word of length $\ell$.

For each $t_i$, consider the same construction sending $t_i$ to $f$ for a word $\mathrm{bin}(i)$, and to $p_i$ otherwise. For each $i$, define both transitions from $p_i$ as self-loops. Define both transitions from $f$ as self-loops.

In this construction, each word applied after a word of length $2\ell$ obviously has no effect. Consider a word $w$ of length $2\ell$, $w = w_1w_2$, where both $w_1$ and $w_2$ have length $\ell$. If $w_1 = w_2$, then the image of $S$ under the mapping defined by $w$ has size $k$. Otherwise, $w$ synchronizes two states $s_i$ and $s_j$ with $\mathrm{bin}(i) = w_1$ and $\mathrm{bin}(j) = w_2$ and maps all other states to different states. Thus, the rank of $S$ equals $k - 1$. □

The size of the whole automaton is $O(|S| \log |S|)$, thus $S$ can be large comparing to the size of the whole set of states in the automaton.

Since the Sync Set problem is PSPACE-complete in strongly connected automata, pairwise synchronization of states in a subset does not imply that this subset is synchronizable for this class of automata unless P = PSPACE. Thus, it is reasonable to ask the following question.

**Question 81.** *How large can be the rank of a subset of states in a strongly connected automaton such that each pair of states in this subset can be synchronized?*

For the rest of section, fix a monotonic automaton $\mathcal{A} = (Q, \Sigma, \delta)$ and an order $q_1, \ldots, q_n$ of its states preserved by all transitions. As shown by the next theorem, the situation in monotonic automata is in some sense opposite to the situation described by Theorem 80.

**Theorem 82.** *Let $S \subseteq Q$ be a subset of states of $\mathcal{A}$. Then $S$ is synchronizable if and only if any two states in $S$ can be synchronized.*

*Proof.* Obviously, any subset of a synchronizable set is synchronizable.

In the other direction, if any two states in $S$ can be synchronized, then the minimal state $q_\ell = \min S$ and the maximal state $q_r = \max S$ in $S$ can be synchronized by a word $w \in \Sigma^*$. Let $q = \delta(q_\ell, w) = \delta(q_r, w)$. Then the interval $[q_\ell, q_r] = \{q_\ell, \ldots, q_r\}$ is synchronized by $w$, because each state of $[q_\ell, q_r]$ is mapped to the interval $[\delta(q_\ell, w), \delta(q_r, w)] = \{q\}$, since $\mathcal{A}$ is monotonic. Thus, $S \subseteq [q_\ell, q_r]$ is synchronizable. □

**Corollary 83.** *The problem of checking whether a given set $S$ is synchronizable (the* Sync Set *problem) can be solved in $O(|Q|^2 \cdot |\Sigma|)$ time and space for monotonic automata.*

*Proof.* By Theorem 82 it is enough to check that each pair of states in $S$ can be synchronized, which can be done by solving the reachability problem in the subautomaton of the power automaton, built on all 2-element and 1-element subsets of $Q$ [113]. There are $\frac{|Q|(|Q|+1)}{2}$ states in this subautomaton $\mathcal{A}^2$. We need to check that from each state $\{q_i, q_j\}$, $q_i, q_j \in S$, in $\mathcal{A}^2$ some singleton set is reachable. Consider the underlying digraph of $\mathcal{A}^2$ and reverse all arcs in it. Then we need to check that in this new digraph each vertex $\{q_i, q_j\}$, $q_i, q_j \in S$, is reachable from some singleton. To check it, run breadth-first search simultaneously from all singletons [31].

To construct the subautomaton we need $O(|Q|^2 \cdot |\Sigma|)$ time and space, and breadth-first search requires time and space linear in the number of arcs of the digraph. □

**Corollary 84.** *A shortest word synchronizing a given subset $S$ of states can be found in $O(|Q|^4 \cdot |\Sigma|)$ time and $O(|Q|^2 \cdot |\Sigma|)$ space for monotonic automata.*

*Proof.* Consider the following algorithm. For each pair of states, find a shortest word synchronizing this pair. This can be done by solving the shortest path problem in the subautomaton of the power automaton, build on all 2-element and 1-element subsets of $Q$ [113]. Let $W$ be the set of all such words that synchronize $S$. Output the shortest word in $W$.

By an argument similar to the proof of Theorem 82, any shortest word synchronizing $\{\min S, \max S\}$ is a shortest word synchronizing $S$, thus the algorithm finds a shortest word synchronizing $S$. Since

finding a shortest synchronizing word for a pair of states requires $O(|Q|^2 \cdot |\Sigma|)$ time and there are $O(|Q|^2)$ such words, the set $W$ can be found in $O(|Q|^4 \cdot |\Sigma|)$ time. Finding a shortest word in $W$ synchronizing $S$ requires additional $O(|Q|^4)$ time, since we have to check each word. Thus, the total time required by the algorithm is $O(|Q|^4 \cdot |\Sigma|)$.

The algorithm requires $O(|Q|^2 \cdot |\Sigma|)$ space for the subautomaton construction. We don't have to store $W$, since we can check the words in $W$ one by one and store only the shortest one, so we need $O(|Q|^2 \cdot |\Sigma|)$ space. □

The same problem (denoted Shortest Set Sync Word) for weakly acyclic automata is discussed in Section 6.2.

**Corollary 85.** *A synchronizable subset of states of maximum size can be found in $O(|Q|^4 \cdot |\Sigma| + |Q|^5)$ time and $O(|Q|^2 \cdot |\Sigma|)$ space in monotonic automata.*

*Proof.* For each synchronizable pair $q_i, q_j$ of states, find a word synchronizing this pair (in the same way as described in Corollary 83), and then find all states that are mapped by this word to the same state as $q_i$ and $q_j$. Output the pair with the largest synchronizable set constructed in such a way.

To prove that this algorithm is correct, observe that each word synchronizing a pair $q_i, q_j$ of states synchronizes also all the states $q_k$, $q_i < q_k < q_j$. On the other hand, if $q_i = \min S$ and $q_j = \max S$ for a synchronizable set $S$ of maximum size (which is an interval), any word synchronizing $q_i$ and $q_j$ synchronizes only $S$.

The described algorithm requires $O(|Q|^2 \cdot (|Q|^2 \cdot |\Sigma| + |Q|^3))$ (for each pair of states, we need to find a synchronizing word which requires $O(|Q|^2 \cdot |\Sigma|)$ time, and then apply this word to each state, which requires $O(|Q|^3)$) time. Since we need to store only the set of maximum size, the algorithm requires $O(|Q|^2 \cdot |\Sigma|)$ space. □

The problem of finding a synchronizable subset of states of maximum size in the general case (the Max Sync Set) is discussed in Section 6.3.

The algorithms proposed in this section run polynomial time, but the degrees of these polynomials are quite high. A natural question is to find faster algorithms for the described problems. Another interesting question is whether the results can be generalized to orientable automata.

## 7.3 Lower Bounds for Synchronizing Words

The length of a shortest word synchronizing an $n$-state monotonic automaton is at most $n-1$ [3]. In this section we investigate a more general question of bounding the length of a shortest word synchronizing a subset of states in an $n$-state automaton. For a more general class of orientable automata a bound of $(n-1)^2$ is known [37], but for monotonic automata a smaller upper bound can be proved.

**Theorem 86.** *Let $S$ be a synchronizable set of states in a monotonic $n$-state automaton $\mathcal{A}$. Then for $n \geq 8$ the length of a shortest word synchronizing $S$ is at most $\frac{(n-2)^2}{4}$.*

*Proof.* Let $\mathcal{A} = (Q, \Sigma, \delta)$, and $\{q_1, \ldots, q_n\}$ be an order of the states preserved by all transitions of $\mathcal{A}$. Define $q_\ell = \min S$, $q_r = \max S$. We can assume that $S$ can be mapped only to the states in $[q_\ell, q_r]$. Indeed, assume without loss of generality that $q_i$, $i < \ell$, is a state such that $S$ can be mapped to $q_i$, and it is the smallest such state. Then by monotonicity there exists a word mapping $q_r$ to $q_i$ by taking only transitions going to smaller states. This word then synchronizes $S$ and has length at most $n - 1 \leq \frac{(n-2)^2}{4}$ for $n \geq 8$.

Now we can assume that $S$ can be mapped only to states in $[q_\ell, q_r]$. This means that $S$ can be mapped to the states $q_i, q_j$ in $[q_\ell, q_r]$, and no state outside $[q_i, q_j]$ is reachable from any state of $[q_i, q_j]$. Indeed, the set of states reachable from both $q_\ell, q_r$ contains $q_i$ and $q_j$, and if some state outside $[q_i, q_j]$ is reachable from $[q_i, q_j]$ we can synchronize $S$ to a state outside $[q_i, q_j]$, which contradicts the definition of the interval $[q_i, q_j]$. If both $q_\ell$ and $q_r$ are mapped to states inside $[q_i, q_j]$, they can be then synchronized by applying a word of length at most $j - i$, for example by applying a word $w'$ composed of only letters mapping the consecutive images of $q_j$ to states with smaller indexes by the same reasoning as below. By our assumptions, $q_i$ is reachable from each state in $[q_i, q_j]$, thus such a word $w'$ exists.

Suppose now that $w = w_1 \dots w_m$ is a shortest word synchronizing $S$. Consider the sequence of pairs $(t_k, s_k) = (\delta(q_\ell, w_1 \dots w_k), \delta(q_r, w_1 \dots w_k))$, $k = 1, 2, \dots, m$. As $w$ is a shortest word synchronizing $S$, and synchronization of $S$ is equivalent to synchronization of $\{q_\ell, q_r\}$, no pair appears in this sequence twice, and the only pair with equal components is $(s_m, t_m)$. Because of monotonicity, $t_k \leq s_k$ for each $1 \leq k \leq m$. Thus, the maximum length of $w$ is reached when $q_i = q_j$, since after both images are in $[q_i, q_j]$ the remaining length of a synchronizing word is at most $j - i$. Observe that if $q_1$ or $q_n$ is in $S$, $S$ again can be synchronized by a word of length $n - 1$. Thus, we can assume that $|S| \leq n - 2$, and thus the length of $w$ is at most $(i - 1)(n - 3 - i) \leq \frac{(n-2)^2}{4}$. □

The bound is almost tight for monotonic automata over a three-letter alphabet as shown by the following example.

**Theorem 87.** *For each $m \geq 1$, there exists a $(2m+3)$-state monotonic automaton $\mathcal{A}$ over a three-letter alphabet, which has a subset $S$ of states, such that the length of a shortest word synchronizing $S$ is $m^2 + m$.*

*Proof.* Consider the following monotonic automaton $\mathcal{A} = (Q, \Sigma, \delta)$, $Q = \{q_1, \dots, q_{2m+3}\}$. Let $\Sigma = \{0, 1, 2\}$. Let states $q_1$, $q_{m+2}$ and $q_{2m+3}$ be sink states. For every state $q_i$, $2 \leq i \leq m + 1$, we set $\delta(q_i, 0) = q_{i+1}$, $\delta(q_i, 1) = q_i$, $\delta(q_i, 2) = q_1$. For every state $q_i$, $m + 4 \leq i \leq 2m + 2$, we set $\delta(q_i, 0) = q_{2m+3}$, $\delta(q_i, 1) = q_{i-1}$, $\delta(q_i, 2) = q_i$. Finally we define $\delta(q_{m+3}, 0) = q_{2m+2}$, $\delta(q_{m+3}, 1) = q_{m+3}$, $\delta(q_{m+3}, 2) = q_{m+2}$. See Figure 7.1 for an illustration of the construction.

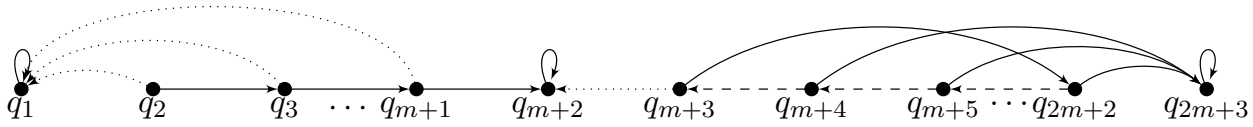


Figure 7.1: The automaton providing a lower bound for subset synchronization in monotonic automata over a three-letter alphabet. Solid arrows represent transitions for the letter 0, dashed – for the letter 1, dotted – for the letter 2. The states $q_1, q_{m+2}, q_{2m+3}$ are sink states, self-loops are omitted.

All transitions of $\mathcal{A}$ respect the order $q_1, \dots, q_{2m+3}$, so $\mathcal{A}$ is monotonic. Let us show that the shortest word synchronizing the set $S = \{q_2, q_{2m+2}\}$ is $w = 1^{m-1}(01^{m-1})^m 2$. Let $S'$ be a set of states such that $q_i, q_j \in S'$, $2 \leq i \leq m + 1$, $m + 3 \leq j \leq 2m + 2$. The set $S'$ can be mapped only to $q_{m+2}$, because $\mathcal{A}$ is monotonic. Hence if any state of $S'$ is mapped by a word to $q_1$ or to $q_{2m+3}$, then this word cannot synchronize $S'$.

We start with the set $S = \{q_2, q_{2m+2}\}$. There is only one letter 1 that does not map the state $q_2$ to $q_1$ or the state $q_{2m+2}$ to $q_{2m+3}$, and maps $S$ not to itself. Indeed, 0 maps $q_{2m+2}$ to $q_{2m+3}$ and 2 maps $q_2$ to $q_1$. Thus, any shortest synchronizing word can start only with 1. Consider now the set

$\{\delta(q_2,1),\delta(q_{2m+2},1)\}=\{q_2,q_{2m+1}\}$. There is only letter 1 that does not map the state $q_3$ to $q_1$, or $q_{2m+1}$ to $q_{2m+3}$ and maps this set not to itself. Indeed, 0 maps $q_{2m+1}$ to $q_{2m+3}$ and 2 maps $q_3$ to $q_1$. So the second letter of the shortest synchronizing word can only be 1. By a similar reasoning (at each step there is exactly one letter that maps a pair of states not to itself and does not map the states to the sink states $q_1$ and $q_{2m+3}$), we deduce that any shortest synchronizing word has to begin with $1^{m-1}(01^{m-1})^m$ and it is easy too see that $1^{m-1}(01^{m-1})^m2$ synchronizes $S$. Thus, $w$ is a shortest word synchronizing $S$, and its length is $m^2+m$. □

For an $n$-state automaton, the lower bound on the length of a shortest word in this theorem is $\frac{(n-2)^2-1}{4}$, which is very close to the lower bound $\frac{(n-2)^2}{4}$ from Theorem 86.

By taking $q_2$ and $q_{2m+2}$ as initial states in two equal copies of the automaton in the proof of Theorem 87, and taking $q_{m+2}$ as the only accepting state in both copies, we obtain the following result.

**Corollary 88.** *A shortest word accepted by two $(2m+3)$-state monotonic automata which differ only by their initial states can have length $m^2+m$.*

For binary monotonic automata, our lower bound is slightly smaller, but still quadratic.

**Theorem 89.** *For each $m\geq 1$, there exists a $(4m+3)$-state binary monotonic automaton $\mathcal{A}$, which has a subset $S$ of states such that the length of a shortest word synchronizing $S$ is at least $m^2$.*

*Proof.* Consider the following automaton $\mathcal{A}=(Q,\Sigma,\delta)$ with $Q=\{q_1,\ldots,q_{4m+3}\}$, $\Sigma=\{0,1\}$. Define $\delta$ as follows. Set $q_1,q_{2m+2},q_{4m+3}$ to be sink states. Define $\delta(q_i,1)=q_{i-1}$ for all $i\neq 1,2m+2,4m+3$. For each $i$, $2\leq i\leq m+1$, define $\delta(q_i,0)=q_{i+m}$, and for each $i$, $m+2\leq i\leq 2m+1$, define $\delta(q_i,0)=q_{2m+2}$. For each $i$, $2m+3\leq i\leq 3m+3$, define $\delta(q_i,0)=q_{m+i-1}$, and for each $i$, $3m+4\leq i\leq 4m+2$, define $\delta(q_i,0)=q_{4m+3}$. The defined binary automaton is monotonic, since all its transitions respect the order $q_1,\ldots,q_{4m+3}$. See Figure 7.2 for an example of the construction.

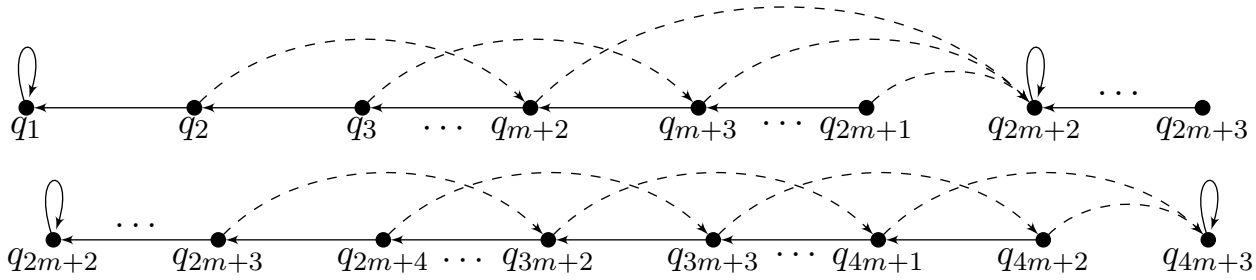


Figure 7.2: The automaton providing a lower bound for subset synchronization in binary monotonic automata. Dashed arrows represent transitions for the letter 0, solid – for the letter 1. The states $q_1,q_{2m+2},q_{4m+3}$ are sink states. The picture is divided into two parts because of its width.

Define $S=\{q_{m+2},q_{4m+2}\}$. Let us prove that a shortest word synchronizing $S$ has length at least $m^2$.

The set $S$ can only be mapped to $q_{2m+2}$, since it is a sink state between $\min S$ and $\max S$. Thus, no word synchronizing $S$ maps any its state to $q_1$ or $q_{4m+3}$. Consider now an interval $[q_i,q_j]$ for $2\leq i\leq m+1$, $2m+3\leq j\leq 4m+2$ and note that applying 0 reduces its length by 1 (or maps its right end to $q_{4m+3}$), and applying 1 maps its ends to the ends of another interval of this form with the same length (or maps its left end to $q_1$). The maximal length of a segment of this form that

allows its left end to be mapped to $q_{2m+2}$ is $2m+1$, so before any end of the interval is mapped to $q_{2m+2}$, the letter 0 has to be applied at least $m$ times. Each application of 0 moves the right end of the intervals $m-1$ states to the right, so each application of 0 requires $m-1$ applications of 1 so that 0 can be applied one more time. Thus, the word mapping $S$ to $q_{2m+2}$ has length at least $m^2$. Note that $S$ can be synchronized by a word $w = (1^{m-1}0)^m 1^{2m}$ of length $|w| = m^2 + 2m$. □

For an $n$-state binary monotonic automaton we get a lower bound of $\frac{(n-3)^2}{16}$ from this theorem.

By removing the first and the last state (and leaving all transitions to the removed states undefined) in the automata in the both series, we get the following results.

**Corollary 90.** *For infinitely many $n$, there exists an $n$-state monotonic partial automaton over a three-letter alphabet with shortest carefully synchronizing word of length at least $\frac{(n-1)^2}{4} + 1$.*

**Corollary 91.** *For infinitely many $n$, there exists an $n$-state monotonic binary partial automaton with shortest carefully synchronizing word of length at least $\frac{(n-1)^2}{16}$.*

**Question 92.** *Find upper bounds on the length of a shortest carefully synchronizing words for monotonic partial automata.*

A related question is to measure the shortest length of a word accepted simultaneously by $k$ monotonic automata. This question is important, for example, for investigation of the computational complexity of the Finite Automata Intersection problem, see Section 7.4 for the details. For alphabet of unbounded size, a partial answer is provided by the following theorem.

**Theorem 93.** *For any $k > 0$, there exist $k$ 3-state monotonic automata over a $k$-letter alphabet, such that the length of a shortest word accepted by all automata is at least $2^k - 1$.*

*Proof.* To prove the theorem, we show how to imitate a simple binary counter with $k$ monotonic automata. Consider the following family $\mathcal{A}_i = (Q_i, \Sigma, \delta_i)$, $1 \leq i \leq k$. Here $Q_i = \{f_i, s_i, t_i\}$, $\Sigma = \{a_1, \ldots, a_k\}$ and $\delta_i$ are defined as follows. We define $\delta_i(s_i, a_i) = t_i$, and $\delta_i(s_i, a_j) = f_i$, $\delta_i(t_i, a_j) = s_i$ for $i < j$. All yet undefined transitions are self-loops. In each $\mathcal{A}_i$, we take $s_i$ as the initial state, and $t_i$ as the only accepting state.

By induction, the length of a shortest word accepted by all automata is $2^k - 1$, since each next automaton can be mapped to its accepting state only when all previous automata are already in their accepting states, and this operation maps all previous automata to their initial states. □

Hence, we get a lower bound of $2^{\frac{n}{3}} - 1$, where $n$ is the total number of states in the automata. Thus the most obvious candidate for a certificate to show that Finite Automata Intersection is in NP for monotonic automata fails.

The proof of Theorem 93 implies the following interesting result.

**Corollary 94.** *For each $k$ there exists a monotonic partial automaton with $3k$ states, alphabet of size $k+1$ and a shortest carefully synchronizing word of length at least $3^k$.*

*Proof.* Remove the states $f_i$ in all automata from the construction of Theorem 93 and leave all transitions to this states undefined. Then add a letter $a$ such that $\delta(t_i, a) = t_k$ for $1 \leq i \leq k$. Thus we get a carefully synchronizing automaton imitating a binary counter. The last step is to note that $2^{\frac{1}{2}} < 3^{\frac{1}{3}}$, and to imitate a ternary counter in the exactly same way instead. □

We note that the ternary counter is optimal since $2^{\frac{1}{2}} < 3^{\frac{1}{3}}$ and $3^{\frac{1}{3}} > \ell^{\frac{1}{\ell}}$ for $\ell \geq 4$.

The bound in this corollary is of the same order as in the result of Rystsov [88] and Martyugin [71], but our example is monotonic and has simpler structure. Since each carefully synchronizing partial automaton has a carefully synchronizing word of length $O(3^{\frac{n}{3}})$ [88], the bound is asymptotically tight.

Some additional optimization can be done for the described construction. In particular, after counting to $3^k - 1$, only two states in the set of reached states can be synchronized by a new letter. Then counting is repeated (to $3^{k-1} - 1$) and again two states are synchronized, and so on. Thus, the bound will become $3^k + 3^{k-1} + \ldots + 3 + 1$. However, the bound remains of the same order and is still smaller than the bounds of Rystsov and Martyugin for general automata.

For a constant number of letters the considered problems remain open.

**Question 95.** *What is the length of a shortest word accepted simultaneously by $k$ monotonic automata over an alphabet of constant size? What is the length of a shortest word carefully synchronizing a monotonic partial automaton with $n$ states and alphabet of constant size?*

## 7.4 Complexity Results

In this section, we obtain computational complexity results for several problems related to subset synchronization in monotonic automata. We improve Eppstein's construction [37] to make it suitable for monotonic automata. We shall need the following NP-complete SAT problem [104].

SAT
*Input*: A set $X$ of $n$ Boolean variables and a set $C$ of $m$ clauses;
*Output*: Yes if there exists an assignment of values to the variables in $X$ such that all clauses in $C$ are satisfied, No otherwise.

Provided a set $X$ of Boolean variables $x_1, \ldots, x_n$ and a clause $c_j$, construct the following automaton $\mathcal{A}_j = (Q, \Sigma, \delta)$. Take

$$Q = \{q_1, \ldots, q_{n+1}\} \cup \{q'_2, \ldots, q'_n\} \cup \{s, t\}.$$

Let $\Sigma = \{0, 1, r\}$. Define the transition function $\delta$ as follows. For each $i$, $1 \leq i \leq n$, map a state $q_i$ to $q'_{i+1}$ (or to $t$ if $i = n$) by a letter $x \in \{0, 1\}$ if the assignment $x_i = x$ satisfies $c_j$, and to $q_{i+1}$ otherwise. For each $i$, $2 \leq i \leq n-1$, set $\delta(q'_i, x) = \delta(q'_{i+1}, x)$ for $x \in \{0, 1\}$. Set $\delta(q'_n, x) = t$ for $x \in \{0, 1\}$. Define transitions from $t$ for letters $0, 1$ as self-loops. Finally, define $\delta(q, r) = s$ for $q \in Q \setminus \{t\}$, $\delta(t, r) = t$. See Figure 7.3 for an example.

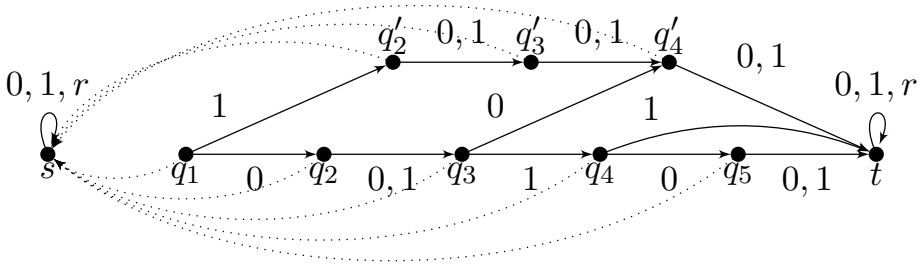


Figure 7.3: The automaton $\mathcal{A}_j$ for a clause $c_j = (x_1 \vee \overline{x}_3 \vee x_4)$. Dotted arrows represent transitions for the letter $r$.

Note that $\mathcal{A}_j$ is monotonic, since it respects the order

$$s, q_1, q_2, q_2', q_3, q_3', \ldots, q_n, q_n', q_{n+1}, t.$$

It is also weakly acyclic, since its underlying digraph has no simple cycles of length at least 2.

Also, provided the number of variables $n$, construct an automaton $T = (Q_T, \Sigma, \delta_T)$ as follows. Take $Q_T = \{a, p_1, \ldots, p_{n+1}, b\}$, $\Sigma = \{0, 1, r\}$. Define $\delta(p_i, x) = p_{i+1}$ for each $i$, $1 \leq i \leq n$, and $x \in \{0, 1\}$, and $\delta(p_{n+1}, x) = b$ for $x \in \{0, 1\}$. Define also $\delta(a, x) = a$ and $\delta(b, x) = b$ for each $x \in \Sigma$, and $\delta(p_i, r) = a$ for $1 \leq i \leq n + 1$. See Figure 7.4 for an example. This automaton is monotonic, since it respects the order $a, p_1, \ldots, p_{n+1}, b$, and it is obviously weakly acyclic.

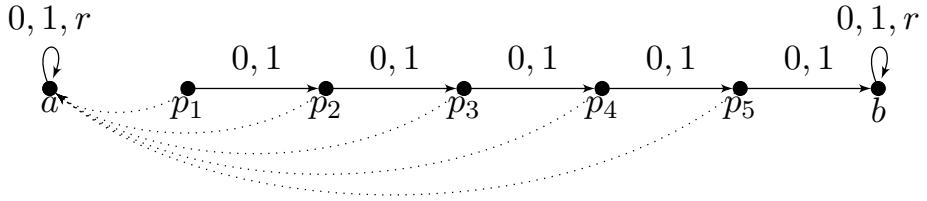


Figure 7.4: The automaton $T$ for $n = 4$ variables. Dotted arrows represent transitions for the letter $r$.

First, we prove NP-completeness of the following problem.

FINITE AUTOMATA INTERSECTION
*Input*: Automata $\mathcal{A}_1, \ldots, \mathcal{A}_k$ (with initial and accepting states);
*Output*: Yes if there is a word which is accepted by all automata, No otherwise.

This problem is PSPACE-complete for general automata [66], and NP-complete for binary weakly acyclic automata (Proposition 76). Some results on this problem are surveyed in [52]. Blondin et al. [18] provide further results on the problem. Some complexity lower bounds are presented in [116] and [38].

**Theorem 96.** *The* FINITE AUTOMATA INTERSECTION *problem is NP-complete for monotonic weakly acyclic automata over a three-letter alphabet.*

*Proof.* The fact that the problem is in NP follows from the fact that FINITE AUTOMATA INTERSECTION for weakly acyclic automata is in NP (Proposition 76).

To prove hardness, we reduce the SAT problem. For each clause $c_j \in C$, construct an automaton $\mathcal{A}_j$, and set $q_1$ as its initial state and $t$ as its only accepting state. Construct also the automaton $T$ with the initial state $p_1$ and accepting state $a$.

We claim that $C$ is satisfiable if and only if all automata in $\{\mathcal{A}_j \mid c_j \in C\} \cup \{T\}$ accept a common word $w$. Indeed, assume that there is a common word accepted by all these automata. Then none of the first $n$ letters of this word can be $r$, otherwise all automata $\mathcal{A}_j$ are mapped to $s$, which is a non-accepting sink state. The next letter has to be $r$, otherwise $T$ is mapped to $b$, which is a non-accepting sink state. But that means that in each $\mathcal{A}_j$, the set $q_1$ is mapped by an $n$-letter word $z_1 \ldots z_n$ to the accepting state $t$. Thus, by construction, the assignment $x_i = z_i$ satisfies all clauses in $C$.

By the same reasoning, if the assignment $x_i = z_i$, $1 \leq i \leq n$, satisfies all clauses in $C$, then $z_1 \ldots z_n r$ is a word accepted by all automata. □

Now we switch to a related SET RANK problem. Its general case is discussed in Section 6.4. Recall that it is formulated as follows.

SET RANK
*Input*: An automaton $\mathcal{A}$ and a set $S$ of its states;
*Output*: The rank of $S$.

To get inapproximability results for monotonic automata, we use the following problem.

MAX-3SAT
*Input*: A set $X$ of $n$ Boolean variables and a set $C$ of $m$ 3-term clauses;
*Output*: The maximum number of clauses that can be simultaneously satisfied by some assignment of values to the variables.

This problem cannot be approximated in polynomial time within a factor of $\frac{7}{8}+\epsilon$ for any $\epsilon > 0$ unless P = NP [50].

**Theorem 97.** *The* SET RANK *problem cannot be approximated in polynomial time within a factor of* $\frac{9}{8}-\epsilon$ *for any* $\epsilon > 0$ *in monotonic weakly acyclic automata over a three-letter alphabet unless* $P = NP$.

*Proof.* We reduce the MAX-3SAT problem. For each clause $c_j \in C$, construct an automaton $\mathcal{A}_j$. Construct also $m$ copies of the automaton $T$, denoted $T_j$, $1 \le j \le m$. Define an automaton $\mathcal{A}$ with the set of states which is the union of all sets of states of $\{\mathcal{A}_j, T_j \mid 1 \le j \le m\}$, alphabet $\Sigma$ and transition functions defined in all constructed automata. For each $j$, identify the state $t$ in $\mathcal{A}_j$ with the state $a$ in $T_j$. Take $S$ to be the set of states $q_1$ from each automaton $\mathcal{A}_j$, and $p_1$ from each $T_j$. The constructed automaton is monotonic and weakly acyclic.

If $h$ is the minimum number of clauses in $C$ that are not satisfied by an assignment, the set $S$ has rank $m+h$. Indeed, consider an assignment $x_i = z_i$, $1 \le i \le n$, not satisfying exactly $h$ clauses in $C$. Then the word $z_1 \dots z_n r$ has rank $m+h$ with respect to $S$.

In the other direction, let $w$ be a word of minimum rank with respect to the set $S$. If any of the first $n$ letters of $w$ is $r$, then $q_1$ in each $\mathcal{A}_i$ is mapped to $s$ in the corresponding automaton, and thus $w$ has rank $2m$ with respect to $S$. The same is true if $(n+1)$st letter of $w$ is not $r$, because then $p_1$ in each $T_i$ is mapped to $b$ in the corresponding automaton. If first $n$ letters $z_1, \dots, z_n$ of $w$ are not $r$, and the next letter is $r$, then the assignment $x_i = z_i$ does not satisfy exactly $h'$ clauses, where $m+h'$ is the rank of the word $w$ with respect to $S$. For the word of minimum rank, we get the required equality.

It is NP-hard to decide between (i) all clauses in $C$ are satisfiable and (ii) at most $(\frac{7}{8}+\epsilon)m$ clauses in $C$ can be satisfied by an assignment [50]. In the case (i), the rank of $S$ is $m$, in the case (ii) it is at least $m + (\frac{1}{8}-\epsilon)m$. Since it is NP-hard to decide between this two options, we get $(\frac{9}{8}-\epsilon)$-inapproximability for any $\epsilon > 0$. □

By using an argument similar to the proof of Theorem 97, we can show inapproximability of the maximization version of FINITE AUTOMATA INTERSECTION (where we are asked to find a maximum number of automata accepting a common word). Indeed, take $m$ copies of $T$ together with the set $\{\mathcal{A}_j \mid c_j \in C\}$ as the input of FINITE AUTOMATA INTERSECTION and reduce MAX-3SAT to it (input and accepting states are assigned according to the construction in Theorem 96). Then the maximum number of automata accepting a common word is $m+g$, where $g$ is the maximum number of simultaneously satisfied clauses in $C$, since all copies of $T$ have to accept this word. Thus it is NP-hard to decide between (i) all $2m$ automata accept a common word and (ii) at most $m + (\frac{7}{8}+\epsilon)m$ automata accept a common word, and we get the following result.

**Corollary 98.** *The maximization version of the* FINITE AUTOMATA INTERSECTION *problem cannot be approximated in polynomial time within a factor of* $\frac{15}{16}+\epsilon$ *for any* $\epsilon > 0$ *in monotonic weakly acyclic automata over a three-letter alphabet unless* $P = NP$.

Consider now the Careful Synchronization problem for monotonic automata.

**Theorem 99.** *The* Careful Synchronization *problem is NP-hard for monotonic automata over a four-letter alphabet.*

*Proof.* We reduce the SAT problem. Let $\mathcal{A}'_j$ be $\mathcal{A}_j$ with alphabet restricted to $\{0,1\}$ and without the state $s$, and $T'$ be $T$ with alphabet restricted to $\{0,1\}$ and without the states $a, b$. Let $\Sigma' = \{0,1,y,z\}$. Provided $X$ and $C$, construct $\mathcal{A}'_j$ for each clause $c_j$, and also construct $T'$. Let $Q'$ be the union of all states of each $\mathcal{A}'_j$, all states of $T'$ and a new state $f$. We expand already defined (for the letters $0,1$) transition function $\delta$ as follows. Define $y$ to map all states in each $\mathcal{A}'_j$ to $q_1$ in this gadget, and all states in $T'$ to $p_1$. Finally, define $z$ to map the state $t$ in each $\mathcal{A}_j$, the state $p_{n+1}$ in $T$ and the state $f$ to $f$. Leave all other transition undefined. We denote thus obtained automaton as $\mathcal{A}' = (Q', \Sigma', \delta')$.

Any word $w$ carefully synchronizing $\mathcal{A}'$ begins with $y$, since it is the only letter defined for all states. After applying it, the set $S$ of reached states consists of $q_1$ in each $\mathcal{A}_j$, $p_1$ in $T$, and $f$. To reach $f$ from this set, we need to apply $z$ at least once. Right before applying $z$, the set of reached states must consist of exactly $n$ letters of the set $\{0,1\}$, because application of $y$ takes us back to $S$, and applying $0,1$ more than $n$ times is not defined for the state $p_1$. Thus, in each $\mathcal{A}_j$ the state $q_1$ must be mapped to $t$ by $n$ 0s and 1s which is possible if and only if $C$ is satisfiable. □

**Question 100.** *What is the complexity of the mentioned problems for binary monotonic automata? Do the problems discussed in this section belong to NP for monotonic automata?*

We note that it does not matter in the provided reductions whether a linear order preserved by all transitions is known or not.

# Chapter 8

# Finite Prefix Codes

All automata in this chapter are deterministic, so we say partial automaton for partial DFA, and complete automaton, or just "automaton" for complete DFA

## 8.1 Introduction

Despite the interest in synchronizing finite prefix codes, the computational complexity of finding short synchronizing words for them has not been studied so far. In this chapter, we provide a systematic investigation of this topic.

Each recognizable (by a finite automaton) maximal prefix code can be represented by an automaton decoding the star of this code (see Section 2.4). For a finite code, this automaton can be exponentially smaller than the representation of the code by listing all its words (consider, for example, the code of all words of some fixed length). This can, of course, happen even if the code is synchronizing. An important example here is the code $0\{0,1\}^{n-1} \cup 1\{0,1\}^n$. The minimized decoder of this code is a famous Wielandt automaton with $n+1$ states [5], while the literal automaton contains $2^{n-1}+2^n$ states, see Figure 8.1 for the case $n=3$. In different applications, the first or the second way of representing the code can be useful. In some cases large codes having a short description may be represented by a minimized decoder, while in other applications the code can be described by simply providing the list of all codewords. The number of states of the literal decoder is equal to the number of different prefixes of the codewords, and thus the representations of a prefix code by listing all its codewords and by providing its literal automaton are polynomially equivalent. We study the complexity of problems for both arbitrary and literal decoders of finite prefix codes.

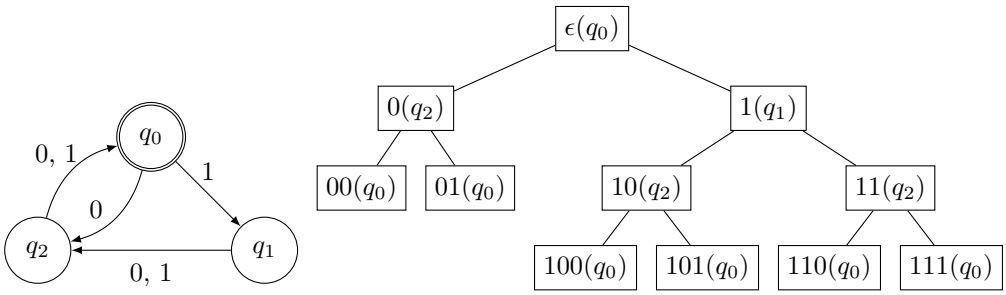


Figure 8.1: The Wielandt automaton on three states and the tree of the code $0\{0,1\} \cup 1\{0,1\}^2$.

In this chapter we study the existence of approximation algorithms for the problem Short Sync Word of finding a shortest synchronizing words in several classes of deterministic automata decoding prefix codes. In Section 8.2 we provide a strong inapproximability result for this problem in the class of strongly connected automata. Section 8.3 is devoted to the same problem in acyclic automata, which are then used in Section 8.4 to show logarithmic inapproximability of Short Sync Word in the class of Huffman decoders. In Section 8.5 we provide a much stronger inapproximability result for partial Huffman decoders. In Section 8.6 we provide several algorithms for literal Huffman decoders and conjecture that Short Sync Word can be solved in polynomial time in this class. Finally, in Section 8.7 we apply the developed techniques to the problems of finding shortest mortal and avoiding words.

## 8.2 The Construction of Gawrychowski and Straszak

In this section we briefly recall the construction of a gadget invented by Gawrychowski and Straszak [43] to show $n^{1-\varepsilon}$-inapproximability of the Short Sync Word problem in the general class of automata. Below we will use this construction several times.

Suppose that we have a constraint satisfiablity problem (CSP) with $N$ variables and $M$ constraints such that each constraint is satisfied by at most $K$ assignments (see [43] for the definitions and missing details). Following the results in [43], we can assume that $N, K \leq M^{\varepsilon}$, and also that either the CSP is satisfiable, or at most $\frac{1}{M^{1-\varepsilon}}$ fraction of all constraints can be satisfied by an assignment. It is possible to construct the following ternary automaton $A_\phi$ in polynomial time. For each constraint $C$ the automaton $A_\phi$ contains a corresponding binary (over $\{0,1\}$) gadget $T_C$ which is a compressed tree (that is, an acyclic digraph) of height $N$ and the number of states at most $N^2K$ having different leaves corresponding to satisfying and non-satisfying assignments. The automaton $A_\phi$ also contains a sink state $s$ such that all the leaves corresponding to satisfying assignments are mapped to $s$, and all other leaves are mapped to the roots of the corresponding trees. The third letter is defined to map all the states of each gadget to its root and to map $s$ to itself. For every $\varepsilon > 0$ it is possible to construct such an automaton with at most $MN^2K \leq M^{1+3\varepsilon}$ states in polynomial time. Moreover, for a satisfiable CSP we get an automaton with a shortest synchronizing word of length at most $N + 1 = O(M^{\varepsilon})$, and for a non-satisfiable CSP the length of a shortest synchronizing word is at least $NM^{1-\varepsilon} \geq M^{1-\varepsilon}$. Since $\varepsilon$ can be chosen arbitrary small, this provides a gap-preserving reduction with a gap of $M^{1-\varepsilon}$.

The described construction can be modified to get the same inapproximability in the class of strongly connected automata.

**Theorem 101.** *The* Short Sync Word *problem cannot be approximated in polynomial time within a factor of $n^{1-\varepsilon}$ for every $\varepsilon > 0$ for $n$-state binary strongly connected automata unless $P = NP$.*

*Proof.* Consider the automaton $A_\phi$ described above. Add a new letter $c$ that cyclically permutes the roots of all gadgets, maps $s$ to the root of one of the gadgets and acts as a self-loop for all the remaining states. Observe that thus constructed automaton $A$ is strongly connected and has the property that every non-satisfying assignment satisfies at most the fraction of $\frac{1}{M^{1-\varepsilon}}$ of all constraints. Thus, the gap between the length of a shortest synchronizing word for a satisfying and non-satisfying assignment is still $\Theta(M^{1-\varepsilon})$.

It remains to make the automaton binary. This can be done by using Lemma 3 of [11]. This way we get a binary automaton with $\Theta(M^{1+3\varepsilon})$ states and a gap between $\Theta(M^{\varepsilon})$ and $\Theta(M^{1-\varepsilon})$ in the length of a shortest synchronizing word. By choosing appropriate small enough $\varepsilon$, we get a reduction with gap $n^{1-\varepsilon}$ for binary strongly connected $n$-state automata, which proves the statement. □

## 8.3 Acyclic Automata

In this section we investigate the simply-defined classes of weakly acyclic and strongly acyclic automata. The results for strongly acyclic automata are used in Section 8.4 to obtain inapproximability for Huffman decoders. Even though the automata in the classes of weakly and strongly acyclic automata are very restricted and have a very simple structure, the inapproximability bounds for them are quite strong. Thus we believe that these classes are of independent interest.

We will need the following problem.

SET COVER
*Input*: A set $X$ of $p$ elements and a family $C$ of $m$ subsets of $X$;
*Output*: A subfamily of $C$ of minimum size covering $X$.

A family $C'$ of subsets of $X$ is said to *cover* $X$ if $X$ is a subset of the union of the sets in $C'$. For every $\gamma > 0$, the SET COVER problem with $|C| \leq |X|^{1+\gamma}$ cannot be approximated in polynomial time within a factor of $c' \log p$ for some $c' > 0$ unless P = NP [11].

**Theorem 102.** *The* SHORT SYNC WORD *problem cannot be approximated in polynomial time within a factor of $c \log n$ for some $c > 0$ for $n$-state strongly acyclic automata over an alphabet of size $n^{1+\gamma}$ for every $\gamma > 0$ unless $P = NP$.*

*Proof.* We reduce the SET COVER problem. Provided $X$ and $C$, we construct the automaton $A = (Q, \Sigma, \delta)$ as follows. To each set $c_k$ in $C$ we assign a letter $k \in \Sigma$. To each element $x_j$ in $X$ we assign a "pipe" of states $q_1^{(j)}, \ldots, q_p^{(j)}$ in $Q$. Additionally, we construct a state $f$ in $Q$.

For $1 \leq i \leq p-1$ and all $k$ and $j$ we define $\delta(q_i^{(j)}, k) = f$ if $c_k$ contains $x_j$, and $\delta(q_i^{(j)}, k) = q_{i+1}^{(j)}$ otherwise. We also define $\delta(q_p^{(j)}, k) = f$ for all $j$ and $k$.

We claim that the length of a shortest synchronizing word for $A$ is equal to the minimum size of a set cover in $C$. Let $C'$ be a set cover of minimum size. Then a concatenation of the letters corresponding to the elements of $C'$ is a synchronizing word of corresponding length.

In the other direction, consider a shortest synchronizing word $w$ for $A$. No letter appears in $w$ at least twice. If the length of $w$ is less than $p$, then by construction of $A$ the subset of elements in $C$ corresponding to the letters in $w$ form a set cover. Otherwise we can take an arbitrary subfamily of $C$ of size $p$ which is a set cover (such subfamily trivially exists if $C$ covers $X$).

The resulting automaton has $p^2 + 1$ states and $m$ letters. Thus we get a reduction with gap $c' \log p \geq c'' \log \sqrt{|Q|} = \frac{1}{2} c'' \log |Q|$ for some $c'' > 0$. Because of the mentioned result of Berlinkov, we can also assume that $m < p^{1+\gamma}$ for arbitrary small $\gamma > 0$. □

Now we are going to extend this result to the case of binary weakly acyclic automata.

**Corollary 103.** *The* SHORT SYNC WORD *problem cannot be approximated in polynomial time within a factor of $c \log n$ for some $c > 0$ for $n$-state binary weakly acyclic automata unless $P = NP$.*

*Proof.* We extend the construction from the proof of Theorem 102 by using Lemma 3 of [11]. If we start with a strongly acyclic automaton with $p$ states and $k$ letters, this results in a binary weakly acyclic automaton with $4pk$ states. Moreover, the length of a shortest word of the new automaton is between $\ell(\log k + 1)$ and $(\ell + 1)(\log k + 1)$, where $\ell$ is the length of a shortest word of the original automaton. Since we can assume $p < k < p^{1+\gamma}$ for arbitrary small $\gamma > 0$, we have $\log n = \Theta(\log p)$, where $n$ is the number of states of the new automaton. Thus we get a gap of $\Theta(\log p) = \Theta(\log n)$. □

For ternary strongly acyclic automata it is possible to get $(2 - \varepsilon)$-inapproximability.

**Theorem 104.** *The* Short Sync Word *problem cannot be approximated in polynomial time within a factor of* $2-\varepsilon$ *for every* $\varepsilon > 0$ *for* $n$*-state strongly acyclic automata over an alphabet of size three unless* $P = NP$*.*

## 8.4 Huffman Decoders

We start with a statement relating strongly acyclic automata to Huffman decoders.

**Lemma 105.** *Let* $A$ *be a synchronizing strongly acyclic automaton over an alphabet of size* $k$*. Let* $\ell$ *be the length of a shortest synchronizing word for* $A$*. Then there exists a Huffman decoder* $A_H$ *over an alphabet of size* $k+2$ *with the same length of a shortest synchronizing word, and* $A_H$ *can be constructed in polynomial time.*

*Proof.* Provided a strongly acyclic automaton $A = (Q, \Sigma, \delta)$ we construct a Huffman decoder $A_H = (Q_H, \Sigma_H, \delta_H)$.

Since $A$ is a synchronizing strongly acyclic automaton, it has a unique sink state $f$. We define the alphabet $\Sigma_H$ as the union of $\Sigma$ with two additional letters $b_1, b_2$. The set of states $Q_H$ is the union of $Q$ with some auxiliary states defined as follows. Consider the set $S$ of states in $A$ having no incoming transitions. Construct a full binary tree with the root $f$ having $S$ as the set of its leaves (if $|S|$ is not a power of two, some subtrees of the tree can be merged). Define $b_1$ to map each state of this tree to the left child, and $b_2$ to the right child. Transfer the action of $\delta$ to $\delta_H$ for all states in $Q$ and all letters in $\Sigma$. For all the internal states of the tree define all the letters of $\Sigma$ to map these states to $f$. Finally, for all the states in $Q$ define the action of $b_1, b_2$ in the same way as some fixed letter in $\Sigma$.

Observe that any word $w$ over alphabet $\Sigma$ synchronizing $A$ also synchronizes $A_H$. In the other direction, any synchronizing word for $A_H$ has to synchronize $Q$, which means that each state in $Q$ has to be mapped to $f$ first, so the length of a shortest synchronizing word for $A_H$ is at least the length of a shortest synchronizing word for $A$. □

Now we use Lemma 105 to get preliminary inapproximability results for Huffman decoders.

**Corollary 106.** *(i) The* Short Sync Word *problem is NP-complete for Huffman decoders over an alphabet of size* 4*.*

*(ii) The* Short Sync Word *problem cannot be approximated in polynomial time within a factor of* $2-\varepsilon$ *for every* $\varepsilon > 0$ *for Huffman decoders over an alphabet of size* 5 *unless* $P = NP$*.*

*(iii) For every* $\gamma > 0$*, the* Short Sync Word *problem cannot be approximated in polynomial time within a factor of* $c \log n$ *for some* $c > 0$ *for Huffman decoders over an alphabet of size* $n^{1+\gamma}$ *unless* $P = NP$*.*

*Proof.* (i) The automaton in the Eppstein's proof of NP-completeness of Short Sync Word [37] is strongly acyclic. Then the reduction described in Lemma 105 can be applied.

(ii) A direct consequence of Theorem 104 and Lemma 105.

(iii) A direct consequence of Theorem 102 and Lemma 105. □

Now we show how to get a better inapproximability result for binary Huffman decoders using the composition of synchronizing prefix codes. We present a more general result for the composition of synchronizing codes which is of its own interest. This result shows how to change the size of the alphabet of a synchronizing complete code in such a way that the approximate length of a shortest synchronizing pair for it is preserved.

A pair $(\ell_X, r_X)$ of words in $X^*$ is called *absorbing* if $\ell_X\Sigma^* \cap \Sigma^* r_X \subseteq X^*$. The *length* of a pair is the total length of two word. A complete code admits an absorbing pair if and only if it is synchronizing [14].

Let $Y$ be a code over $\Sigma_Y$ and $Z$ be a code over $\Sigma_Z$. Suppose that there exists a bijection $\beta : \Sigma_Y \to Z$. The *composition* $Y \circ_\beta Z$ is then defined as the code $X = \{\beta(y) \mid y \in Y\}$ over the alphabet $\Sigma_Z$ [14]. Here $\beta(y)$ is defined as $\beta(y_1)\beta(y_2)\dots\beta(y_k)$ for $y = y_1y_2\dots y_k$, $y_i \in \Sigma_Y$. Sometimes $\beta$ is omitted in the notation of composition.

**Theorem 107.** *Let $Y$ and $Z$ be two synchronizing complete codes, such that $Z$ is finite and $m$ and $M$ are the lengths of a shortest and a longest codeword in $Z$. Suppose that the composition $Y \circ Z$ is defined. Then the code $X = Y \circ Z$ is synchronizing, and the length of a shortest absorbing pair for $X$ is between $m\ell$ and $2M\ell + 2c$, where $\ell$ is the length of a shortest absorbing pair for $Y$ and $c$ is the length of a shortest absorbing pair for $Z$.*

*Proof.* Let $Y \subseteq \Sigma_Y^*$, $X, Z \subseteq \Sigma_Z^*$, and $\beta : \Sigma_Y \to Z$ be such that $X = Y \circ_\beta Z$. First, assume that $Y$ and $Z$ are synchronizing, and let $(\ell_Y, r_Y)$, $(\ell_Z, r_Z)$ be shortest absorbing pairs for $Y$ and $Z$. Then $\ell_Y\Sigma_Y^* \cap \Sigma_Y^* r_Y \subseteq Y^*$ and $\ell_Z\Sigma_Z^* \cap \Sigma_Z^* r_Z \subseteq Z^*$. We will show that $p_1 = (\beta(\ell_Y)\ell_Z r_Z\beta(r_Y), \beta(\ell_Y)\ell_Z r_Z\beta(r_Y))$ is an absorbing pair for $X$. Consider the set $\beta(\ell_Y)\ell_Z r_Z\beta(r_Y)\Sigma_Z^* \cap \Sigma_Z^*\beta(\ell_Y)\ell_Z r_Z\beta(r_Y)$. It is a subset of the set $\beta(\ell_Y)(\ell_Z\Sigma_Z^* \cap \Sigma_Z^* r_Z)\beta(r_Y) \subseteq \beta(\ell_Y)Z^*\beta(r_Y) = \beta(\ell_Y\Sigma_Y^* r_Y) \subseteq \beta(Y^*) = X^*$. Thus, $p_1$ is an absorbing pair for $X$. Moreover, the length of this pair is between $2m(|\ell_Y| + |r_Y|) + 2(|\ell_Z| + |r_Z|)$ and $2M(|\ell_Y| + |r_Y|) + 2(|\ell_Z| + |r_Z|)$.

Conversely, assume that $(\ell_X, r_X)$ is a shortest absorbing pair for $X$, hence $\ell_X\Sigma_Z^* \cap \Sigma_Z^* r_X \subseteq X^*$. Then by the definition of composition $X^* \subseteq Z^*$ and $\ell_X, r_X \in Z^*$; thus, $(\ell_X, r_X)$ is also absorbing for $Z$. Next, let $\ell_Y = \beta^{-1}(\ell_X)$, $r_Y = \beta^{-1}(r_X)$, $\ell_Y, r_Y \in Y^*$. Then $\beta(\ell_Y\Sigma_Y^* \cap \Sigma_Y^* r_Y) = \ell_X Z^* \cap Z^* r_X \subseteq \ell_X\Sigma_Z^* \cap \Sigma_Z^* r_X \subseteq X^* = \beta(Y^*)$. Since the mapping $\beta$ is injective, $\ell_Y B^* \cap B^* r_Y \subseteq Y^*$. Consequently $Y$ is synchronizing, and $(\ell_Y, r_Y)$ is an absorbing pair for it of length between $\frac{1}{M}(|\ell_X| + |r_X|)$ and $\frac{1}{m}(|\ell_X| + |r_X|)$.

Summarizing, we get that the length of a shortest absorbing pair for $X$ is between $m(|\ell_Y| + |r_Y|)$ and $2M(|\ell_Y| + |r_Y|) + 2(|\ell_Z| + |r_Z|)$. □

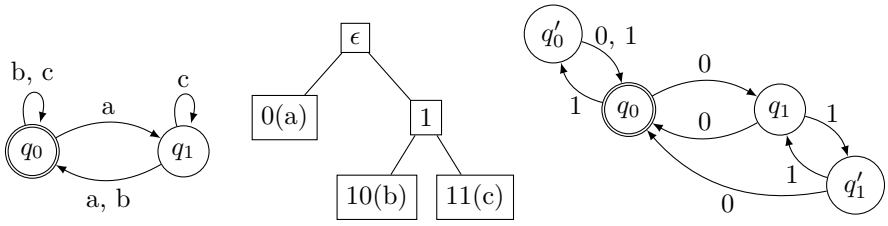


Figure 8.2: An automaton recognizing some infinite maximal prefix code, the tree of a finite maximal prefix code and an automaton recognizing their composition.

In the case of finite maximal prefix codes the first element of the absorbing pair can be taken as an empty word. The second element is then a synchronizing word. For recognizable maximal prefix codes $Y$ and $Z$, where $Z$ is finite, a Huffman decoder recognizing the star of $X = Y \circ Z$ can be constructed as follows. Let $H_Y$ be a Huffman decoder for $Y$. Consider the full tree $T_Z$ for $Z$, where each edge is marked by the corresponding letter. For each state $q$ in $H_Y$ we substitute the transitions going from this state with a copy of $T_Z$ as follows. The root of $T_Z$ coincides with $q$, and the inner vertices are new states of the resulting automaton. Suppose that $v$ is a leaf of $T_Z$, and the path from the root to $v$ is marked by a word $w$. Let $a$ be the letter of the alphabet of $H_Y$ which

is mapped to the word $w$ in the composition. Then the image of $q$ under the mapping defined by $a$ is merged with $v$. In such a way we get a Huffman decoder with $\Theta(n_Y n_Z)$ states, where $n_Y, n_Z$ is the number of states in $H_Y$ and $T_Z$. By the definition of composition, this decoder has the same alphabet as $Z$. See Figure 8.2 for an example.

**Corollary 108.** *The* Short Sync Word *problem cannot be approximated in polynomial time within a factor of $c \log n$ for some $c > 0$ for binary $n$-states Huffman decoders unless $P = NP$.*

*Proof.* We start with claim (iii) in Corollary 106 and use Theorem 107 to reduce the size of the alphabet. Thus, we reduce Short Sync Word for Huffman decoders over an alphabet of size $n^{1+\gamma}$ to Short Sync Word for binary Huffman decoders.

Assume that the size of the alphabet $k = n^{1+\gamma}$ is a power of two (if no, duplicate some letter the required number of times). We take the code $0\{0,1\}^{\log k - 1} \cup 1\{0,1\}^{\log k}$ as $Z$. This is a code where some words are of length $\log k$ and the other words are of length $\log k + 1$ (after minimization the star of this code is recognized by a Wielandt automaton with $\log k + 1$ states discussed in the introduction). This code has a synchronizing word of length $\Theta((\log k)^2)$ [5]. The number of vertices in the tree of this code is $\Theta(n)$.

Let $\ell$ be the length of a shortest synchronizing word for the original automaton. By Theorem 107, the length of a shortest synchronizing word for the result of the composition is between $\ell \log k$ and $2(\log k + 1)\ell + \Theta((\log k)^2) = \Theta((\ell + \log k) \log k)$.

For the Set Cover problem the inapproximability result holds even if we assume that the size of the optimal solution is of size at least $d \log |X|$ for some $d > 0$. Indeed, if $d$ is a constant we can check all the subsets of $C$ of size at most $d \log |X|$ in polynomial time. Thus, we can assume that $\ell \geq \log k$ implying $(\ell + \log k) \log k = \Theta(\ell \log k)$. Hence after the composition the length of a shortest synchronizing word is changed by at most constant multiplicative factor, and we we get a gap-reserving reduction with gap $\Theta(\log n)$. The resulting automaton is of size $\Theta(n^{2+\gamma})$, and the $(2 + \gamma)$ dependence is hidden in the constant $c$ in the statement of this corollary. □

## 8.5 Partial Huffman Decoders

In this section we investigate automata recognizing the star of a non-maximal finite prefix code. Such codes have some noticeable properties which do not hold for maximal finite prefix codes. For example, there exist non-trivial non-maximal finite prefix codes with finite synchronization delay, which provides guarantees on the synchronization time [22]. This allows to read a stream of correctly transmitted compressed data from arbitrary position, which can be useful for audio and video decompression.

We provide a lower bound on the approximability of the Short Sync Word problem for partial Huffman decoders by extending the idea used to prove inapproximability for Huffman decoders in the previous sections. First we prove the result for alphabet of size 5 and then use a composition with a maximal finite prefix code to get the same result for the binary case.

**Theorem 109.** *The* Short Sync Word *problem cannot be approximated within a factor of $n^{\frac{1}{2}-\varepsilon}$ for every $\varepsilon > 0$ for $n$-state partial Huffman decoders over an alphabet of size 5 unless $P = NP$.*

*Proof.* First we prove inapproximability for the class of partial strongly acyclic automata, that is, automata having no simple cycles but loops in the sink state. We start with the CSP problem described in Section 8.2 with all the restriction defined there. Having an instance of this problem with $N$ variables and $M$ constraints such that each constraint is satisfied by at most $K$ assignments, we construct an automaton $A^b_\phi$ over the alphabet $\{0,1\}$. For each constraint $j$, we construct $M$

identical compressed trees $T_j^1, \ldots, T_j^M$ corresponding to this constraint (also described in Section 8.2). Then for $1 \leq i \leq M-1$ we merge the leaves of $T_j^i$ corresponding to non-satisfying assignments with the root of $T_j^{i+1}$, and delete all the leaves corresponding to satisfying assignments (leaving all the transition leading to deleted states undefined). For each $T_j^M$, we again delete all the leaves corresponding to satisfying assignments and merge all the leaves corresponding to non-satisfying assignments with a new state $s$. This state is a self-loop, that is, $0, 1$ map $s$ to itself. Now we define an additional letter $a$ and $M$ new states $r_1, \ldots, r_M$. We define $a$ to map $r_j$ to the root of $T_j^1$. Finally, we add $N+1$ new states $c_0, \ldots, c_N$ such that $a$ maps $c_0$ to $c_1$, and $0, 1$ map $c_i$ to $c_{i+1}$ for $1 \leq i \leq N-1$, and map $c_N$ to $s$. All other transitions are left undefined.

If $a$ is applied first, the set $S$ of states to be synchronized is $c_1$ together with the roots of $T_j^1$ for all $j$. Observe that $a$ cannot be applied anymore, since it would result in mapping all the active states of the automaton to void. If a letter other than $a$ is applied first, a superset of $S$ must be synchronized then.

If there exists a satisfying assignment $x_1, \ldots, x_N$ then the word $ax_1 \ldots x_N$ is synchronizing, since it maps all the states but $c_0$ to void. Otherwise, to synchronize the automaton we need to pass through $M$ compressed trees, since each tree can map only at most $M^\varepsilon$ states to void (since for every non-satisfiable CSP the maximum number of satisfiable constraints is $M^\varepsilon$ in the construction, see Section 8.2). Thus we get a gap of $M^{1-\varepsilon} = n^{\frac{1}{2}-\varepsilon}$ for the class of $n$-state strongly acyclic partial automata.

Now we are going to transfer this result to the case of partial Huffman decoders. We extend the idea of Lemma 105. All we need is to define transitions leading from $s$ to the states having no incoming transitions (which are $r_1, \ldots, r_M$ together with $c_0$). The only difference is that now we have to make sure that $a$ cannot be applied too early resulting in mapping all the states of the compressed trees to void leaving the state $s$ active.

To do that, we introduce two new letters $b_1, b_2$ and perform branching as described in Lemma 105. Thus we get $M+1$ leaves of the constructed full binary tree. To each leave we attach a chain of states of length $MN$ ending in the root of $T_j^1$ (or in $c_0$). This means that we introduce $MN$ new states and define the letters $b_1, b_2$ to map a state in each chain to the next state in the same chain. This guarantees that if the letter $a$ appears twice in a word of length at most $MN$, this word maps all the states of the automaton to void. Finally, the action of $b_1, b_2$ on the compressed trees and the states $c_0, \ldots, c_N$ repeats, for example, the action of the letter 0.

The number of states of the automaton in the construction is $O(M^{2+3\varepsilon})$ The gap is then $M^{1-2\varepsilon}$. By choosing small enough $\varepsilon$ we thus get a gap of $n^{\frac{1}{2}-\varepsilon}$ as required. □

The next lemma shows that under some restrictions it is possible to reduce the alphabet of a non-maximal prefix code in a way that approximate length of a shortest synchronizing word is preserved. A word is called *non-mortal* for a prefix code $X$ if it is a factor of some word in $X^*$.

**Lemma 110.** *Let $Y, Z$ be synchronizing prefix codes such that $Z$ is finite and maximal. Let $m$ and $M$ be the lengths of a shortest and a longest codeword in $Z$. Suppose that $Y \circ_\beta Z$ is defined for some $\beta$. If there exists a synchronizing word $w_Z$ for $Z$ such that $\beta^{-1}(w)$ is a non-mortal word for $Y$, then the composition $X = Y \circ_\beta Z$ is synchronizing. Moreover, then the length of a shortest synchronizing word for $X$ is between $m\ell$ and $M\ell + |w_Z|$, where $\ell$ is the length of a shortest synchronizing word for $Y$.*

*Proof.* Let $Y \subseteq \Sigma_Y^*$, $X, Z \subseteq \Sigma_Z^*$, and $\beta : \Sigma_Y \to Z$ be such that $X = Y \circ_\beta Z$. Let $w_Y$ be a synchronizing word for $Y$. Then $w_Z\beta(w_Y)$ is a synchronizing word for $X$ of length at most $M\ell + |w_Z|$. In the other direction, let $w_X$ be a synchronizing word for $X$. Then $\beta^{-1}(w_X)$ is a synchronizing word for $Y$. Thus, $|w_X| \geq m\ell$. □

**Corollary 111.** *The* Short Sync Word *problem cannot be approximated in polynomial time within a factor of* $n^{\frac{1}{2}-\varepsilon}$ *for every* $\varepsilon > 0$ *for binary* $n$*-state partial Huffman decoders unless* $P = NP$*.*

*Proof.* We use the composition of the automaton constructed in the proof of Theorem 109 with the prefix code $\{aaa, aab, ab, ba, bb\}$ having a synchronizing word $baab$. The word $baab$ is a concatenation of two different codewords, so their pre-images can be taken to be $a$ and 0, resulting in a non-mortal word $a0$ for $A$, so we can use Lemma 110. □

## 8.6 Literal Huffman Decoders

In this section we deal with literal Huffman decoders. We call the literal automaton of a finite maximal prefix code a *literal Huffman decoder*. We will need the following useful lemma.

**Lemma 112** ( [12, Lemma 16])**.** *For every* $n$*-state literal Huffman decoder over an alphabet of size* $k$ *there exists a word of length* $\lceil \log_k n \rceil$ *and rank at most* $\lceil \log_k n \rceil$*.*

Note that if a $n$-state literal Huffman decoder has a synchronizing word of length at most $O(\log n)$, this word can be found in polynomial time by examining all words of length up to $O(\log n)$. Thus, in further algorithms we will assume that the length of a shortest synchronizing word is greater than this value. Lemma 112 stays that a word of rank at most $\lceil \log_k n \rceil$ can also be found in polynomial time.

**Theorem 113.** *There exists a* $O(\log n)$*-approximation polynomial time algorithm for the* Short Sync Word *problem for literal Huffman decoders.*

*Proof.* Let $A = (Q, \Sigma, \delta)$ be a literal Huffman decoder, and $|\Sigma| = k$. Let $w$ be a word of rank at most $\lceil \log_k n \rceil$ found as described above. Let $Q'$ be the image of $Q$ under the mapping defined by $w$, i.e. $Q' = \delta(Q, w)$. Define by $v$ the word subsequently merging pairs of states in $Q'$ with shortest possible words. Note than a shortest word synchronizing $A$ has to synchronize every pair of states, in particular, one that requires a longest word. Thus the length of $v$ is at most $\lceil \log_k n \rceil$ times greater than the length of a shortest word synchronizing $A$. Then the word $wv$ is a $O(\log n)$-approximation for the Short Sync Word problem. □

**Theorem 114.** *For every* $\varepsilon > 0$*, there exists a* $(1+\varepsilon)$*-approximation* $O(n^{\log n})$*-time algorithm for the problem* Short Sync Word *for* $n$*-state literal Huffman decoders.*

*Proof.* Let $A = (Q, \Sigma, \delta)$ be a literal Huffman decoder, and $|\Sigma| = k$. First we check all words of length at most $\frac{1}{\varepsilon}\lceil \log_k n \rceil$, whether they are synchronizing. The number of these words is polynomial, and the check can be performed in polynomial time. If a synchronizing word is found then we have an exact solution. Otherwise, a shortest synchronizing word must be longer than that and we proceed to the second stage.

Let $w$ be a word of rank at most $\lceil \log_k n \rceil$ found as before. Now we construct the power automaton $A^{\leq \lceil \log_k n \rceil}$ restricted to all the subsets of size at most $\lceil \log_k n \rceil$. Using it, we find a shortest word synchronizing the subset $\delta(Q, w)$; let this word be $v$. We return $wv$.

Let $w'$ be a shortest synchronizing word for $|A|$. Clearly, $|w'| \geq |v|$ and $\varepsilon|w'| > \lceil \log_k n \rceil$. Thus $|wv| \leq (1+\varepsilon)|w'|$, so $wv$ is a $(1+\varepsilon)$-approximation as required. □

In view of the presented results we propose the following conjecture.

**Conjecture 115.** *There exists an exact polynomial time algorithm for the* Short Sync Word *problem for literal Huffman decoders.*

Finally, we remark that it is possible to define the notion of the literal automaton of a non-maximal finite prefix code in the same way. In this case we leave undefined the transitions for a state $w$ and a letter $a$ such that $w$ is a proper prefix of a codeword, but $wa$ is neither a proper prefix of a codeword nor a codeword itself. However, the statement of Lemma 112 is false for partial automata. Indeed, consider a two-word prefix code $\{(0^n1^n)^n, (1^n0^n)^n\}$. Its literal automaton has $2n^2 - 1$ states, and a shortest synchronizing word for it is $0^{n+1}$ of length $n + 1$. Every word of length at most $n$ which is defined for at least one state is of the form $0^*1^*$ or $1^*0^*$ and thus has rank at least $n - 1$.

## 8.7 Mortal and Avoiding Words

The techniques described in this chapter can be easily adapted to get the same inapproximability for the Short Mortal Word problem defined as follows.

> Short Mortal Word
> *Input*: A partial automaton $A$ with at least one undefined transition;
> *Output*: The length of a shortest mortal word for $A$.

This problem is connected for instance to the famous Restivo's conjecture [87].

**Theorem 116.** *Unless $P = NP$, the* Short Mortal Word *problem cannot be approximated in polynomial time within a factor of*

*(i) $n^{1-\varepsilon}$ for every $\varepsilon > 0$ for $n$-state binary strongly connected partial automata;*

*(ii) $c \log n$ for some $c > 0$ for $n$-state binary partial Huffman decoders.*

*Proof.* It can be seen that in Theorem 101 and Corollary 108 we construct an automaton with a state $s$ such that each state has to visit $s$ before synchronization. Introduce a new state $s'$ having all the transitions the same as $s$, and for $s$ set the only defined transition (for an arbitrary letter) to map to $s'$. Then it is easy to see that the constructions work for mortal words instead of synchronizing. □

Moreover, it is easy to get a $O(\log n)$-approximation polynomial time algorithm for Short Mortal Word for literal Huffman decoders following the idea of Theorem 113. Indeed, it follows from Lemma 112 that either there exists a mortal word of length at most $\lceil \log_k n \rceil$, or there exists a word $w$ of rank at most $\lceil \log_k n \rceil$. In the latter case we can find a word which is a concatenation of $w$ and a shortest word mapping all this states to nowhere one by one. By the arguments similar to the proof of Theorem 113 we then get the following.

**Proposition 117.** *There exists a $O(\log n)$-approximation polynomial time algorithm for the* Short Mortal Word *problem for $n$-state literal Huffman decoders. This algorithm always finds a mortal word of length $O(n \log n)$.*

Another connected and important problem is to find a shortest avoiding word. Given an automaton $A = (Q, \Sigma, \delta)$, a word $w$ is called *avoiding* for a state $q \in Q$ if $q$ is not contained in the image of $Q$, that is, $q \notin \delta(Q, w)$. Avoiding words play an important role in the recent improvement on the upper bound on the length of a shortest synchronizing word [106]. They are in some sense dual to synchronizing words.

> Short Avoiding word
> *Input*: An automaton $A$ and its state $q$ admitting a word avoiding $q$;
> *Output*: The length of a shortest word avoiding $q$ in $A$.

If $q$ is not the root of a literal Huffman decoder $A$ (that is, not the state corresponding to the empty prefix), then a shortest avoiding word consists of just one letter. So avoiding is non-trivial only for the root state.

**Proposition 118.** *For every $\varepsilon > 0$, there exists a $(1+\varepsilon)$-approximation $O(n^{\log n})$-time algorithm for the problem* Short Sync Word *for $n$-state literal Huffman decoders.*

*Proof.* We use the same algorithm as in the proof of Theorem 114. The only difference is that we check whether the words are avoiding instead of synchronizing. □

## 8.8 Concluding Remarks

For prefix codes, a synchronizing word is usually required to map all the states to the root [14]. One can see that this property holds for all the constructions of the chapter. Moreover, in all the constructions the length of a shortest synchronizing word is linear in the number of states of the automaton. Thus, if we restrict to this case, we still get the same inapproximability results. Also, it should be noted that all the inapproximability results are proved by providing a gap-preserving reduction, thus proving NP-hardness of approximating the Short Sync Word problem within a given factor.

# Conclusions and Further Directions

# Chapter 9

# Conclusions and Further Directions

While finite automata are used extensively as a tool in coding theory, we believe that the connection between the domains of synchronizing automata and coding theory is not as strong as it can be. In particular, there is a number of natural questions in synchronizing automata which have not been studied much for their codes counterparts, and vice versa. We describe some open questions which are relevant to (and sometimes implied by) the results of this thesis.

As shown in this thesis, one can go from the classical and most widely studied setting of complete DFAs to partial DFAs and then to unambiguous NFAs, and still build a theory of synchronizing automata with very similar properties. The only requirement is to ask the automata to be strongly connected, which is a very natural restriction.

In this strongly connected case, we obtained upper bounds of $O(n^2)$ (Theorem 39) and $n^5$ (Theorem 50) for the reset threshold of partial DFAs and unambiguous NFAs respectively. While the bound for partial DFAs cannot be asymptotically improved without progress in the original Černý conjecture, the case of unambiguous NFAs looks much more promising for further progress. As for the lower bounds, the Černý automata have the largest known reset threshold even in the class of strongly connected unambiguous NFAs. It would be very interesting to check that experimentally for unambiguous automata with few states, as well as to try to find other series of non-deterministic unambiguous automata with large reset threshold. To the best of our knowledge, all known non-trivial slowly synchronizing series are deterministic.

As shown by Theorem 50, the rank conjecture is also meaningful for strongly connected unambiguous NFAs. However, in view of the results of Chapter 3, we believe that the strongly connected case of this conjecture should be made more precise. Bounds on the length of words of minimum non-zero rank can be also asked for finite codes and their subfamilies, both in terms of the total length of all codewords (Černý-type questions) and in terms of the length of the longest codeword (Restivo-type questions).

The tight connection with codes naturally motivates the cases of automata recognizing the star of a finite code. This includes both arbitrary such automata and prefix automata of finite codes. For prefix automata of finite prefix codes, the almost-linear upper bound matches the best known result for maximal prefix codes (Corollary 45), but the bound for general finite codes (Theorem 53) is quite large.

Biskup and Plandowski propose a synchronizing counterpart to the Restivo's conjecture [17]. It stays that for a synchronizing finite maximal prefix code $X$ there always exists a synchronizing word of length $\mathcal{O}(k^2)$, where $k$ is the length of the longest codeword in $X$. This conjecture remains open, and obviously the same question may be asked for general finite codes instead of maximal prefix ones.

Finally, strongly connected unambiguous NFAs may not be the most general case for the Černý-type conjectures. It would be interesting to find classes of ambiguous NFAs with good synchronization properties.

The computational aspect of the mentioned questions is also not fully understood. Checking whether a strongly connected partial DFA is synchronizing can be done in quadratic time (Theorem 37). It would be interesting though to improve this time to sub-quadratic, or to find a conditional lower bound. The case of strongly connected unambiguous NFAs allows an $\mathcal{O}(n^5)$-time algorithm to check whether it is synchronizing (Theorem 50). Any improvement to that should provide new insights into the structure of such automata and might help to get better upper bounds for the reset threshold.

Chapters 6 and 7 show that synchronization of a subset of states in a non-synchronizing complete DFA is usually hard. This includes finding a synchronizable set of maximum cardinality (Section 6.3), computing the rank of a subset (Theorem 6.4), and so on. The only known class where many such problems are solvable in polynomial time is the class of monotonic automata (Section 7.2). It would be thus interesting to find other classes which allow easy synchronization in such extended settings. One candidate for that is the class of literal automata of prefix codes, where finding a relatively short synchronizing word is easy (Theorem 113). A more general question can be asked: provided a finite synchronizing code, how hard is it to approximate the length of a shortest synchronizing word for it? Chapter 8 shows that the way how the code is defined matters a lot: this problem is difficult for finite prefix codes defined by arbitrary DFAs (Corollary 111), but is much easier for prefix automata (Theorem 113). More precisely, Conjecture 115 stays that for the literal automaton of a maximal prefix code the length of the shortest synchronizing word can be found in polynomial time.

# Bibliography


[1] Luca Aceto, Augusto Burgueño, and Kim Guldstrand Larsen. Model checking via reachability testing for timed automata. In Bernhard Steffen, editor, *TACAS 1998, LNCS vol. 1384*, pages 263–280, 1998. 14

[2] Jorge Almeida and Benjamin Steinberg. Matrix mortality and the Černý-Pin conjecture. In Volker Diekert and Dirk Nowotka, editors, *DLT 2009. LNCS, vol. 5583*, pages 67–80, 2009. 26, 60

[3] Dimitry S. Ananichev and Mikhail V. Volkov. Synchronizing monotonic automata. *Theoretical Computer Science*, 327(3):225–239, 2004. 20, 60, 74

[4] Dmitry S. Ananichev. The mortality threshold for partially monotonic automata. In Clelia De Felice and Antonio Restivo, editors, *DLT 2005. LNCS, vol. 3572*, pages 112–121. Springer, Berlin, Heidelberg, 2005. 20

[5] Dmitry S. Ananichev, Mikhail V. Volkov, and Vladimir V. Gusev. Primitive digraphs with large exponents and slowly synchronizing automata. *Journal of Mathematical Sciences*, 192(3):263–278, 2013. 26, 29, 30, 57, 82, 87

[6] Parvaneh Babari, Karin Quaas, and Mahsa Shirmohammadi. Synchronizing Data Words for Register Automata. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *MFCS 2016*, volume 58 of *LIPIcs*, pages 15:1–15:15, 2016. 15

[7] Marie-Pierre Béal, Eugen Czeizler, Jarkko Kari, and Dominique Perrin. Unambiguous automata. *Mathematics in Computer Science*, 1(4):625–638, 2008. 15, 17

[8] Marie-Pierre Béal and Dominique Perrin. A quadratic algorithm for road coloring. *Discrete Applied Mathematics*, 169:15–29, 2014. 33, 34

[9] Jordan Bell and Brett Stevens. A survey of known results and research areas for N-queens. *Discrete Mathematics*, 309(1):1 – 31, 2009. 62

[10] Mikhail V. Berlinkov. Approximating the minimum length of synchronizing words is hard. *Theory of Computing Systems*, 54(2):211–223, 2014. 23

[11] Mikhail V. Berlinkov. On two algorithmic problems about synchronizing automata. In Arseny M. Shur and Mikhail V. Volkov, editors, *DLT 2014. LNCS, vol. 8633*, pages 61–67. Springer, Cham, 2014. 16, 20, 24, 28, 35, 39, 83, 84

[12] Mikhail V. Berlinkov and Marek Szykuła. Algebraic synchronization criterion and computing reset words. *Information Sciences*, 369:718–730, 2016. 42, 43, 44, 45, 53, 89

[13] Abraham Berman and Robert J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 1994. 27

[14] Jean Berstel, Dominique Perrin, and Christophe Reutenauer. *Codes and Automata*. Encyclopedia of Mathematics and its Applications 129. Cambridge University Press, 2010. 12, 15, 18, 19, 23, 37, 50, 57, 86, 91

[15] Marek Biskup. *Error Resilience in Compressed Data – Selected Topics*. PhD thesis, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, 2008. 12

[16] Marek Biskup. A word that does not appear in encoded message as a resynchronization marker. In *2008 IEEE Information Theory Workshop, ITW 2008, Porto, Portugal, May 5-9, 2008*, pages 199–203. IEEE, 2008. 13, 14

[17] Marek Biskup and Wojciech Plandowski. Shortest synchronizing strings for Huffman codes. *Theoretical Computer Science*, 410(38-40):3925–3941, 2009. 45, 53, 93

[18] Michael Blondin, Andreas Krebs, and Pierre McKenzie. The complexity of intersecting finite automata having few final states. *Computational complexity*, 25(4):775–814, 2016. 79

[19] Eugenija A. Bondar and Mikhail V. Volkov. Completely reachable automata. In Cezar Câmpeanu, Florin Manea, and Jeffrey Shallit, editors, *DCFS 2016. LNCS, vol. 9777*, pages 1–17. Springer, Cham, 2016. 71

[20] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In Antoni W. Mazurkiewicz and Józef Winkowski, editors, *CONCUR 1997, LNCS vol. 1243*, pages 135–150, 1997. 14

[21] Manfred Broy, Bengt Jonsson, J-P Katoen, Martin Leucker, and Alexander Pretschner, editors. *Model-based testing of reactive systems*. LNCS vol. 3472. Springer, 2005. 9

[22] Veronique Bruyère. On maximal codes with bounded synchronization delay. *Theoretical Computer Science*, 204(1):11–28, 1998. 18, 87

[23] Janusz A. Brzozowski and Faith E. Fich. Languages of R-trivial monoids. *Journal of Computer and System Sciences*, 20(1):32–49, 1980. 20

[24] Angela Cardoso. *The Černý Conjecture and Other Synchronization Problems*. PhD thesis, University of Porto, Portugal, 2014. 21

[25] Arturo Carpi. On synchronizing unambiguous automata. *Theoretical Computer Science*, 60(3):285 – 296, 1988. 22, 50, 51, 54

[26] Arturo Carpi and Flavio D'Alessandro. Strongly transitive automata and the Černý conjecture. *Acta Informatica*, 46(8):591–607, 2009. 32

[27] Arturo Carpi and Flavio D'Alessandro. On the Hybrid Černý-Road Coloring Problem and Hamiltonian paths. In Yuan Gao, Hanlin Lu, Shinnosuke Seki, and Sheng Yu, editors, *DLT 2010. LNCS, vol. 6224*, pages 124–135, 2010. 29

[28] Jan Černý. Poznámka k homogénnym eksperimentom s konečnými automatami. *Matematicko-fyzikálny Časopis Slovenskej Akadémie Vied*, 14(3):208–216, 1964. In Slovak. 15, 22, 42

[29] Dmitry Chistikov, Pavel V. Martyugin, and Mahsa Shirmohammadi. Synchronizing automata over nested words. *Journal of Automata, Languages and Combinatorics*, 24(2-4):219–251, 2019. 15

[30] Richard Cole, Kirstin Ost, and Stefan Schirra. Edge-coloring bipartite multigraphs in $O(E \log D)$ time. *Combinatorica*, 21(1):5–12, 2001. 33

[31] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009. 20, 73

[32] Michiel de Bondt, Henk Don, and Hans Zantema. Lower bounds for synchronizing word lengths in partial automata. *International Journal of Foundations of Computer Science*, 30(1):29–60, 2019. 22

[33] Laurent Doyen, Line Juhl, Kim Guldstrand Larsen, Nicolas Markey, and Mahsa Shirmohammadi. Synchronizing words for weighted and timed automata. In Venkatesh Raman and S. P. Suresh, editors, *FSTTCS 2014*, volume 29 of *LIPIcs*, pages 121–132, 2014. 15

[34] Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. Robust synchronization in Markov decision processes. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014, LNCS vol. 8704*, pages 234–248, 2014. 15

[35] Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. The complexity of synchronizing Markov decision processes. *Journal of Computer and System Sciences*, 100:96–129, 2019. 15

[36] L. Dubuc. Sur les automates circulaires et la conjecture de Černý. *RAIRO – Theoretical Informatics and Applications*, 32(1-3):21–34, 1998. 33

[37] David Eppstein. Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19(3):500–510, 1990. 11, 15, 20, 21, 23, 41, 42, 68, 74, 78, 85

[38] Henning Fernau and Andreas Krebs. Problems on finite automata and the Exponential Time Hypothesis. In Yo-Sub Han and Kai Salomaa, editors, *CIAA 2016, LNCS vol. 9705*, pages 89–100, 2016. 79

[39] Gabriele Fici, Elena V. Pribavkina, and Jacques Sakarovitch. On the minimal uncompletable word problem. *CoRR*, abs/1002.1928, 2010. 23

[40] N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. 57

[41] Christopher F Freiling, Douglas S Jungreis, François Théberge, and Kenneth Zeger. Almost all complete binary prefix codes have a self-synchronizing string. *IEEE Transactions on Information Theory*, 49(9):2219–2225, 2003. 12

[42] Joel Friedman. On the road coloring problem. *Proceedings of the American Mathematical Society*, 110:1133–1135, 1990. 66

[43] Paweł Gawrychowski and Damian Straszak. Strong inapproximability of the shortest reset word. In F. Giuseppe Italiano, Giovanni Pighizzini, and T. Donald Sannella, editors, *MFCS 2015. LNCS, vol. 9234*, pages 243–255, 2015. 23, 62, 83

[44] Michael Gerbush and Brent Heeringa. Approximating minimum reset sequences. In Michael Domaratzki and Kai Salomaa, editors, *CIAA 2010, LNCS vol. 6482*, pages 154–162, 2011. 24

[45] Balázs Gerencsér, Vladimir V. Gusev, and Raphaël M. Jungers. Primitive sets of nonnegative matrices and synchronizing automata. *SIAM Journal on Matrix Analysis and Applications*, 39(1):83–98, 2018. 71

[46] Vladimir V. Gusev. Lower bounds for the length of reset words in eulerian automata. *International Journal of Foundations of Computer Science*, 24(2):251–262, 2013. 29, 32

[47] Vladimir V. Gusev and Elena V. Pribavkina. On non-complete sets and Restivo's conjecture. In Giancarlo Mauri and Alberto Leporati, editors, *DLT 2011, LNCS vol. 6795*, pages 239–250, 2011. 23

[48] Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR 2009, LNCS vol. 5710*, pages 369–383, 2009. 14

[49] Tero Harju and Dirk Nowotka. On unique factorizations of primitive words. *Theoretical Computer Science*, 356(1-2):186–189, 2006. 45

[50] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001. 80

[51] B. R. Heap and M. S. Lynn. The structure of powers of nonnegative matrices: I. the index of convergence. *SIAM Journal on Applied Mathematics*, 14(3):610–639, 1966. 29

[52] Markus Holzer and Martin Kutrib. Descriptional and computational complexity of finite automata – a survey. *Information and Computation*, 209(3):456–470, 2011. 79

[53] John Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Zvi Kohavi and Azaria Paz, editors, *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971. 42

[54] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952. 11

[55] Roy Hunter and A Harry Robinson. International digital facsimile coding standards. *Proceedings of the IEEE*, 68(7):854–867, 1980. 14

[56] Balázs Imreh and Magnus Steinby. Directable nondeterministic automata. *Acta Cybernetica*, 14(1):105–115, 1999. 15, 22

[57] Galina Jirásková and Tomáš Masopust. On the state and computational complexity of the reverse of acyclic minimal DFAs. In Nelma Moreira and Rogério Reis, editors, *CIAA 2012. LNCS, vol. 7381*, pages 229–239. Springer, Heidelberg, 2012. 20

[58] Juhani Karhumäki, Ján Manuch, and Wojciech Plandowski. A defect theorem for bi-infinite words. *Theoretical Computer Science*, 292(1):237–243, 2003. 58

[59] Jarkko Kari. A counter example to a conjecture concerning synchronizing words in finite automata. *Bulletin of the EATCS*, 73:146, 2001. 26

[60] Jarkko Kari. Synchronizing finite automata on Eulerian digraphs. *Theoretical Computer Science*, 295(1):223–232, 2003. 31, 33, 34, 43, 66

[61] Jarkko Kari, Andrew Ryzhikov, and Anton Varonka. Words of minimum rank in deterministic finite automata. In *DLT 2019, LNCS vol. 11647*, pages 74–87, 2019. 24, 40

[62] Stefan Kiefer and Corto Mascle. On Finite Monoids over Nonnegative Integer Matrices and Short Killing Words. In Rolf Niedermeier and Christophe Paul, editors, *STACS 2019*, volume 126 of *LIPIcs*, pages 43:1–43:13, 2019. 22, 50, 51, 52, 53, 54

[63] A. A. Klyachko, I. K. Rystsov, and M. A. Spivak. In extremal combinatorial problem associated with the bound on the length of a synchronizing word in an automaton. *Cybernetics*, 23(2):165–171, 1987. 26, 30

[64] Eryk Kopczyński. Half-positional determinacy of infinite games. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP 2006. LNCS, vol. 4052*, pages 336–347, 2006. 20

[65] Eryk Kopczyński. *Half-Positional Determinacy of Infinite Games*. PhD thesis, Warsaw University, Poland, 2008. 20

[66] Dexter Kozen. Lower Bounds for Natural Proof Systems. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS'77, pages 254–266, 1977. 70, 79

[67] Wai-Man Lam and Sanjeev R Kulkarni. Extended synchronizing codewords for binary prefix codes. *IEEE Transactions on Information Theory*, 42(3):984–987, 1996. 14

[68] Kim Guldstrand Larsen, Simon Laursen, and Jiří Srba. Synchronizing Strategies under Partial Observability. In P. Baldan and D. Gorla, editors, *CONCUR 2014. LNCS, vol. 8704*, pages 188–202. Springer, 2014. 15

[69] S-M Lei and M-T Sun. An entropy coding system for digital HDTV applications. *IEEE transactions on circuits and systems for video technology*, 1(1):147–155, 1991. 14

[70] Pavel V. Martugin. A series of slowly synchronizing automata with a zero state over a small alphabet. *Information and Computation*, 206(9):1197 – 1203, 2008. 54

[71] Pavel V. Martyugin. Complexity of problems concerning carefully synchronizing words for PFA and directing words for NFA. In Farid Ablayev and Ernst W. Mayr, editors, *CSR 2010. LNCS, vol. 6072*, pages 288–302. Springer, Heidelberg, 2010. 22, 78

[72] Pavel V. Martyugin. A lower bound for the length of the shortest carefully synchronizing words. *Russian Mathematics*, 54(1):46–54, 2010. 22

[73] Pavel V. Martyugin. Complexity of problems concerning reset words for cyclic and Eulerian automata. *Theoretical Computer Science*, 450(Supplement C):3 – 9, 2012. 23

[74] Ernst W Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal on Computing*, 13(3):441–460, 1984. 14

[75] Maksymilian Mika and Marek Szykuła. Complexity of the Frobenius monoid problem for a finite language. *CoRR*, abs/1902.06702, 2019. 14, 23

[76] Clemens Müllner and Andrew Ryzhikov. Palindromic subsequences in finite words. In Carlos Martín-Vide, Alexander Okhotin, and Dana Shapira, editors, *LATA 2019, LNCS vol. 11417*, pages 460–468, 2019. 24

[77] Balas K. Natarajan. An algorithmic approach to the automated design of parts orienters. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pages 132–142. IEEE Computer Society, 1986. 11, 15, 21

[78] Balas K. Natarajan. Some paradigms for the automated design of parts feeders. *The International Journal of Robotics Research*, 8(6):98–109, 1989. 11

[79] Jean Néraud and Carla Selmi. On codes with a finite deciphering delay: constructing uncompletable words. *Theoretical Computer Science*, 255(1):151 – 162, 2001. 54

[80] Michael S Paterson. Unsolvability in 3×3 matrices. *Studies in Applied Mathematics*, 49(1):105–107, 1970. 14

[81] Dominique Perrin and Andrew Ryzhikov. The degree of a finite set of words. *To appear in FSTTCS 2020*. 24

[82] Jean-Eric Pin. Utilisation de l'algèbre linéaire en théorie des automates. In *Actes du 1er Colloque AFCET-SMF de Mathématiques Appliquées II*, AFCET, pages 85–92, 1978. In French. 43

[83] Jean-Eric Pin. Le problème de la synchronisation et la conjecture de Cerný. In A. De Luca, editor, *Non-commutative structures in algebra and geometric combinatorics vol. 109*, Quaderni de la Ricerca Scientifica, pages 37–48. CNR (Consiglio nazionale delle ricerche, Italy), 1981. In French. 26

[84] Jean-Eric Pin. On two combinatorial problems arising from automata theory. In P. Camion J.F. Maurras C. Berge, D. Bresson and F. Sterboul, editors, *Combinatorial Mathematics Proceedings of the International Colloquium on Graph Theory and Combinatorics*, volume 75 of *North-Holland Mathematics Studies*, pages 535–548. North-Holland, 1983. 26, 30, 39, 43

[85] Maurice Pouzet, Robert Woodrow, and Nejib Zaguia. Generating boxes from ordered sets and graphs. *Order*, 9(2):111–126, 1992. 51

[86] Elena V. Pribavkina. Slowly synchronizing automata with zero and noncomplete sets. *Mathematical Notes*, 90(3):422–430, 2011. 23, 54, 56

[87] Antonio Restivo. Some remarks on complete subsets of a free monoid. *Quaderni de La ricerca scientifica, CNR Roma*, 109:19–25, 1981. 23, 90

[88] Igor K. Rystsov. Asymptotic estimate of the length of a diagnostic word for a finite automaton. *Cybernetics*, 16(2):194–198, 1980. 22, 78

[89] Igor K. Rystsov. Polynomial complete problems in automata theory. *Information Processing Letters*, 16(3):147–151, 1983. 21, 67

[90] Igor K. Rystsov. Rank of a finite automaton. *Cybernetics and Systems Analysis*, 28(3):323–328, 1992. 66, 67

[91] Igor K. Rystsov. Reset words for commutative and solvable automata. *Theoretical Computer Science*, 172(1):273–279, 1997. 48, 49, 54, 60

[92] Andrew Ryzhikov. Synchronization problems in automata without non-trivial cycles. In Arnaud Carayol and Cyril Nicaud, editors, *CIAA 2017. LNCS, vol. 10329*, pages 188–200, 2017. 24, 61

[93] Andrew Ryzhikov. Mortality and synchronization of unambiguous finite automata. In Robert Mercaş and Daniel Reidenbach, editors, *WORDS 2019, LNCS vol. 11682*, pages 299–311, 2019. 24

[94] Andrew Ryzhikov. On automata recognizing birecurrent sets. *Theoretical Computer Science*, 753:76–79, 2019. 24

[95] Andrew Ryzhikov. Synchronization problems in automata without non-trivial cycles. *Theoretical Computer Science*, 787:77–88, 2019. 24

[96] Andrew Ryzhikov and Anton Shemyakov. Subset synchronization in monotonic automata. In Juhani Karhumäki and Aleksi Saarela, editors, *Proceedings of the Fourth Russian Finnish Symposium on Discrete Mathematics, TUCS Lecture Notes 26*, pages 154–164, 2017. 24

[97] Andrew Ryzhikov and Anton Shemyakov. Subset synchronization in monotonic automata. *Fundamenta Informaticae*, 162(2-3):205–221, 2018. 24

[98] Andrew Ryzhikov and Marek Szykuła. Finding Short Synchronizing Words for Prefix Codes. In Igor Potapov, Paul Spirakis, and James Worrell, editors, *MFCS 2018*, volume 117 of *LIPIcs*, pages 21:1–21:14, 2018. 24, 45

[99] George S. Sacerdote and Richard L. Tenney. The decidability of the reachability problem for vector addition systems (preliminary version). In John E. Hopcroft, Emily P. Friedman, and Michael A. Harrison, editors, *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 61–76. ACM, 1977. 14

[100] Sven Sandberg. Homing and synchronizing sequences. In Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors, *Model-Based Testing of Reactive Systems: Advanced Lectures. LNCS, vol. 3472*, pages 5–33. 2005. 15, 21, 62

[101] Marcel-Paul Schützenberger. On synchronizing prefix codes. *Information and Control*, 11(4):396 – 401, 1967. 12

[102] Tamara Shcherbak. The interval rank of monotonic automata. In Jacques Farré, Igor Litovsky, and Sylvain Schmitz, editors, *CIAA 2005. LNCS, vol. 3845*, pages 273–281. 2006. 20

[103] Yaroslav Shitov. An Improvement to a Recent Upper Bound for Synchronizing Words of Finite Automata. *Journal of Automata, Languages and Combinatorics*, 24(2–4):367–373, 2019. 14, 22, 42, 43, 47, 48

[104] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 3rd edition, 2012. 23, 68, 78

[105] Marek Szykuła. Checking whether an automaton is monotonic is NP-complete. In Frank Drewes, editor, *CIAA 2015. LNCS, vol. 9223*, pages 279–291. 2015. 20, 21

[106] Marek Szykuła. Improving the Upper Bound on the Length of the Shortest Reset Word. In *STACS 2018*, LIPIcs, pages 56:1–56:13. 2018. 14, 22, 42, 43, 47, 48, 90

[107] Marek Szykuła and Vojtěch Vorel. An extremal series of Eulerian synchronizing automata. In Srečko Brlek and Christophe Reutenauer, editors, *DLT 2016. LNCS, vol. 9840*, pages 380–392, 2016. 32

[108] Avraham N. Trahtman. The Cerný conjecture for aperiodic automata. *Discrete Mathematics & Theoretical Computer Science*, 9(2), 2007. 20, 60

[109] Nicholas F. Travers and James P. Crutchfield. Exact Synchronization for Finite-State Sources. *Journal of Statistical Physics*, 145(5):1181–1201, 2011. 16, 37, 41

[110] Luca Trevisan. Notes on state minimization. https://people.eecs.berkeley.edu/~luca/cs172-07/notemindfa.pdf, 2007. 37

[111] Uraz Cengiz Türker and Hüsnü Yenigün. Complexities of some problems related to synchronizing, non-synchronizing and monotonic automata. *International Journal of Foundations of Computer Science*, 26(01):99–121, 2015. 62

[112] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, 2001. 23

[113] Mikhail V. Volkov. Synchronizing automata and the Černý conjecture. In Carlos Martín-Vide, Friedrich Otto, and Henning Fernau, editors, *LATA 2008. LNCS, vol. 5196*, pages 11–27, 2008. 11, 14, 15, 16, 20, 21, 22, 56, 73

[114] Vojtěch Vorel. Subset synchronization and careful synchronization of binary finite automata. *International Journal of Foundations of Computer Science*, 27(5):557–577, 2016. 21, 22, 28, 63, 65, 68

[115] Vojtch Vorel. Complexity of a problem concerning reset words for Eulerian binary automata. *Information and Computation*, 253(Part 3):497 – 509, 2017. 23

[116] Michael Wehar. Hardness results for intersection non-emptiness. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP 2014, LNCS vol. 8573*, pages 354–362, 2014. 79

[117] C.M. Weinbaum. Unique subwords in nonperiodic words. *Proceedings of the American Mathematical Society*, 109(3):615–619, 1990. 45

[118] Helmut Wielandt. Unzerlegbare, nicht negative Matrizen. *Mathematische Zeitschrift*, 52(1):642–648, 1950. 29

[119] Petra Wolf. Synchronization under dynamic constraints. *To appear in FSTTCS 2020*, CoRR, abs/1910.01935, 2019. 11

[120] James Worrell. Reachability problems for continuous linear dynamical systems (invited paper). In Luca Aceto and David de Frutos-Escrig, editors, *CONCUR 2015*, volume 42 of *LIPIcs*, pages 17–17, 2015. 14

[121] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(6):103–128, 2007. 63, 67