Lecture Notes on Synchronizing Automata

Mikhail V. Volkov

April 16, 2016

Since the text is still unstable, please mention not only the line number but also the version date when sending me a correction!

Contents

Introduction

Aim and Scope

These notes are based on the lecture course that the author presented at Hunter College of the City University of New York in the spring semester of the academic year 2015/16. They intend to quickly introduce the area to a freshman with a minimum background but at the same time to provide a survey of the present state-of-the-art that might be useful for specialists. This twofold intention reflects in the 2-layer structure of the text: almost all sections contain supplements (typeset in a smaller font) that collect, along with various comments on the core material of the corresponding section, an account of related recent developments and open issues. A freshman should feel free to skip these supplements, at least at the first reading, while a specialist may read them in grasshopper's mode looking for a discussion of some topics of his/her particular interest.

Overview

Chapter ch:history 1 introduces the central concept of these lecture notes—synchronizing automata—and briefly discusses its history, including its frequent rediscoveries in various branches of mathematics, computer science and engineering. Chapter ch:algorithms 2 presents two important polynomial algorithms: for recognizing synchronizability and for finding a reset word. Chapter ch:complexity 3 gives an account of the research on computational complexity of several problems related to synchronizing automata. In Chapter ch:cerny_conjecture 4 we introduce and discuss the Černý conjecture, the main open problem in the area. Chapter ch:road_coloring 5 presents a solution to the Road Coloring Problem, another major problem related to synchronizing automata.

Prerequisites

Exercises

Most sections are provided with exercises. Each exercise is labeled by one of the three labels: easy, medium, or difficult. Easy exercises are indeed very simple questions provided mainly for self-control. Medium level exercises may require some calculation and/or an application of a result shown in the corresponding section. Difficult exercises may require a somewhat lengthy calculation or a non-trivial idea, but in most cases they are difficult from the viewpoint of the current section only—when the reader gets acquainted with further results and algorithms, previously 'difficult' exercises will become easy for him/her. Hints for some exercises can be found at the end of the lecture notes.

Disclaimers and Acknowledgements

The author is grateful to Gregory Javens for providing solutions to a majority of exercises and helping in eliminating several typos.

Chapter 1

History and Motivation

ch:history

1.1 Finite Automata

sec:dfa

A *finite automaton* is a simple but extremely productive concept that captures the very important idea of an object interacting with its environment. This notion originates in the seminal work [38] by Alan Turing published in 1936. Here is a quotation from this paper.

> "Computing is normally done by writing certain symbols on paper. We may suppose that this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that computation is carried out on one-dimensional paper, i.e. on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite. If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrarily small extent. The effect of this restriction of the number of symbols is not very serious. It is always possible to use sequences of symbols in place of single symbols. [...] The behaviour of the computer at any moment is determined by the symbols which he is observing, and the 'state of mind' at that moment. We may suppose that there is a bound to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use

> successive observations. We will also suppose that the number of states of mind which need to be taken into account is finite. The reason for this are of the same character as those which restrict the number of symbols. If we admitted an infinity of states of mind, some of them will be 'arbitrarily close' and will be confused. Again, the restriction is not one which seriously affects computation, since the use of more complicated states of mind can be avoided by writing more symbols on the tape. Let us imagine the operations performed by the computer to be split up into 'simple operations' which are so elementary that it is not easy to imagine them further divided. Every such operation consists of some change of the physical system consisting of the computer and his tape. We know the state of the system if we know the sequence of symbols on the tape, which of these are observed by the computer (possibly with a special order), and the state of mind of the computer."

We see that a direct yet careful analysis of the process of computation inevitably reveals three characteristic features underlying the idea of a finite automaton: finiteness of the set of input symbols, finiteness of the set of inner states ('states of mind' in Turing's terminology), and determinism, meaning that the next state on each step is determined by the current state and the current input symbol(s).

Yet another early source that should be mentioned when one speaks about the origin of the concept of a finite automaton is the paper [24] by Warren McCulloch and Walter Pitts published in 1943. And the 'official' year of birth of finite automata in the form in which we use them since then is 1956 when the book [35] appeared, being a collection of influential papers that laid a firm foundation for the future developments of automata theory.

There are many species of finite automata; in these lecture notes we mainly deal with the simplest version: complete deterministic finite automata. A *complete deterministic finite automaton* (a DFA, for short) is a triple $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, where Q is the *state set*, Σ is the *input alphabet*, and $\delta \colon Q \times \Sigma \to Q$ is the *transition function*. The elements of Q and Σ are referred to as *states* and (*input*) *letters* respectively. We imagine that the DFA evolves in discrete time; in each moment of time it is in a certain state $q \in Q$. During the next time unit, exactly one input letter $a \in \Sigma$ arrives, making the automaton transit to the state $\delta(q, a)$.

Let us comment on the meanings of the three adjectives (complete, deterministic, and finite) in the above definition. The attribute 'finite' refers to the fact that both the state set and the input alphabet are always supposed to be finite; we also tacitly assume that both these sets are non-empty. The attribute 'deterministic' means that the state $\delta(q,a)$ is uniquely determined by the pair $(q,a) \in Q \times \Sigma$ so that the transition rule is indeed a function. The attribute 'complete' emphasizes that the transition function is assumed to be totally defined: $\delta(q,a)$ must exist for each pair $(q,a) \in Q \times \Sigma$.

Further in these lecture notes we shall encounter some other sorts of finite automata. E.g., we shall meet *nondeterministic automata* whose transition rules form a relation between the state set and the input alphabet rather than a function; *partial automata* whose transition function may be undefined at some pairs (state, input letter); *automata with output* which are additionally equipped with a function assigning to each pair (state, input letter) a symbol in some output alphabet, etc. As we do not want to overload this introductory section with a variety of concepts, we postpone the corresponding definitions until they become necessary. The same tactic is used with respect to the notions of *initial* and *final* states of an automaton. These concepts are of crucial importance if an automaton is considered as an acceptor but they are not immediately required in these lecture notes, and therefore, we shall not recall them right now.

Finite automata admit a very convenient visual representation as labeled graphs. We assume that the reader is acquainted with the idea of a graph; still we should explain which graphs we are going to use. Here a *graph* means a quadruple of sets and maps: the set of *vertices* V, the set of *edges* E, a map $h\colon E \to V$ that maps every edge to its *head* vertex and a map $t\colon E \to V$ that maps every edge to its *tail* vertex. Notice that in a graph, several edges may share the same tail and head; such edges are called *parallel*. Loops are also allowed—a *loop* is an edge whose tail and head coincide. (Thus, our graphs are in fact directed multigraphs but since no other graph species show up in these lecture notes, we use a short name.) A *labeled graph* is a graph equipped with an extra map $\lambda\colon E \to \Lambda$ that maps every edge to its *label*, where Λ is a set called the *label alphabet*. We denote an edge with head v, tail v' and label a by $v \xrightarrow{a} v'$.

Now, given a DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, we represent it by the labeled graph with the vertex set Q, the label alphabet Σ, and the set of edges

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

Thus, the transition from q to q' caused by a is represented by the edge with head q, tail q' and label a.

For instance, Figure 1.1 shows the DFA $\mathscr{C}_4$ with four states denoted 0,1,2,3, two input letters a and b, and the transition function

$$\delta(i,a) = \begin{cases} 1 & \text{if } i = 0, \\ i & \text{if } i = 1,2,3; \end{cases} \qquad \delta(i,b) = i+1 \pmod 4 \quad \text{for } i = 0,1,2,3.$$

Here and below we adopt the convention that edges bearing multiple la-

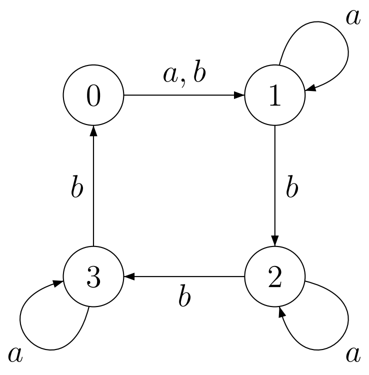

Figure 1.1: The automaton $\mathscr{C}_4$ KV:fig:C4

bels represent bunches of parallel edges. In particular, the edge $0 \xrightarrow{a,b} 1$ in Figure 1.1 represents the two parallel edges $0 \xrightarrow{a} 1$ and $0 \xrightarrow{b} 1$.

Given an alphabet Σ, a *word* over Σ is a finite sequence of letters from Σ. We do not exclude the empty sequence form this definition; that is, we allow the *empty word* which we denote by ε. The set of all words over Σ including ε is denoted by Σ^* and is referred to as the *free monoid over* Σ. Sometimes we will also need the set of all non-empty words over Σ denoted Σ^+. If $w = a_1 \cdots a_\ell$ with $a_1, \dots, a_\ell \in \Sigma$ is a word from Σ^+, the number ℓ is said to be the *length* of w and is denoted by $|w|$. The length of the empty word is defined to be 0.

Given a DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, the transition function δ extends to a function $Q \times \Sigma^* \to Q$ (still denoted by δ) in the following way. For each $q \in Q$, we let $\delta(q,\varepsilon) := q$ and if $w = a_1 \cdots a_\ell$ with $a_1, \dots, a_\ell \in \Sigma$ is a word from Σ^+, we let

$$\delta(q,w) := \delta(\dots\delta(\delta(q,a_1),a_2),\dots,a_\ell).$$

Let $\mathcal{P}(Q)$ stand for the set of all non-empty subsets of the set Q. The function δ can be further extended to a function $\mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$ (again denoted by δ) by letting $\delta(P, w) := \{\delta(q, w) \mid q \in P\}$ for every non-empty subset $P \subseteq Q$.

When we consider a fixed DFA, we often simplify notation by writing $q \,.\, w$ for $\delta(q, w)$ and $P \,.\, w$ for $\delta(P, w)$.

Exercises for Section sec:dfa 1.1

exer:1.1.1 *Exercise* 1.1.1 (easy). Calculate the number of words of length ℓ over the alphabet $\{a_1, \dots, a_k\}$.

exer:1.1.2 *Exercise* 1.1.2 (easy). Calculate the number of DFAs with the state set $\{1, \dots, n\}$ and the input alphabet $\{a_1, \dots, a_k\}$.

In this lecture notes we systematically employ graphical representations of DFAs, following the wisdom "A picture is worth a thousand words". The reader should realize, however, that the pictorial language has some natural limitations: it is hard to expect that a picture could conveniently represent a DFA with, say, 100 states and 50 input letters. Another popular way to represent DFAs utilizes transition tables. Given a DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, its *transition table* is basically a matrix with rows labeled by the input letters and columns labeled by states and whose entry in the a-th row and the q-th column is $\delta(q, a)$ for all $a \in \Sigma$ and $q_i n Q$. For instance, the following is the transition table of the automaton $\mathscr{C}_4$ shown in Figure KV:fig:C4 1.1.

Table 1.1: Transition table of the automaton $\mathscr{C}_4$ tb:transisiton table

	0	1	2	3
a	1	1	2	3
b	1	2	3	0

exer:1.1.3 *Exercise* 1.1.3 (easy). A DFA is given by the following transition table.

	0	1	2	3	4
a	1	2	1	3	4
b	2	1	0	3	4
c	0	4	3	3	1

Represent this DFA by a labeled graph.

1.2 Synchronizing Automata

sec:sa

def:sa&sw

Definition 1.2.1. A complete deterministic finite automaton $\mathscr{A} = \langle Q, \Sigma, \delta\rangle$ is called *synchronizing* if there exists a word $w \in \Sigma^*$ that brings the automaton to one particular state no matter at which state in Q the word w is applied: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$. Any word with this property is said to be a *reset* word for the automaton.

Using the notational convention introduced in Section 1.1, we can alternatively express the definition of a reset word by the equality $|Q\,.\,w| = 1$.

exam:C4

Example 1.2.1. The automaton $\mathscr{C}_4$ presented in Figure 1.1 is synchronizing.

The claim of Example 1.2.1 is not completely trivial: if one just looks at Figure 1.1, it is not easy to guess a reset word for $\mathscr{C}_4$. One of the possible reset words for $\mathscr{C}_4$ is *abbbabbba*: applying it at any state brings the automaton to the state 1. (Later we shall see that *abbbabbba* is in fact a reset word of minimum length for $\mathscr{C}_4$.) This follows from the straightforward computation presented in Table 1.2.

Table 1.2: Step-by-step action of the word *abbbabbba* in the automaton $\mathscr{C}_4$ tb:ab3ab3a

Start state	a	b	b	b	a	b	b	b	a
0	1	2	3	0	1	2	3	0	1
1	1	2	3	0	1	2	3	0	1
2	2	3	0	1	1	2	3	0	1
3	3	0	1	2	2	3	0	1	1

The computation shown in Table 1.2 can be also used to explain the choice of the term 'synchronizing'. Imagine that we have four copies of the automaton $\mathscr{C}_4$, all in different states initially. If we simultaneously apply the word *abbbabbba* to each of the copies, we see that at the end they 'synchronize', that is, they all reach the same state (and since then, their behaviours under simultaneous applications of any further word will remain synchronous).

The reader should be advised that in the literature synchronizing automata and reset words sometimes appear under various alternative names: some authors use adjectives like 'directable', 'cofinal', 'collapsible' for automata and adjectives like 'directing', 'recurrent', 'synchronizing' for words. On the other hand, the term 'synchronization' is often used in a meaning different from ours. Do not get confused by this!

Exercises for Section 1.2

sec:sa

exer:1.2.1 *Exercise* 1.2.1 (easy). Verify that the word *abbababbba* also is a reset word for the automaton $\mathscr{C}_4$.

exer:1.2.2 *Exercise* 1.2.2 (easy). Both the words *abbbabbba* and *abbababbba* reset the automaton $\mathscr{C}_4$ to the state 1, i.e., applying each of these word at any state brings the automaton to the state 1. Is it possible to reset the automaton to the state 2? If so, which word does the job? And what about two other states?

exer:1.2.3 *Exercise* 1.2.3 (easy). Give an example of a DFA which is not synchronizing.

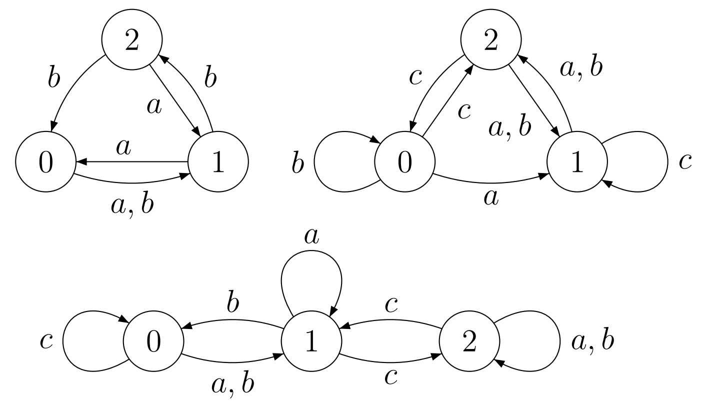

Figure 1.2: Three 3-state automata fig:3sa3

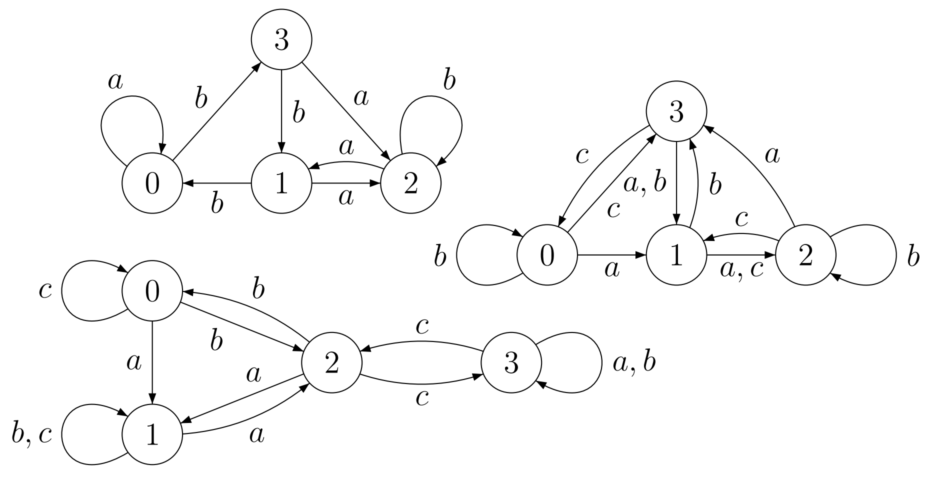

Figure 1.3: Three 4-state automata fig:3sa4

exer:1.2.4 *Exercise* 1.2.4 (medium). Verify that each of the 3-state automata shown in Figure fig:3sa3 1.2 is synchronizing and find a reset word for each of them.

exer:1.2.5 *Exercise* 1.2.5 (medium). Verify that each of the 4-state automata shown in Figure fig:3sa4 1.3 is synchronizing and find a reset word for each of them.

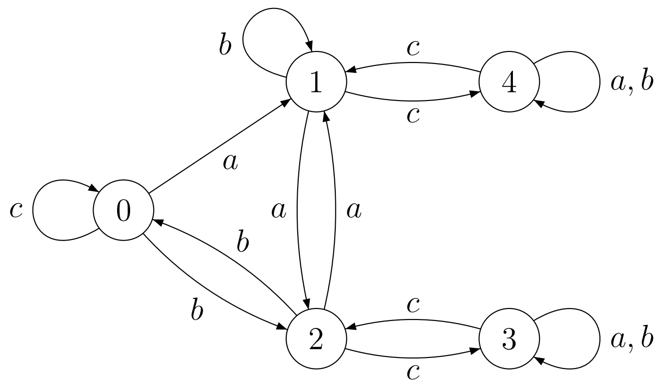

Figure 1.4: The automaton $\mathscr{R}_5$ fig:Roman

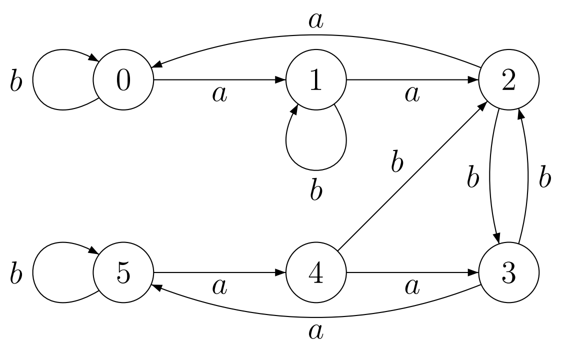

Figure 1.5: The automaton $\mathscr{K}_6$ fig:Kari

exer:1.2.6 *Exercise* 1.2.6 (difficult). Verify that the automaton $\mathscr{R}_5$ shown in Figure fig:Roman 1.4 is synchronizing and find a reset word for it.

exer:1.2.7 *Exercise* 1.2.7 (difficult). Verify that the automaton $\mathscr{K}_6$ shown in Figure fig:Kari 1.5 is synchronizing and find a reset word for it.

Let Σ be an alphabet and let u, v be words over Σ. We denote by uv the word obtained by concatenating u and v, i.e., if $u = a_1 \cdots a_k$, $v = b_1 \cdots b_\ell$ with $a_1, \dots a_k, b_1, \dots b_\ell \in \Sigma$, we have $uv = a_1 \cdots a_k b_1 \cdots b_\ell$. We refer to uv as to the *product* of u with v and say that u and v are *factors* of the product. Observe that $|uv| = |u| + |v|$.

exer:1.2.8 *Exercise* 1.2.8 (easy). Let $\mathscr{A} = \langle Q, \Sigma, \delta\rangle$ be a synchronizing automaton and let $w \in \Sigma^*$ be a reset word for $\mathscr{A}$. Show that for every $v \in \Sigma^*$, the products vw and wv also are reset words for $\mathscr{A}$. In particular, if $\mathscr{A}$ has a reset word of some length ℓ, then $\mathscr{A}$ has a reset word of every length $m \geq \ell$.

An *ideal* of the free monoid Σ^* is a subset of Σ^* that contains every product of which it contains at least one factor. Thus, Exercise exer:1.2.8 1.2.8 tells us that the reset words of a given synchronizing automaton with the input alphabet Σ form a non-empty ideal of Σ^*.

exer:1.2.9 *Exercise* 1.2.9 (difficult). Let Σ be a finite alphabet and I a non-empty ideal of Σ^*. Does there exist a synchronizing automaton with the input alphabet Σ having I as the set of all reset words?

exer:1.2.10 *Exercise* 1.2.10 (medium). Let Σ be a finite alphabet and w a non-empty word over Σ. Does there exist a synchronizing automaton with the input alphabet Σ having w as a reset word?

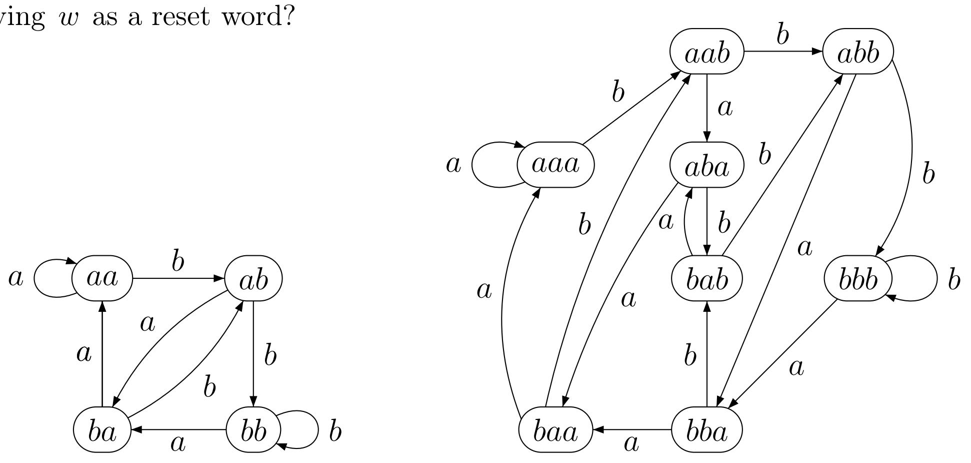

Figure 1.6: De Bruijn's automata of orders 2 and 3 over $\{a, b\}$ fig:DeBr

Let Σ be an alphabet and $n \geq 2$ an integer. We denote by Σ^n the set of all words of length n over Σ. *De Bruijn's automaton of order n over Σ* is the automaton of the form $\langle \Sigma^n, \Sigma, \delta\rangle$, whose transition function is defined as follows: for all $a_1, a_2, \ldots, a_n, b \in \Sigma$,

$$\delta(a_1 a_2 \cdots a_n, b) := a_2 \cdots a_n b.$$

(The action of b on $a_1 a_2 \cdots a_n$ removes the first letter of $a_1 a_2 \cdots a_n$ and appends the letter b.) De Bruijn's automata of orders 2 and 3 over the alphabet $\{a, b\}$ are shown in Figure fig:DeBr 1.6.

exer:1.2.11 *Exercise* 1.2.11 (easy). Show that De Bruijn's automaton of order n over Σ is synchronizing and each word in Σ^n serves as its reset word.

Comments and Supplements for Section 1.2

Example 1.2.1 is borrowed from the paper [7] by Jan Černý. We discuss this seminal paper in more detail in the next section and in Chapters 2 and 4. The 3-state automata of Figure 1.2 are taken from Avraam Trahtman's paper [36]. The first of the 4-state automata shown in Figure 1.3 first appeared in 1971 in [8] while the two others were found 25 years later, in [36]. The automata $\mathscr{R}_5$ and $\mathscr{K}_6$ was discovered by Adam Roman [32] and respectively Jarkko Kari [17]. We will meet the DFAs from Exercises 1.2.4–1.2.4 again in Chapter 4.

The DFAs called De Bruijn's automata here are often referred to as *De Bruijn's graphs* in the literature. They are named after Nicolaas Govert de Bruijn, but were suggested independently by de Bruijn [10] and Irving John Good [15]. The property of De Bruijn's automata registered in Exercise 1.2.11 is known as *definiteness*: a DFA $\mathscr{A}$ is called *definite* if there exists a positive integer k such that the current state of $\mathscr{A}$ is uniquely determined by the last k input letters consumed by $\mathscr{A}$. Definite automata attracted much attention in the early 1960s, see, e.g., [29]. As they form a proper subclass of synchronizing automata, one may consider the notion of definiteness as a precursor for the notion of synchronizability.

1.3 Origins of Synchronizing Automata

sec:origins

The concept of a synchronizing automaton as presented in Section 1.2 was formalized at the beginning of the 1960s. In the vast literature on synchronizing automata, the 1964 paper [7] by Jan Černý, a Slovak computer scientist, serves as a standard reference[1]. However, it would be fair to mention other researchers who independently and about the same time came to the very same idea. Chung Laung Liu's PhD thesis [23] submitted in 1962 contains a whole chapter[2] devoted to rather a systematic study of synchronizing automata. Moreover, the term 'synchronizing automata' seems to originate from Liu's thesis: Liu used the term 'synchronizable' while Černý called such automata 'directable'. As Prof. Liu wrote (in an email communication of March 10, 2016), he and "his PhD thesis advisor Dean Arden first came up with the notion of synchronization in the early 1960s". Synchronizing automata also appeared in the 1963 technical report [20] by Arthur E. Laemmel under the name 'resettable machines'. Laemmel's report is less elaborated in comparison with [7] and [23] but still contains some valuable observations.

[1]The reader can find an English translation of this important paper in Appendix 1.
[2]This chapter is reproduced in Appendix 2.

In Černý's paper [7] the notion of a synchronizing automaton arose within the classic framework of Edward F. Moore's 'Gedanken-experiments' [25]. For Moore and his followers finite automata served as a mathematical model of devices working in discrete mode, such as computers or relay control systems. This leads to the following natural problem: how can we restore control over such a device if we do not know its current state but can observe outputs produced by the device under various actions? Moore [25] has shown that under certain conditions one can uniquely determine the state at which the automaton arrives after a suitable sequence of actions (called an *experiment*). Moore's experiments were adaptive, that is, each next action was selected on the basis of the outputs caused by the previous actions. Seymour Ginsburg [14] considered more restricted experiments that he called *uniform*. A uniform experiment[3] is just a fixed sequence of actions, that is, a word over the input alphabet; thus, in Ginsburg's experiments outputs were only used for calculating the resulting state at the end of an experiment. From this, just one further step was needed to come to the setting in which outputs were not used at all. It should be noted that this setting is by no means artificial—there exist many practical situations when it is technically impossible to observe output signals. (Think of a satellite which loops around the Moon and cannot be controlled from the Earth while 'behind' the Moon.)

In Liu's thesis three motivating applications were mentioned, see [23, Section 4.7]. The first one is resetting an automaton which current state is an unknown to a preselected state, so it is exactly the same problem as discussed in the preceding paragraph. The second application is a variation of the first when one deals with several copies of identical automata that are in different initial states and can accept identical input sequences in parallel, and one wants to make these copies work synchronously. The third application relates synchronizing automata to variable-length codes—Liu explains how synchronizing automata can provide codes able to restore synchronization between sender and recipient after a channel error. This connection which is indeed of utmost importance is discussed in more detail in the next section.

Laemmel's motivations were similar too: he also referred to Ginsburg's experiments like did Černý and also mentioned connections between his 'resettable machines' and variable-length codes. Specifically, Laemmel related synchronizing automata to so-called ergodic codes considered by Marcel-Paul Schützenberger [33].

[3]After [13], the name *homing sequence* has become standard for the notion.

It is not surprising that synchronizing automata were independently and simultaneously invented by several researchers: the notion was very natural by itself and fitted fairly well in what was considered as the mainstream of automata theory in the 1960s. Implicitly, the concept of a synchronizing automaton has been around since the earliest days of automata theory, that is, since 1956. We have already mentioned Schützenberger's paper [33]; as an entertaining example, we reproduce here an automaton that appeared in W. Ross Ashby's classic book 'An Introduction to Cybernetics' [3], see pp. 60–61 in this book or Appendix 3 to these lecture notes. There Ashby presents a puzzle dealing with taming two ghostly noises, Singing and Laughter, in a haunted mansion. Each of the noises can be either on or off, and their behaviours depend on combinations of two possible actions, playing the organ or burning incense. Under a suitable encoding, this leads to the DFA with 4 states and 4 input letters depicted in Figure fig:Ashby 1.7. Here 00 encodes

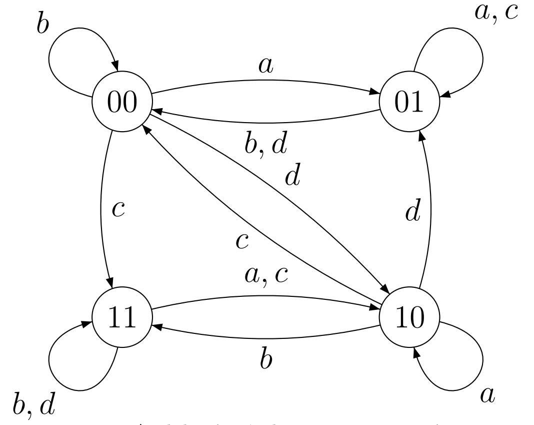

Figure 1.7: Ashby's 'ghost taming' automaton fig:Ashby

the state when both Singing and Laughter are silent, 01 stands for the state when Singing is of but Laughter is on, etc. Similarly, a stands for the transition that happens when neither the organ is played nor incense is burned, b encodes the transition caused by organ-playing in the absence of incense-burning, etc. The problem is to ensure silence, in other words, to bring the automaton in Figure fig:Ashby 1.7 to the state 00. As the reader sees, this is exactly the problem of finding a reset word for the automaton. While Ashby only solves the problem under the additional assumption that the automaton is in the state 11, his suggested solution is encoded by the word acb. It is easy to check that acb is in fact a reset word so that applying the corresponding sequence of actions will get the house quiet from any initial configuration. It is not clear whether or not Ashby realized this nice feature of his automaton, and moreover, the fact that Ashby's automaton is indeed synchronizing seems to be overlooked for many years.

Since the 1960s, the notion of a synchronizing automaton has been frequently rediscovered. One of the reasons for this was that the pioneering sources [7, 23, 20] were not easily accessible. Černý's paper [7] was written in Slovak and published in a local journal; because of this, it remained unknown in the English-speaking world for quite some time. As for Liu's thesis [23], even though Liu did publish a part of its results in a journal (cf. [22]), he did not include there any result related to synchronization, and thus, his contribution to the theory of synchronizing automata had not become widely known and got eventually forgotten. Laemmel's report [20] and its follow-up report by Laemmel and Beulahi Rudner [21] were not broadly circulated and basically remained unknown to anyone.

Another, more conceptual reason for synchronizing automata to be rediscovered again and again was that from time to time they suddenly 'popped up' in various areas of mathematics, computer science, and engineering that at first seemed to be rather unrelated to any of the problems that motivated [7, 23, 20]. We exhibit two examples of this sort in Sections sec:orienteer 1.5 and sec:dynamics 1.6 below, after giving a brief overview of the role of synchronizing automata in the theory of variable-length codes.

1.4 Synchronizing Automata and Codes

sec:codes

Since we do not assume the reader's acquaintance with coding theory, we start with a '5-minute tour', presenting its basic ideas in a nutshell.

Suppose we deal with data presented as a huge word w in some finite source alphabet Θ, and we know—or can estimate—the probability of occurrence in w for each letter from Θ. A good example is a long text in a natural language, like Marcel Proust's "À la recherche du temps perdu" with its approx. 9,609,000 characters, each letter, punctuation mark, and space being counted as one character. One can quite accurately estimate the probability of occurrence in this text for each character using available information about relative frequencies of letters in French. For instance, 'e' occurs in French words approx. twice as often as 'a' and the frequency of occurrence of 'k' is less than 0.9% of that of 'l'.

If we want to digitalize the data (for storing or transmitting them), we may encode the letters of Θ with some words over a smaller alphabet Σ, usually, the *binary alphabet* $\{0,1\}$. We refer to the words over $\{0,1\}$ as *binary words*. A (*lossless*) *encoding* of Θ is a map $\chi\colon \Theta \to \Sigma^+$ such that its extension to Θ^+ is injective, that is, every word $w \in \Theta^+$ is uniquely determined by the word from Σ^+ obtained by successive replacements of the letters of w with their images under χ. If $\Sigma = \{0,1\}$, we call χ a *binary encoding*. Given a set $X \subset \Sigma^+$, any encoding χ such that $X = \Theta\chi$ is referred to as an encoding with the *code* X.

Any encoding of letters of Θ with a set of binary words of constant length (such as ANSII-codes) requires $\lceil \log_2 |\Theta| \rceil$ bits for each letter and thus $|w| \cdot \lceil \log_2 |\Theta| \rceil$ bits for the whole word w. However, by a clever variable-length encoding we may save much space (in the case of data storage) and/or time (in the case of data transmission). For this, we should encode letters that occur in w more frequently with shorter binary words while letters with low probability of occurrence in w may be encoded with longer binary words without much harm. This simple idea was already used in Samuel Morse's telegraphic code of the 19th century: 'e', the most common letter in English has the shortest Morse code, a single dot.

A complication has to be taken into account when a variable-length encoding is used: the process of decoding, i.e., restoring a word w from the stream of bits that encodes w, may be not easy in general.

[Example]

There is however a class of encodings for which this complication does not appear. Let Σ be an alphabet. If $x, y \in \Sigma^*$ are such that $x = yz$ for some $z \in \Sigma^*$, then y is called a *prefix* of x; if z is non-empty, the prefix y is said to be *proper*. A *prefix code* over Σ is a set X of words from Σ^+ such that no word of X is a prefix of another word of X. Data encoded with a prefix code can be decoded on-the-fly: a decoder just keeps finding and removing prefixes that form valid code words from the incoming stream.

[Illustration: prefix codes as binary trees]

examp:you use a code c

Example 1.4.1. Consider the following set of binary words:

$$C = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}.$$

Clearly, it is a prefix code. We can use this code to efficiently encode the sentence YOU USE A CODE C of length 16. The source alphabet of this sentence consists of 9 characters (8 letters and space) so that every constant-length binary encoding of the sentence requires $16 \cdot \lceil \log_2 9 \rceil = 64$ bits. Since space occurs 4 times, each of C,E,O,U occurs twice, and each of A,D,S,Y occurs once, one may use the following variable-length encoding of the source alphabet:

space	C	E	O	U	A	D	S	Y
10	000	010	110	111	0010	0011	0110	0111

The sentence YOU USE A CODE C is then encoded with the binary word

0111 | 110 | 111 | 10 | 111 | 0110 | 010 | 10 | 0010 | 10 | 000 | 110 | 0011 | 010 | 10 | 000

of length 48 (vertical bars separating code words are inserted for readability only). We have thus reduced the binary representation size by 25%.

Claude Shannon's source coding theorem [34] provides a lower bound on the minimum size of binary encodings of a given word in terms of frequencies of its letters occurring. If $w = a_1 \cdots a_\ell$ with $a_1, \ldots, a_\ell \in \Theta$ is a word from Θ^+ and $a \in \Theta$ is a letter, the *frequency* p_a of a in w is the ratio of the number of occurrences of a in the sequence $a_1, \ldots, a_\ell$ to the length of w, that is, in symbols,

$$p_a := \frac{|\{i \mid a_i = a\}|}{|w|}.$$

The *entropy* of w is then defined as

$$H(w) := -\sum_{a\in\Theta} p_a \log_2 p_a. \tag{1.1}$$ eq:entropy

The source coding theorem implies that any binary encoding of w such that w can be uniquely recovered from the encoding requires at least $H(w)|w|$ bits. In Example examp:you use a code c 1.4.1, The entropy of the sentence YOU USE A CODE C from Example examp:you use a code c 1.4.1 is

$$-\frac{1}{4}\log_2\frac{1}{4} - 4\cdot\frac{1}{8}\log_2\frac{1}{8} - 4\cdot\frac{1}{16}\log_2\frac{1}{16} = \frac{1}{2} + \frac{3}{2} + 1 = 3,$$

whence any uniquely decodable binary encoding of the sentence must have at least $3 \cdot 16 = 48$ bit. Therefore the code C suggested in Example examp:you use a code c 1.4.1 provides an optimal encoding. It is known that, in general, the most economical binary presentation of data that can be achieved by any variable-length encoding can always be achieved by a suitable encoding with a prefix code.

A prefix code is *maximal* if it is not contained in another prefix code over the same alphabet. A maximal prefix code X over Σ is *synchronized* if there is a word z such that for any word $y \in \Sigma^*$, the word yz can be decomposed as a product of words from X. Such a word y is called a *synchronizing word* for X. The advantage of synchronized codes is that they are able to recover after a loss of synchronization between the decoder and the coder caused by channel errors: in the case of such a loss, it suffices to transmit a synchronizing word and the following symbols will be decoded correctly. Moreover, since the probability that a word $x \in \Sigma^*$ contains a fixed factor z tends to 1 as the length of x increases, synchronized codes eventually resynchronize by themselves, after sufficiently many symbols being sent. (As shown in [5], the latter property in fact characterizes synchronized codes.)

The binary code $C = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$ used in Example examp:you use a code c 1.4.1 can illustrate this notion. Indeed, C is a maximal prefix code and one can check that each of the words 010, 011110, 011111110, … is a synchronizing word for C. Suppose that the code word 000 has been sent but, due to a channel error, the word 100 has been received. The decoder, which is unaware of the error, interprets 10 as a code word, and thus, loses synchronization as it treats the next bit as a part of another code word. However, with a high probability this synchronization loss only propagates for a short while since the decoder resynchronizes as soon as it encounters one of the factors 010, 011110, 011111110, … in the received stream of symbols. A few samples of such streams are shown in Figure fig:decoding 1.8 in which vertical bars show the partition of each stream into code words and the boldfaced code words indicate the position at which the decoder resynchronizes.

Sent	0 0 0 | 0 0 1 0 | **0 1 1 1** | …
Received	1 0 | 0 0 0 | 1 0 | **0 1 1 1** | …
Sent	0 0 0 | 0 1 1 1 | 1 1 0 | 0 0 1 1 | 0 0 0 | 1 0 | **1 1 0** | …
Received	1 0 | 0 0 1 1 | 1 1 1 | 0 0 0 | 1 1 0 | 0 0 1 0 | **1 1 0** | …
Sent	0 0 0 | 0 0 0 | 1 1 1 | **1 0** | …
Received	1 0 | 0 0 0 | 0 1 1 1 | **1 0** | …

Figure 1.8: Restoring synchronization fig:decoding

The obvious similarity in terminology suggests that synchronized codes are somehow related to synchronizing automata. Indeed, the relationship exists and is in fact very close. If X is a finite prefix code over an alphabet

Σ, then its decoding can be implemented by a finite automaton that is defined as follows. Let Q be the set of all proper prefixes of the words in X (including the empty word ε). For $q \in Q$ and $a \in \Sigma$, define

$$q \,.\, a = \begin{cases} qa & \text{if } qa \text{ is a proper prefix of a word of } X\,, \\ 1 & \text{if } qa \in X\,. \end{cases}$$

The resulting automaton $\mathscr{A}_X$ is complete whenever the code X is maximal.

[Illustration: constructing the decoder of a prefix code from its tree]

It is easy to show that $\mathscr{A}_X$ is a synchronizing automaton if and only if X is a synchronized code. Moreover, a word x is synchronizing for X if and only if x is a reset word for $\mathscr{A}_X$ and sends all vertices in Q to the vertex 1.

[Proof]

Exercises for Section sec:codes 1.4

exer:1.4.1 *Exercise* 1.4.1 (easy). The maximal prefix code

$$D = \{0000, 0001, 001, 0100, 0101, 011, 100, 101, 11\}$$

can also be used to provide an optimal binary encoding for the sentence YOU USE A CODE C from Example examp:you use a code c 1.4.1. Represent D by a binary tree and construct its decoder $\mathscr{A}_D$.

exer:1.4.2 *Exercise* 1.4.2 (medium). Is the code D from Exercise exer:1.4.1 1.4.1 synchronized?

exer:1.4.3 *Exercise* 1.4.3 (easy). Calculate the entropy of the sentence YOU USE THE CODE D, AND SO DO WE (each letter, punctuation mark, and space being counted as one character) and determine Shannon's a lower bound on the minimum size of binary encodings of this sentence.

exer:1.4.4 *Exercise* 1.4.4 (difficult). Construct a maximal prefix code that provides an optimal binary encoding of the sentence from Exercise exer:1.4.3 1.4.3. Is your code synchronized?

1.5 Reinventing by Engineers

sec:orienter

An additional source of problems related to synchronizing automata has come from *robotics* or, more precisely, from part handling problems in industrial automation such as part feeding, fixturing, loading, assembly and packing. Within this framework, the concept of a synchronizing automaton was independently rediscovered in the mid-1980s by Balas Natarajan [26, 27] who showed how synchronizing automata can be used to design sensor-free orienters for polygonal parts.

We explain the idea of using synchronizing automata in this area by the following illustrative example.

Suppose that one of the parts of a certain device has the shape shown in Figure 1.9. Such parts arrive at manufacturing sites in boxes and they

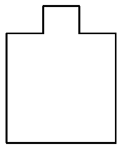

Figure 1.9: A polygonal part fig:detail

need to be sorted and oriented before assembly. For simplicity, assume that only four initial orientations of the part of Figure 1.9 are possible, namely, the ones shown in Figure 1.10. Further, suppose that prior the assembly

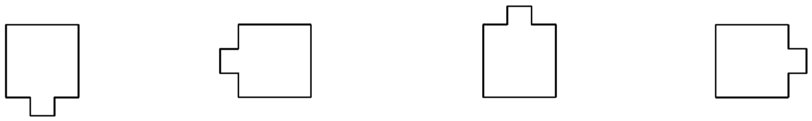

Figure 1.10: Four possible orientations fig:orientatio

the part should take the 'bump-left' orientation (the second from the left in Figure 1.10). Thus, one has to construct an orienter which action will put the part in the prescribed position independently of its initial orientation.

Of course, there are many ways to design such an orienter but practical considerations favor methods which require little or no sensing, employ simple devices, and are as robust as possible. For our particular case, these goals can be achieved as follows. We put parts to be oriented on a conveyer belt which takes them to the assembly point and let the stream of the parts encounter a series of passive obstacles placed along the belt. We need two type of obstacles: tall and short. A tall obstacle should be tall enough in order that any part on the belt encounters this obstacle by its rightmost low angle (we assume that the belt is moving from left to right). Being carried by the belt, the part then is forced to turn 90° clockwise as shown in Figure 1.11. A

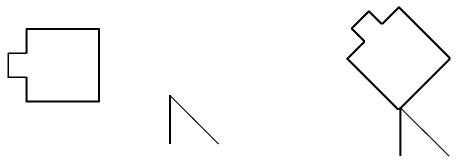

Figure 1.11: The action of the tall obstacle

fig:obstacle action

short obstacle has the same effect whenever the part is in the 'bump-down' orientation (the first from the left in Figure 1.10); otherwise it does not touch the part which therefore passes by without changing the orientation.

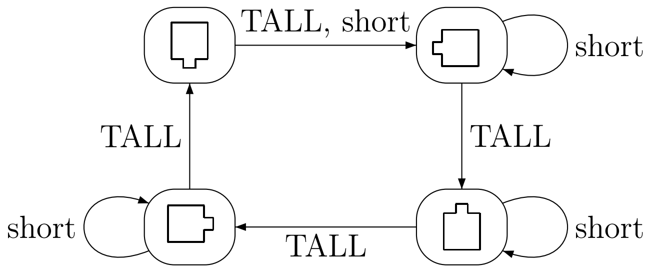

Figure 1.12: The actions of the obstacles summarized

fig:orienter

The scheme in Figure 1.12 summarizes how the aforementioned obstacles effect the orientation of the part. The reader immediately recognizes the synchronizing automaton from Figure 1.1. Since it is reset by the word *abbbabbba* (see Example 1.2.1, we conclude that the series of obstacles

short–TALL–TALL–TALL–short–TALL–TALL–TALL–short

yields the desired sensorless orienter.

Exercises for Section 1.5

exer:1.5.1

Exercise 1.5.1 (medium). Consider once more the polygonal part shown in Figure 1.9. Suppose that we can arrange two conveyer belts so that when the part slides from one belt onto the other, it turns 180° around its horizontal axis. This adds a new action to the ones shown in Figure 1.12 and leads to the following picture (in which the new action is called *turn*). Find a reset word of minimum length for the DFA depicted in Figure 1.13.

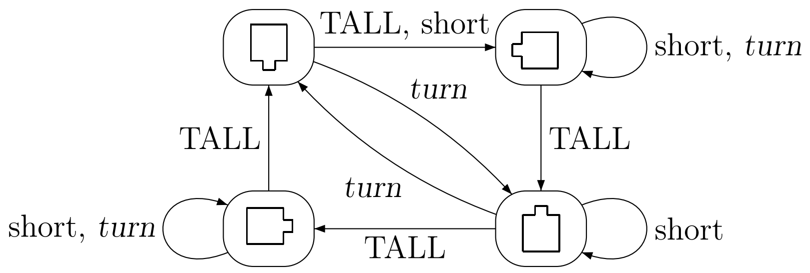

Figure 1.13: Adding the rotation around the horizontal axis fig:new orient

exer:1.5.2 *Exercise* 1.5.2 (medium). Now suppose that two conveyer belts are arranged so that when the part from Figure fig:detail 1.9 slides from one belt onto the other, it turns 180° around its vertical axis. Find a reset word of minimum length for the DFA representing this action along with the actions of obstacles shown in Figure fig:orienter 1.12.

Comments and Supplements for Section sec:orienter 1.5

1.6 Reinventing by Dynamic Theorists

sec:dynamics

In the 1990s, synchronizing automata were rediscovered by dynamic theorists who studied constant length substitutions. A *substitution* on a finite alphabet X is a map $\sigma\colon X \to X^+$; the substitution is said to be of *constant length* if all words $\sigma(x)$, $x \in X$, have the same length. One says that σ satisfies the *coincidence condition* if there exist positive integers m and k such that all words $\sigma^k(x)$ have the same letter in the m-th position. For an example, consider the substitution τ on $X = \{0, 1, 2\}$ defined by

$$0 \mapsto 11,\ 1 \mapsto 12,\ 2 \mapsto 20. \qquad (1.2)$$

eq:substitutio

Calculating the iterations of τ up to τ^4 (see Figure KV:fig:substituion 1.14), we observe that τ

$$\begin{array}{ccccccccc}
0 & \mapsto & 11 & \mapsto & 1212 & \mapsto & 12201220 & \mapsto & 1220201\mathbf{1}112202011 \\
1 & \mapsto & 12 & \mapsto & 1220 & \mapsto & 12202011 & \mapsto & 1220201\mathbf{1}120111212 \\
2 & \mapsto & 20 & \mapsto & 2011 & \mapsto & 20111212 & \mapsto & 2011121\mathbf{2}12201220
\end{array}$$

Figure 1.14: A substitution satisfying the coincidence condition KV:fig:substit

satisfies the coincidence condition (with $k = 4$, $m = 7$).

The importance of the coincidence condition comes from the crucial fact (established by Frederik Michel Dekking [11]) that it is this condition that completely characterizes the constant length substitutions that give rise to dynamical systems measure-theoretically isomorphic to a translation on a compact abelian group, see [31, Chapter 7] for a survey. For us, however, the coincidence condition is primarily interesting as yet another incarnation of synchronizability. Indeed, there is a straightforward bijection between DFAs and constant length substitutions. Each DFA $\mathscr{A} = \langle Q, \Sigma, \delta\rangle$ with $\Sigma = \{a_1, \dots, a_\ell\}$ defines a length ℓ substitution on Q that maps every $q \in Q$ to the word $(q\,.\,a_1)\cdots(q\dot{a}_\ell) \in Q^+$. For instance, the automaton $\mathscr{C}_4$ in Figure KV:fig:C4 1.1 induces the substitution

$$0 \mapsto 11,\ 1 \mapsto 12,\ 2 \mapsto 23,\ 3 \mapsto 30.$$

Conversely, each substitution $\sigma\colon X \to X^+$ such that all words $\sigma(x)$, $x \in X$, have the same length ℓ gives rise to a DFA for which X serves as the state set and which has ℓ input letters $a_1, \dots, a_\ell$, say, acting on X as follows: $x \cdot a_i$ is the symbol in the i-th position of the word $\sigma(x)$. For instance, the substitution τ considered in the previous paragraph defines the automaton shown in Figure KV:fig:C3 1.15. It is easy to realize that under the described bijec-

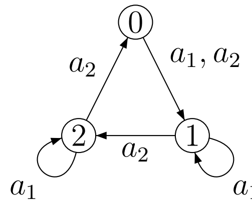

Figure 1.15: The automaton induced by the substitution (eq:substitution 1.2) KV:fig:C3

tion substitutions satisfying the coincidence condition correspond precisely to synchronizing automata, and moreover, given a substitution, the number of iterations at which the coincidence first occurs is equal to the minimum length of reset word for the corresponding automaton.

Comments and Supplements for Section sec:dynamics 1.6

1.7 Algebraic Perspective

sec:algebra

A non-empty set Q endowed with operations $f_1\colon \underbrace{Q \times Q \times \cdots \times Q}_{n_1} \to Q$, $f_2\colon \underbrace{Q \times Q \times \cdots \times QA}_{n_2} \to Q$, ... is called an *algebra of type* $(n_1, n_2, \ldots)$ *with the carrier* Q. An algebra $(Q; f_1, f_2, \ldots)$ of type $(1, 1, \ldots)$ is called *unary*. Automata as we defined them in Section sec:dfa 1.1 fit very well in this framework: a DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ is nothing but the unary algebra with the carrier Q on which for each letter $a \in \Sigma$, the unary operation $f_a\colon Q \to Q$ is defined by the rule $f_a\colon q \mapsto \delta(q, a)$. This viewpoint allows us to apply to automata all standard algebraic notions, e.g., the notions of a subalgebra (*subautomaton*), a *congruence*, a *quotient automaton*.

Observe in passing that synchronizing automata appear in a natural way within this algebraic framework. A *term* in the language of unary algebras is an expression t of the form $x \,.\, w$, where x is a variable and w is a word over an alphabet Σ. An *identity* is a formal equality between two terms. A DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ *satisfies* an identity $t_1 = t_2$, where the words involved in the terms t_1 and t_2 are over Σ, if t_1 and t_2 take the same value under each interpretation of their variables in the set Q. Identities of unary algebras can be of the form either $x \,.\, u = x \,.\, v$ (*homotypical* identities) or $x \,.\, u = y \,.\, v$ with $x \neq y$ (*heterotypical* identities). It is easy to realize that a DFA is synchronizing if and only if it satisfies a heterotypical identity, and thus, studying synchronizing automata may be considered as a part of the equational logic of unary algebras. In particular, synchronizing automata over a fixed alphabet form a *pseudovariety* of unary algebras. See [4] for a survey of numerous publications in this direction; it is fair to say, however, that so far this algebraic approach has not proved to be really useful for understanding the combinatorial nature of synchronizing automata.

Chapter 2

Algorithmic Issues

ch:algorithms

Chapter 3

Complexity Issues

ch:complexity

Chapter 4

The Černý Conjecture

rny_conjecture

Chapter 5

Road Coloring Theorem

:road coloring

5.1 Stating the Problem

cp formulation

Recall our definition of a graph from Section sec:dfa 1.1. A graph is a quadruple $\langle V, E, h, t\rangle$ of sets and maps: V and E are, respectively, the sets of vertices and edges while the maps $h, t\colon E \to V$ map every edge to its head and respectively tail vertex. Given a vertex $v \in V$, its *out-degree* $\operatorname{outdeg}(v)$ is the number of edges for which v serves as the head vertex, i.e.,

$$\operatorname{outdeg}(v) := \big|\{e \in E \mid h(e) = v\}\big|.$$

A graph Γ in which each vertex has the same out-degree (say, k) is called a *graph of constant out-degree* and the number k is referred to as the *out-degree* of Γ and is denoted by $\operatorname{outdeg}(\Gamma)$. Clearly, if $\mathscr{A} = \langle Q, \Sigma, \delta\rangle$ is a DFA, then the underlying graph of $\mathscr{A}$ is a graph of constant out-degree and its out-degree is equal to $|\Sigma|$. Conversely, if Γ is a graph of constant out-degree and Σ is an alphabet such that $|\Sigma = \operatorname{outdeg}(\Gamma)$, one can label the edges of Γ with letters of Σ such that the resulting automaton will be complete and deterministic. Any DFA obtained this way is referred to as a *coloring* of Γ. For instance, Figure fig:two colorings 5.1 shows a graph of out-degree 2 and two of its possible colorings, one of which being nothing but our old chap, the automaton $\mathscr{C}_4$.

Given a graph of constant out-degree, it is reasonable to ask under which conditions it admits a coloring satisfying some 'good' properties. In this chapter we analyze the so-called *Road Coloring Problem* that is certainly the most famous question within this framework. The Road Coloring Problem asks under which conditions graphs of constant out-degree admit a synchronizing coloring.

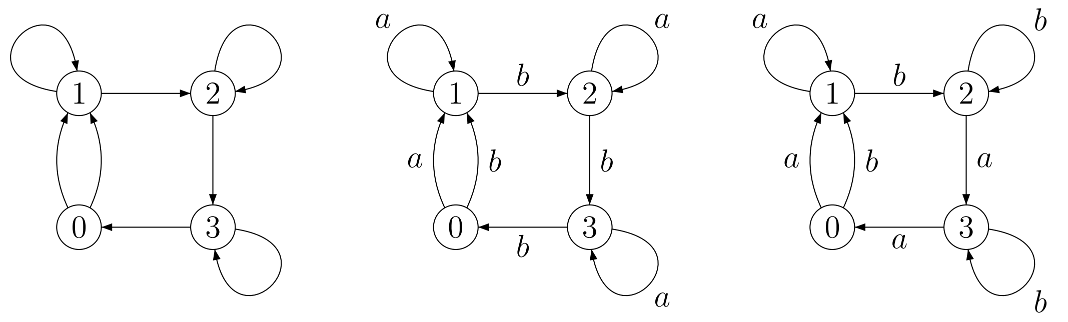

Figure 5.1: A graph and two of its colorings

fig:two colori

The problem was explicitly stated and coined by Roy Adler, L. Wayne Goodwyn, and Benjamin Weiss [1] in 1977; in an implicit form it was present already in an earlier memoir by Adler and Weiss [2]. The original motivation in [2, 1] came from symbolic dynamics, see comments at the end of the section. However, the Road Coloring Problem is quite natural also from the viewpoint of the 'reverse engineering' of synchronizing automata: we aim to relate geometric properties of graphs to combinatorial properties of automata built on those graphs.

For the sake of completeness, we recall some standard graph-theoretical notions which are used in subsequent discussions. Two edges e, e' of a graph $\Gamma = \langle V, E, h, t\rangle$ are said to be *consecutive* if $t(e) = h(e')$. A *path* in Γ is a word over the set E of its edges such that any two adjacent edges in this word are consecutive; the length of the word is called the *length* of the path. Observe that by the above definition the empty word over E is a path (of length 0) referred to as the *empty path*. We say that a path *starts* or *originates* at the head vertex of its first edge and *ends* or *terminates* at the tail vertex of its last edge. We adopt the convention that the empty path may start at any vertex. If a path starts at a vertex v and ends at a vertex v', we refer to is as a *path from v to v'*. A path that terminates at the same vertex at which it starts is called a *cycle*.

A vertex v' is said to be *reachable* from a vertex v if there is a path from v to v'. Clearly, the *reachability relation* is reflexive (thanks to the above convention about the empty path) and transitive, and the mutual reachability relation is an equivalence on the set V. If this equivalence is universal, i.e., if every vertex is reachable from every vertex, the graph is called *strongly connected.* In general The subgraphs induced on the classes of the mutual reachability relation are strongly connected and are called the *strongly connected components* of the graph Γ. The reachability relation induces a partial order on the set of the strongly connected components: a component Γ_1 precedes a component Γ_2 in this order if some vertex of Γ_1 is reachable from some vertex of Γ_2.

Adler, Goodwyn and Weiss considered only strongly connected graphs; as we shall see below this is quite a natural assumption since the general case easily reduces to the case of strongly connected graphs.

We start our analysis of the Road Coloring Problem with registering a simple condition that holds in underlying graphs of strongly connected synchronizing automata.

Proposition 5.1.1. *If a strongly connected graph Γ admits a synchronizing coloring, then the g.c.d. of lengths of all cycles in Γ is equal to 1.*

Proof. Arguing by contradiction, let $k > 1$ be a common divisor of lengths of the cycles in Γ. Let V denote the vertex set of Γ. Take a vertex $v_0 \in V$ and, for $i = 0, 1, \dots, k-1$, let

$$V_i = \{v \in V \mid \text{there exists a path from } v_0 \text{ to } v \text{ of length } i \pmod{k}\}.$$

Since the graph Γ is strongly connected, for each $v \in V$ there exists a path from v_0 to v, whence $V = \bigcup_{i=0}^{k-1} V_i$. We claim that $V_i \cap V_j = \varnothing$ if $i \neq j$.

Let $v \in V_i \cap V_j$ where $i \neq j$. This means that in Γ there are two paths from v_0 to v: of length $\ell \equiv i \,(\mathrm{mod}\ k)$ and of length $m \equiv j \,(\mathrm{mod}\ k)$. Since Γ is strongly connected, there exists also a path from v to v_0 of length n, say. Combining it with each of the two paths above we get a cycle of length $\ell + n$ and a cycle of length $m + n$. Since k divides the length of any cycle in Γ, we have $\ell + n \equiv i + n \equiv 0 \,(\mathrm{mod}\ k)$ and $m + n \equiv j + n \equiv 0 \,(\mathrm{mod}\ k)$, whence $i \equiv j \,(\mathrm{mod}\ k)$, a contradiction.

Thus, V is a disjoint union of $V_0, V_1, \dots, V_{k-1}$, and by the definition each edge in Γ leads from V_i to $V_{i+1\,(\mathrm{mod}\ k)}$. Then Γ definitely cannot be converted into a synchronizing automaton by any coloring of its edges: no paths of the same length ℓ originated in V_0 and V_1 can terminate at the same vertex because they end in $V_{\ell\,(\mathrm{mod}\ k)}$ and in $V_{\ell+1\,(\mathrm{mod}\ k)}$ respectively. □

Graphs satisfying the conclusion of Proposition 5.1.1 are called *primitive*[1]. Adler, Goodwyn and Weiss [1] conjectured that primitivity is not only

[1]In the literature such graphs are sometimes called *aperiodic*. The term 'primitive' comes from the notion of a primitive matrix in the Perron–Frobenius theory of non-negative matrices: it is known (and easy to see) that a graph is primitive if and only if so is its incidence matrix.

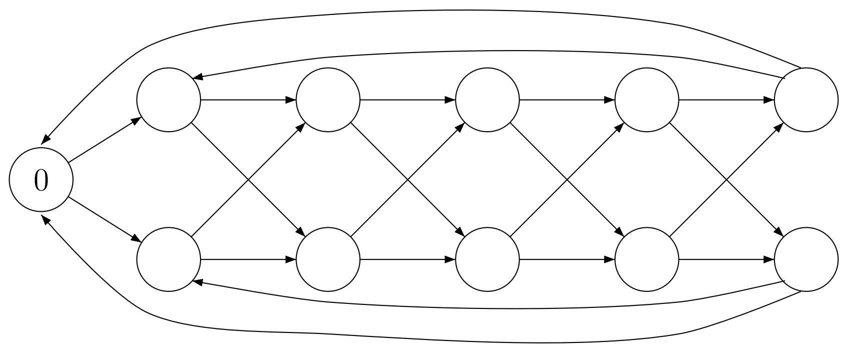

Figure 5.2: Graph admitting a synchronizing coloring fig:gusev

necessary for a graph to have a synchronizing coloring but also sufficient. In other word, they suggested the following *Road Coloring Conjecture*: every strongly connected primitive graph with constant out-degree admits a synchronizing coloring.

The Road Coloring Conjecture has attracted much attention. There were several interesting partial results (see, e.g., [28, 12, 30, 16, 6, 18, 19]), and finally the conjecture was confirmed by Avraam Trahtman [37]. We present Trahtman's solution in Section sec:rcp proof 5.3, after the necessary preparations being made in Section sec:rcp confluence 5.2.

Exercises for Section sec:rcp formulation 5.1

exer:5.1.1 *Exercise* 5.1.1 (easy). Show that a graph of constant out-degree admits a synchronizing coloring whenever the graph has a loop.

exer:5.1.2 *Exercise* 5.1.2 (medium). Verify that every coloring of the graph shown in Figure fig:two colorings 5.1 is synchronizing.

Given a DFA $\mathscr{A}$, colorings of its underlying graph are referred to as *recolorings* of $\mathscr{A}$.

exer:5.1.3 *Exercise* 5.1.3 (hard). Prove that for each $n = 2, 3, \dots$, every recoloring of the Černý automaton $\mathscr{C}_n$ is synchronizing.

exer:5.1.4 *Exercise* 5.1.4 (hard). The graph shown in Figure fig:gusev 5.2 admits both synchronizing and non-synchronizing colorings. Find such colorings, and for the synchronizing one, construct a reset word of minimum length that leads to the vertex 0.

The next two exercises assume that the reader is acquainted with the notions of a semigroup and a group and with the standard notation for cyclic permutations of a finite set. Recall that a subset X of a semigroup S is said to *generate* S if every element in S can be represented as a product of element from X. The (right) *Cayley graph* of S with respect to its generating set X is the DFA $\mathscr{C}_X(S)$ with S as the state set, X as the input alphabet, and the product map $(s,x) \mapsto sx$ as the transition function $S \times X \to S$.

examp:S3 *Example* 5.1.1. Consider the group $\mathbf{S}_3$ of all permutations of the set $\{1,2,3\}$. It is known (and is easy to verify) that $\mathbf{S}_3$ is generated by the set $X = \{\sigma, \tau\}$, where σ stands for the cyclic shift (123) and τ is the transposition (12). The Cayley graph $\mathscr{C}_X(\mathbf{S}_3)$ is shown in Figure fig:cayley S3 5.3, in which id denotes the identical permutation.

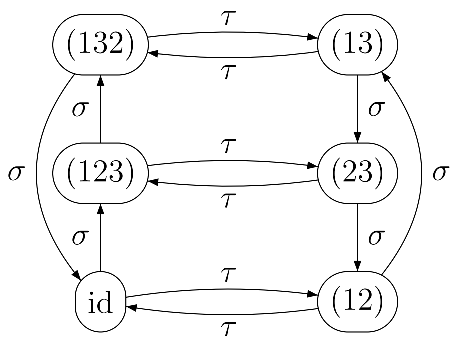

Figure 5.3: The Cayley graph of the group $\mathbf{S}_3$ fig:cayley S3

exer:5.1.5 *Exercise* 5.1.5 (easy). Find a synchronizing recoloring of the Cayley graph $\mathscr{C}_X(\mathbf{S}_3)$ shown in Figure fig:cayley S3 5.3.

exer:5.1.6 *Exercise* 5.1.6 (medium). Does the Cayley graph of the group of all permutations of the set $\{1,2,3,4\}$ with respect to its generating set consisting of the cyclic shift (1234) and the transposition (12) admit a synchronizing recoloring?

Comments and Supplements for Section sec:rcp formulation 5.1

The name 'Road Coloring Problem' suggested in [1] comes from the following interpretation. In every strongly connected synchronizing automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, one can assign to state $q \in Q$ an instruction (a reset word) w_q such that following w_q one will surely arrive at q from any initial state. (Indeed, for this one should first follow an arbitrary reset word leading to some state p, say, and then follow a word that labels a path connecting p and q—such a path exists because of strong connectivity.) Thus, in order to help a traveler lost on a given strongly connected graph Γ of constant out-degree to find his/her way from wherever he/she could be, we should if possible color (that is, label) the edges of Γ such that Γ becomes a synchronizing automaton and then tell the traveler the magic sequence of colors representing a reset word leading to the traveler's destination.

5.2 The Confluence Relation

sec:rcp confluence

Trahtman's proof heavily depends on a neat idea of *confluence* which is due to Karel Culik II, Juhani Karhumäki and Jarkko Kari [9]. If $\mathscr{A} = \langle Q, \Sigma, \delta\rangle$ is a DFA, the *confluence relation* $\sim$ on the set Q is defined as as follows:

$$q \sim q' \iff \forall u \in \Sigma^* \; \exists v \in \Sigma^* \; (q \,.\, u) \,.\, v = (q' \,.\, u) \,.\, v.$$

Any pair $(q, q') \in Q \times Q$ such that $q \neq q'$ and $q \sim q'$ is called *confluent*. One can think on the confluence relation in terms of the following 2-shot game played by two players: Separator (server) and Synchronizer (receiver). Given two states $q, q' \in Q$, Synchronizer aims to bring them together by the action of a suitable word in Σ^* while Separator tries to keep them apart. Separator serves by choosing a word $u \in \Sigma^*$ and applying it at the states q and q'; Synchronizer then returns the serve by choosing a word $v \in \Sigma^*$ and applying it at the states $q \,.\, u$ and $q' \,.\, u$. The pair (q, q') is confluent if and only if it constitutes a starting position at which Synchronizer can win, i.e., Synchronizer has a return ace against every serve of Separator.

[Illustration: Separator serves, Synchronizer returns]

lm:confluence

Lemma 5.2.1. *The confluence relation is a congruence.*

Proof. Clearly, $\sim$ is reflexive and symmetric. To prove transitivity, take $p, q, r \in Q$ such that $p \sim q$ and $q \sim r$. We are to show that $p \sim r$. Consider an arbitrary serve $u \in \Sigma^*$ for the pair (p, r). Since $p \sim q$, there exists a return ace v against u, and since $q \sim r$, there exists a return ace w against uv if the latter word is considered as a serve for (q, r). Then

$$\begin{aligned}
(p \,.\, u) \,.\, vw &= ((p \,.\, u) \,.\, v) \,.\, w \\
&= ((q \,.\, u) \,.\, v) \,.\, w && \text{as } v \text{ is a return ace against } u \text{ for } (p, q) \\
&= (q \,.\, uv) \,.\, w \\
&= (r \,.\, uv) \,.\, w && \text{as } w \text{ is a return ace against } uv \text{ for } (q, r) \\
&= (r \,.\, u) \,.\, vw.
\end{aligned}$$

We see that vw is a return ace against u for (p, r), and thus, $p \sim r$.

It remains to verify that $q \sim q'$ implies $q \,.\, a \sim q' \,.\, a$ for every letter $a \in \Sigma$. This is straightforward: if $u \in \Sigma^*$ is an arbitrary serve for the pair $(q \,.\, a, q' \,.\, a)$, then clearly, any return ace against the word au considered as a serve for (q, q') is a return ace against u for $(q \,.\, a, q' \,.\, a)$. □

prop:ckk **Proposition 5.2.1** (Culik, Karhumäki and Kari [9]). *If every strongly connected primitive graph with constant out-degree and more than one vertex has a coloring with a confluent pair of vertices, then the Road Coloring Conjecture is true.*

Proof. Let Γ be a strongly connected primitive graph with constant out-degree. We show that Γ has a synchronizing coloring by induction on the number of vertices in Γ. If Γ has only one vertex, there is nothing to prove. If Γ has more than one vertex, then, by the assumption, it admits a coloring with a confluent pair of vertices by the letters of some alphabet Σ. Let $\mathscr{A}$ be the automaton resulting from this coloring. By Lemma lm:confluence 5.2.1, confluence relation is a congruence of $\mathscr{A}$. Since the relation is non-trivial, the quotient automaton $\mathscr{A}/{\sim}$ has fewer vertices than A. Since the quotient graph of every strongly connected graph is again strongly connected, the underlying graph $\Gamma/{\sim}$ of $\mathscr{A}/{\sim}$ is strongly connected. Moreover, since each cycle in Γ induces a cycle of the same length in $\Gamma/{\sim}$, the latter graph is primitive as well. Therefore, by the induction assumption, the graph $\Gamma/{\sim}$ admits a synchronizing coloring.

We lift this coloring to a coloring of Γ in the following natural way. If $p \to q$ is an edge in Γ, let $[p], [q]$ denote the $\sim$-congruence classes of p, q. If in the new coloring of the quotient graph, the edge $[p] \to [q]$ has color a', then we recolor $p \to q$ in the color a' as well.

Let $\mathscr{A}'$ be the automaton resulting from the lifted coloring; we will show that $\mathscr{A}'$ is synchronizing. Let w be a reset word for the synchronizing coloring of $\Gamma/{\sim}$. Then w maps the set of all vertices to a set S that is contained in a single congruence class of $\sim$. List all pairs of distinct vertices of S in some (arbitrary) order: $(x_1, y_1), \ldots, (x_n, y_n)$. By the definition of $\sim$, there exist words $v_1, v_2, \ldots, v_n$ such that

$$x_1 \,.\, v_1 = y_1 \,.\, v_1,$$
$$(x_2 \,.\, v_1) \,.\, v_2 = (y_2 \,.\, v_1) \,.\, v_2, \ldots,$$
$$(\cdots (x_n \,.\, v_1) \cdots) \,.\, v_n = (\cdots (y_n \,.\, v_1) \cdots) \,.\, v_n.$$

Therefore the word $wv_1v_2 \cdots v_n$ is a reset word for $\mathscr{A}'$. □

sec:rcp proof

5.3 Trahtman's Proof

Hints for Selected Exercises

Appendix 1: Černý's 1964 paper

This appendix contains the English translation of Černý's seminal paper [7]. The translation made by Jan Černý himself was first published in the CD proceedings of the Workshop on Synchronizing Automata held in Turku, Finland, in July 2004, as a satellite event of the 31st International Colloquium on Automata, Languages and Programming.

A NOTE ON HOMOGENEOUS EXPERIMENTS WITH FINITE AUTOMATA

JAN ČERNÝ

1. Introduction

Problems related to finite automata are receiving increasing attention. This is not surprising since finite automata may serve as a mathematical model of devices working in discrete mode, e.g., computers or relay control systems.

Several papers, e.g., [1,2], deal with the question whether is it possible to find such a homogeneous experiment for a Moore automaton (defined by a finite sequence of input signals) that determines the unique final state of the machine, independently on its initial state. In the papers mentioned above one can find the answer for the case of automata with mutually distinguishable internal states. The crucial role here plays a comparison of input and output relation in the experiment.

The main purpose of this note is to show that same problem can be solved even in the case when (e.g., for some technical reasons) it is impossible to observe output signals. Necessary and sufficient condition are presented for the existence of such an experiment. Finally, minimum of the length of the experiment is estimated if it exists.

2. Notions

A **finite automaton** is a triple $T = (X, Y, g)$, where X, Y are nonempty finite sets and g is a mapping from $X \times Y$ into X. The set X is said to be a **set of states**, Y is said to be a **set of inputs** and g is said to be a **transition mapping** of the automaton T.

In the sequel, if not specified otherwise, “the given automaton” will mean the automaton $T = (X, Y, g)$. The set of all natural numbers will be denoted by N and we use $Y^n = \overset{n}{\underset{j=1}{\times}} Y$ for $n \in N$. Let f be a mapping from the set $M_1 \times M_2$ into the set P and let $y \in M_2$. Then $f(\bullet, y)$ denotes a mapping from the set M_1 into the set P determined by fixed $y \in M_2$.

For $n_i \in N$ $(i = 1, \ldots, k)$ and

$$\begin{gathered} y^{n_1} = (y_{11}, \ldots, y_{1,n_1}) \in Y^{n_1}, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ y^{n_k} = (y_{k1}, \ldots, y_{k,n_k}) \in Y^{n_k}, \end{gathered}$$

we denote $(y^{n_1}, \ldots, y^{n_k}) = (y_{11}, \ldots, y_{1,n_1}, \ldots, y_{k1}, \ldots, y_{k,n_k})$.

Let $n \in N$. We define a mapping g^n of the set $X \times Y^n$ into X as follows:

$$g^n(x, y^n) = g^n(x, y_1, \ldots, y_n) = g(g(\ldots g(x, y_1), y_2) \ldots, y_n)$$

for each $x \in X$ and $y^n = (y_1, \ldots, y_n) \in Y^n$. Then the automaton $T^n = (X, Y^n, g^n)$ is called the n^{th} **power** of T.

Let $x \in X$ and $x_0 \in X$. We say that x is **transformable into** x_0 if there is $n \in N$ and $y^n = (y_1, \ldots, y_n) \in Y^n$ such that

$$g^n(x, y_1, \ldots, y_n) = x_0. \tag{1}$$

We say that x is **transformable into** x_0 **by** y^n if (1) holds.

The automaton T is said to be **connected** if there exists $x_0 \in X$ such that any $x \in X$ is transformable into x_0. T is said to be **strongly connected** if x_1 is transformable into x_2 for each couple $x_1, x_2 \in X$.

It is obvious that each strongly connected automaton is connected, but the converse needs not be true.

Denote for $X_1 \subset X$ and $y \in Y$ the set

$$g(X_1, y) = \{x \in X\colon \text{ there exists } \underline{x} \in X_1,\ g(\underline{x}, y) = x\}.$$

A state $x_0 \in X$ is said to be a **connecting state** if there are $x \in X$, $\underline{x} \in X$, $y \in Y$ such that $g(\{x, \underline{x}\}, y) = \{x_0\}$.

Usually, the behavior of a finite automaton is supposed to be the following. The automaton is in a state x_1 at the beginning, at the time t_1. Then an input y_1 enters and the state of the automaton changes into $x_2 = g(x_1, y_1)$ during the time $t_2 = t_1 + \tau$. Then an input y_2 enters, etc. It is supposed that the "operator" can observe only the inputs $y_1, y_2, \ldots$ whereas the states remain hidden (therefore an automaton is also called "black box"). In some cases the mapping g is known as well and thus only the passed states remain unknown.

An automaton can be represented by a table or in a graphic mode.

Example. $\mathbf{X} = \{1, 2, 3\}$, $\mathbf{Y} = \{0, 1\}$. The table of the transition mapping g is the following.

$y \diagdown x$	1	2	3
0	1	3	2
1	2	1	1

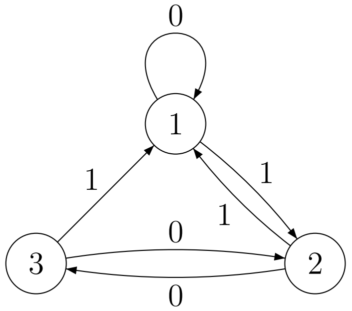

Figure 1

The graph representing the same automaton is in Fig. 1.

In general, the (multi)digraph $G = (V, A)$ of the given automaton can be obtained in the following way. The vertex set equals to the set of states, i.e., $V = X$ and an edge $(x_1, x_2) \in A$ if there exists $y \in Y$ such that $g(x_1, y) = x_2$. Then y is associated with (x_1, x_2) as an additional parameter. It is obvious that Fig. 1 represents a graph with the connecting state 1.

Fig. 2 represents the well known flip-flop circuit.

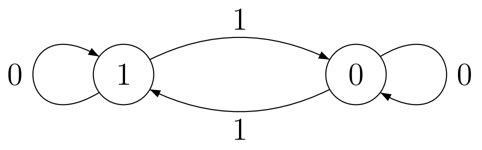

Figure 2

An automaton $T = (X, Y, g)$ is said to be **directable** if there exist

$$n \in N,\ x_0 \in X,\ y^n \in Y^n \text{ such that } g^n(X, y^n) = \{x_0\}. \tag{2}$$

We say that the automaton T is **directable into the state** x_0 if (2) holds.

Obviously, if T is directable then it is connected. Moreover, if it is strongly connected then it is directable into each state $x_0 \in X$, since it is directable into at least one state $x_0 \in X$ and this state is transformable into an arbitrary state because of the strong connectivity of T.

Theorem 1. *Let $T = (X, Y, g)$ be a directable automaton. Then it has at least one connecting state.*

Proof. By contradiction. Let there be no connecting state in the automaton T. Then $g(X, y) = X$ for any $y \in Y$ since g is a one-to-one mapping. Hence for all $n \in N$ and $y^n \in Y^n$ we have $g^n(X, y^n) = X$, a contradiction with (2). The theorem is proved.

One can see that the existence of a connecting state is the necessary condition for the directability of an automaton, but it is not a sufficient condition, even in the case when the automaton is strongly connected. The automaton on the Fig. 1 can be chosen as a counter example. It is obvious that in this case one of the following three relations

$$g^n(x, y^n) = X, \quad g^n(x, y^n) = \{1, 2\}, \quad g^n(x, y^n) = \{1, 3\}$$

holds for any $n \in N$ and $y^n \in Y^n$. Hence the automaton $T = (X, Y, g)$ is not directable.

Theorem 2. Let $T = (X, Y, g)$ be an automaton. T is directable if and only if the following condition (D) holds:

For each $x_1,\ x_2 \in X$ there exist $n \in N,\ y^n \in Y^n$ and $x_3 \in X$ such that

$$g(\{x_1, x_2\},\ y^n) = \{x_3\}. \tag{D}$$

Proof. Let k be the number of elements in the set X. The case $k = 1$ is trivial. Let $k > 1$.

1. If T is directable then there exist $n \in N,\ x_3 \in X,\ y^n \in Y^n$ such that $g(X, y^n) = \{x_3\}$. Hence for each $x_1, x_2 \in X$ we have $g^n(\{x_1, x_2\},\ y^n) = \{x_3\}$, i.e. the condition (D) holds.

2. Let the condition (D) hold. Let us choose arbitrary $x_1,\ \underline{x}_1 \in X,\ x_1 \neq \underline{x}_1$. Following (D), there exist $n_1 \in N,\ x_2 \in X,\ y^{n_1} = (y_{11}, \dots, y_{1n_1}) \in Y^{n_1}$ such that

$$g^{n_1}(\{x_1,\ \underline{x}_1\},\ y^{n_1}) = \{x_2\}$$

which implies that the set $g^{n_1}(X, y^{n_1})$ contains no more than $k-1$ elements. If $g^{n_1}(X, y^{n_1})$ contains only one element the proof is finished. If $g^{n_1}(X, y^{n_1})$ contains another element $\underline{x}_2 \neq x_2$, then there exist $n_2 \in N$, $x_3 \in X$ and such that

$$g^{n_2}(\{x_2,\ \underline{x}_2\},\ y^{n_2}) = \{x_3\},$$

which implies that the set $g^{n_1+n_2}(X, (y^{n_1}, y^{n_2}))$ contains no more than $k-2$ elements. It is obvious that after finite number of steps we obtain y^n such that the set $g^n(X, y^n)$ has only one element and therefore the automaton T is directable, q.e.d.

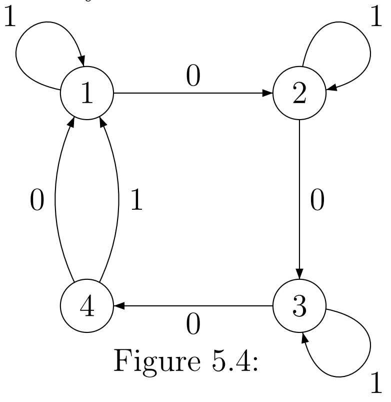

Figure 5.4:

Note. Let $T = (X, Y, g)$ be a given automaton. We say that the automaton $T' = (X', Y', g')$ is **associated to the automaton** T if

1. $X' = X_1 \cup X_2$ where $X_1 = \{(x_1, x_2) : x_1 \in X,\ x_2 \in X,\ x_1 \neq x_2\}$, $X_2 = \{(x, x) : x \in X\}$
2. $Y' = Y$
3. $g'((x_1, x_2), y) = (g(x_1, y),\ g(x_2, y))$ for all $(x_1, x_2) \in X', y \in Y$.

It is an easy consequence of Theorem 2 that T is directable if and only if T' is connected. This fact can be used to decide whether or not is an automaton directable.

Example. Let U_4 be the automaton from the Fig. 3 (the form of the denotation it will be clear later). The associated automaton U_4' is on the Fig. 4, where the states from the subset X_2 are in double circles. One can see that U_4' is connected and consequently U_4 is directable (even into an arbitrary state).

Example. If T is the automaton from the Fig. 1 then T' is on the Fig. 5. One can see that T' is not connected which implies that T is not ditectable (we have already known this fact).

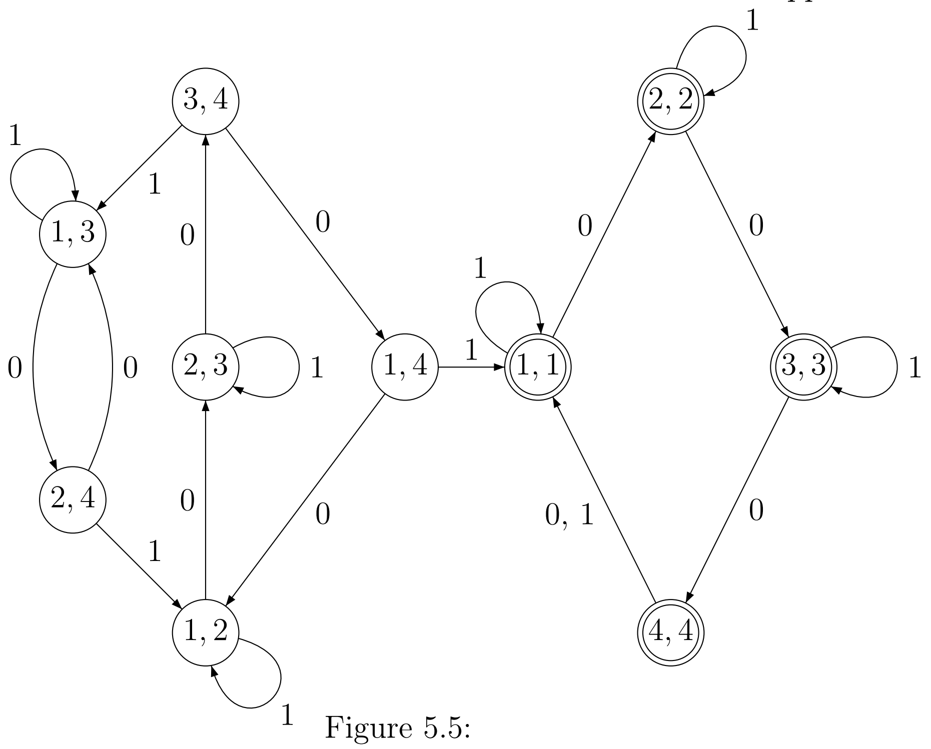

Figure 5.5:

3. Directing Speed

Let Π_k denote the set of all directable automata with k states. Let $k \in N$, $T \in \Pi_k$. Then we denote

$n(T) = \min\{n \in N :$ there exists $x_0 \in X,\ y^n \in Y^n$ such that $g^n(X, y^n) = \{x_0\}\}$

$n(k) = \sup\{n(T) : T \in \Pi_k\}$

If the automaton works in regular time intervals then $n(k)$ expresses the supremum of the number of time intervals being necessary for the directing of the automata from the set Π_k.

In the sequel, we shall introduce the automata U_k. They will help us in finding the lower bound of $n(k)$.

$U_k = (X_k, Y, g_k)$, where $X_k = \{1, 2, \ldots, k\}$, $Y = \{0, 1\}$ and for $x \in X_k$ we have

$$g_k(x, 0) = x + 1,\ g_k(x, 1) = x \text{ for } x < k$$

$$g_k(x, 0) = g_k(x, 1) = 1 \text{ for } x = k.$$

The graphic representation of the automaton U_4 is the Fig. 3.

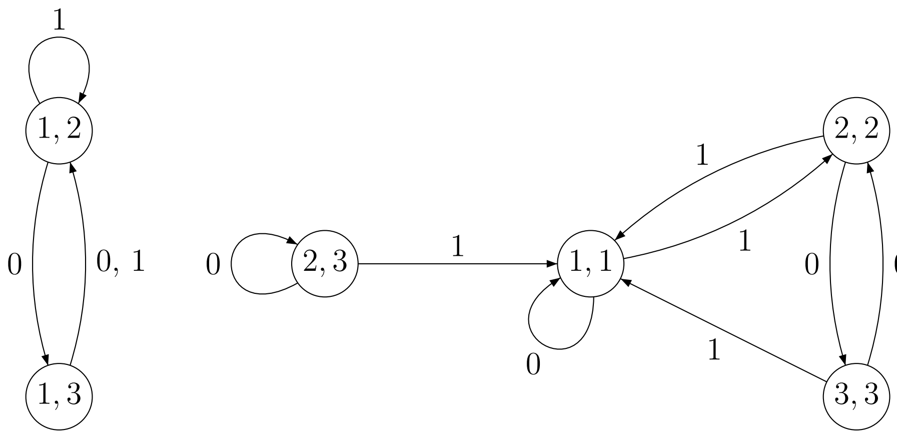

Figure 5.6:

The following lemmas will give some properties of the automata U_k.

Lemma 1. $U_k \in \Pi_k$ and $n(U_k) = (k-1)^2$ for each $k = 2, 3, \ldots$.

Proof. Let us define

$y_j = 1$ for $j = 1 + l.k,\ l = 1, \ldots, k-2$

$y_j = 0$ for $j \neq 1 + l.k,\ j < 1 + k(k-2) = (k-1)^2$

Firstly, we shall prove

$$g_k^{1+lk}(X_k, y_1, \ldots, y_{1+lk}) = \{1, \ldots, k-l-1\} \tag{4}$$

for $l = 1, \ldots, k-2$.

It is evident that (4) holds for $l = 1$.

Let $1 \leq l \leq k-3$ and let (4) hold for l. We shall prove that (4) holds for $l+1$ as well: We have

$$g_k^{(l+1)k}(X_k, y_1, \ldots, y_{(l+1)k}) = g_k^{(l+1)k}(X_k, y_1, \ldots, y_{1+lk}, 0, \ldots, 0) =$$

$$= g_k^{k-1}\{1, \ldots, k-l-1\}, 0, \ldots, 0) = \{1, \ldots, k-l-2, k\}$$

which implies

$$g_k^{1+(l+1)k}(X_k, y_1, \ldots, y_{1+(l+1)k}) = g_k(\{1, \ldots, k-l-2, k\}, 1) = \{1, \ldots, k-(l+1)-1\}$$

and thus (4) holds for $l = 1, \ldots, k-2$. Consequently

$$g_k^{1+(k-2)k}(X_k, y_1, \ldots, y_{1+(k-2)k}) = g_k^{(k-1)^2}(X_k, y_1, \ldots, y_{(k-1)^2}) = \{1\}$$

and $U_k \in \Pi_k$ and
$$n(U_k) \leq (k-1)^2. \tag{5}$$

On the other hand, we define a binary operation $\oplus$ on the set X as follows:
$$x_1 \oplus x_2 = \begin{cases} x_1 + x_2 & \text{for } x_1 + x_2 \leq k, \\ x_1 + x_2 - k & \text{for } x_1 + x_2 > k. \end{cases}$$
(it corresponds to the sum modulo k). For each $X \subset X_k$, $X \neq \varnothing$ define function $\delta(X)$ as follows:

$$\delta(X_k) = 0 \text{ and for } X \neq X_k$$
$$\delta(X) = \max_{x \in X,\ x \oplus 1 \in X_k - X} (\max\{j \in N : x \oplus 1 \in X_k - X, \dots, x \oplus j \in X_k - X\}).$$

($\delta(X)$ expresses the number of elements in the greatest gap in X for X_k ordered in a circle).

Let us denote
$$M_{x,j} = \{x \oplus 1, \dots, x \oplus j\}$$
for $x \in X_k, j \in N, j \leq k$ and let $\tilde{X}$ denotes such a set $M_{x,j}$ that $M_{x,j} \subset X_k - X$ and the number of elements in $M_{x,j}$ is maximum possible. It is evident that the number of elements in the set $\tilde{X}$ denoted by $\text{card}\tilde{X}$ equals to $\delta(X)$.

Evidently $\delta(X_k) = 0$, $\delta(\{x\}) = k - 1$ for any $x \in X_k$ and $\delta(g_k(X, 0)) = \delta(X)$ for each $X \subset X_k$ according to the definition of δ function of the automaton U_k. It is important to determine when the equation
$$\delta(g_k(X, 1)) = \delta(X) + 1$$
holds. It is possible only for $\text{card}\tilde{g}_k(X, 1) = card\tilde{X} + 1$, i.e. if
$$\tilde{X} = \{k - \delta(X), \dots, k - 1\}.$$

Then we have
$$\tilde{g}_k(X, 1) = \{k - \delta(X), \dots, k - 1, k\}$$
Let $n = n(U_k)$ and let $g_k^n(X_k, y^n) = \{x\}$ for $y^n = (y_1, \dots, y_n) \in Y^n$, $x \in X_k$. Obviously $x = 1$ and $y_1 = 1$. Let us denote
$$X(i) = g_k^i(X_k, y_1, \dots, y_i) \quad \text{for } i \in N,\ i \leq n.$$

Let us suppose that $\delta(X(i-1)) = j-1$ and $\delta(X(i)) = j$ for some $i,\ j \in N$. In that case

$$\tilde{X}(i) = \{k-j+1, \ldots, k\}.$$

Let $\delta(X(i+l)) = j+1$ for some l, then $l \geq k$ because the sequence of $k-1$ zeroes is the shortest input sequence transforming the set $\tilde{X}(i)$ into the set $\{k-j, \ldots, k-1\}$ and afterwards the input of $y_{i+k} = 1$ increases the value of $\delta(X(i+k))$ by one. Therefore

$$n = n(U_k) \geq 1 + (k-2)k = (k-1)^2 \tag{6}$$

and (6) together with (5) concludes the proof.

Corollary. $n(k) \geq (k-1)^2$.

Lemma 2. $n(k) \leq 2^k - k - 1$ *for* $k \in N$.

Proof. For $k = 1, 2$ the assertion is trivially holds. Let $k \in N, k > 2$, $T = (X, Y, g) \in \Pi_k,\ n = n(T)$. Let $y^n = (y_1, \ldots, y_n) \in Y^n,\ x \in X$ be such that $g^n(X, y^n) = \{x\}$. Let us denote

$$X(i) = g^i(X, y_1, \ldots, y_i) \quad \text{for } i \in N,\ i \leq n.$$

Since $n = n(T)$ and y^n is the shortest directing sequence we have $X(i) \neq X(j)$ for $i \leq n,\ j \leq n,\ i \neq j$.

Set $X(i)$ contains for $i < n$ at least two and no more than $k-1$ elements. Hence the number of mutually different sets $X(i)$ equals to the number of all subsets of the set X except of the sets $X, \varnothing$ and $k-1$ sets with one element. Consequently

$$n(T) \leq 2^k - k - 1.$$

Since $T \in \Pi_k$ is arbitrary, we have

$$n(k) \leq 2^k - k - 1$$

which concludes the proof.

Theorem 3. *Let k be an arbitrary natural number. Then the following inequality holds:*

$$(k-1)^2 \leq n(k) \leq 2^k - k - 1.$$

Proof. It is a direct consequence of 2 and Corollary of Lemma 1.

Finally, we observe that the upper and lower bounds are equal for $k = 1, 2, 3$ and we have $n(1) = 0$, $n(2) = 1$, $n(3) = 4$ from Theorem 3. For other values of k the difference between the bounds increases rapidly and it would be necessary to precise them. One can expect it mainly as for the upper one.

References

[1] Moore E., *Gedanken-experiments with sequential machines.* Automata studies, Princeton 1956, 129–153

[2] Ginsburg S., *On the length of the smallest uniform experiment which distinguishes the terminal states of a machine*, J. Assoc. Comput. Mach. 5 (1958), 266–280

Appendix 2: Chapter 4 of Liu's 1962 Thesis

Appendix 3: Ashby's problem

The following is an excerpt from pp. 60–61 of Ashby's book 'An Introduction to Cybernetics' [3]. The copyright of the book is held by the Estate of W. Ross Ashby and permits non-profit reproduction and distribution of its text for educational and research reasons.

"**4/15. Materiality**. The reader may now like to test the methods of this chapter as an aid to solving the problem set by the following letter. It justifies the statement made in S.1/2 that cybernetics is not bound to the properties found in terrestrial matter, nor does it draw its laws from them. What is important in cybernetics is the extent to which the observed behaviour is regular and reproducible.

"Graveside"
Wit's End
Haunts.

Dear Friend,

Some time ago I bought this old house, but found it to be haunted by two ghostly noises—a ribald Singing and a sardonic Laughter. As a result it is hardly habitable. There is hope, however, for by actual testing I have found that their behaviour is subject to certain laws, obscure but infallible, and that they can be affected by my playing the organ or burning incense.

In each minute, each noise is either sounding or silent—they show no degrees. What each will do during the ensuing minute depends, in the following exact way, on what has been happening during the preceding minute: The Singing, in the succeeding minute, will go on as it was during the preceding minute (sounding or silent) unless there was organ-playing with no Laughter, in which case it will change to the opposite (sounding to silent, or vice versa). As for the Laughter, if there was incense burning, then it will sound or not *according as the Singing was sounding or not (so that the Laughter copies the Singing a minute later). If however there was no incense burning, the Laughter will do the opposite of what the Singing did.*

At this minute of writing, the Laughter and Singing are both sounding. Please tell me what manipulations of incense and organ I should make to get the house quiet, and to keep it so."

Here is the solution to the puzzle provided in [3, p. 277]:

"If the variables are S = Singing, L = Laughter, X = Organ-playing, Y = Incense-burning, and each take the values 0 or 1 for inactive or active respectively, then the machine with input is soon found to be

		(S, L)			
	↓	00	01	10	11
	00	01	01	10	10
(X, Y)	01	00	00	11	11
	10	11	01	00	10
	11	10	00	01	11

One way to (0,0) is: Stop the incense burning for one minute; next stop the incense and play the organ; finally, start burning the incense again; in future keep it burning and never play the organ."

List of Figures

List of Tables

List of Notation

Index

Author Index

Bibliography

[1] R. L. Adler, L. W. Goodwyn, and B. Weiss. Equivalence of topological Markov shifts. *Israel J. Math*, 27(1):49–63, 1977. 36, 37, 39

[2] R. L. Adler and B. Weiss. Similarity of automorphisms of the torus. *Memoirs Amer. Math. Soc.*, 98, 1970. 36

[3] W. R. Ashby. *An introduction to cybernetics*. Chapman & Hall, 1956. 18, 57, 58

[4] S. Bogdanović, B. Imreh, M. Ćirić, and T. Petković. Directable automata and their generalizations: a survey. *Novi Sad J. Math.*, 29(2):29–69, 1999. 28

[5] R. M. Capocelli, L. Gargano, and U. Vaccaro. On the characterization of statistically synchronizable variable-length codes. *IEEE Transactions on Information Theory*, 34(4):817–825, 1988. 22

[6] A. Carbone. Cycles of relatively prime length and the road coloring problem. *Israel J. Math.*, 123:303–316, 2001. 38

[7] J. Černý. Poznámka k homogénnym eksperimentom s konečnými automatami. *Matematicko-fyzikalny Časopis Slovenskej Akadémie Vied*, 14(3):208–216, 1964. (in Slovak). 16, 17, 19, 45

[8] J. Černý, A. Pirická, and B. Rosenauerová. On directable automata. *Kybernetika*, 7(4):289–298, 1971. 16

[9] K. Culik II, J. Karhumäki, and J. Kari. A note on synchronized automata and Road Coloring Problem. *Int. J. Found. Comput. Sci.*, 13:459–471, 2002. 40, 41

[10] N. G. de Bruijn. A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen*, 49:758–764, 1946. 16

[11] F. M. Dekking. The spectrum of dynamical systems arising from substitutions of constant length. *Z. Wahrsch. Verw. Gebiete*, 41:221–239, 1978. 27

[12] J. Friedman. On the road coloring problem. *Proc. Amer. Math. Soc.*, 110:1133–1135, 1990. 38

[13] A. Gill. State-identification experiments in finite automata. *Inform. Control*, 4(2-3):132–154, 1961. 17

[14] S. Ginsburg. On the length of the smallest uniform experiment which distinguishes the terminal states of a machine. *J. Assoc. Comput. Mach.*, 5:266–280, 1958. 17

[15] I. J. Good. Normal recurring decimals. *J. London Math. Soc.*, 21(3):167–169, 1946. 16

[16] N. Jonoska and S. Suen. Monocyclic decomposition of graphs and the road coloring problem. *Congressum numerantium*, 110:201–209, 1995. 38

[17] J. Kari. A counter example to a conjecture concerning synchronizing words in finite automata. *Bull. European Assoc. Theor. Comput. Sci.*, 73:146, 2001. 16

[18] J. Kari. Synchronization and stability of finite automata. *J. Universal Comp. Sci.*, 2:270–277, 2002. 38

[19] J. Kari. Synchronizing finite automata on Eulerian digraphs. *Theoret. Comput. Sci.*, 295:223–232, 2003. 38

[20] A. E. Laemmel. Study on application of coding theory. Technical Report PIBMRI-895.5-63, Dept. Electrophysics, Microwave Research Inst., Polytechnic Inst. Brooklyn, NY, 1963. 16, 19

[21] A. E. Laemmel and B. Rudner. Study of the application of coding theory. Technical Report PIBEP-69-034, Dept. Electrophysics, Polytechnic Inst. Brooklyn, Farmingdale, NY, 1969. 19

[22] C. L. Liu. Determination of the final state of an automaton whose initial state is unknown. *IEEE Transactions on Electronic Computers*, EC-12(5):918–920, 1963. 19

[23] C. L. Liu. Some memory aspects of finite automata. Technical Report 411, Research Lab. Electronics, Massachusetts Inst. Technology, Cambridge, MA, 1963. 16, 17, 19

[24] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.*, 5:115–133, 1943. 8

[25] E. F. Moore. Gedanken-experiments on sequential machines. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, 1956. 17

[26] B. K. Natarajan. An algorithmic approach to the automated design of parts orienters. In *Proc. 27th Annual Symp. Foundations Comput. Sci.*, pages 132–142. IEEE Press, 1986. 23

[27] B. K. Natarajan. Some paradigms for the automated design of parts feeders. *Internat. J. Robotics Research*, 8(6):89–109, 1989. 23

[28] G. L. O'Brien. The road coloring problem. *Israel J. Math.*, 39:145–154, 1981. 38

[29] M. Perles, M. O. Rabin, and E. Shamir. The theory of definite automata. *IEEE Transactions on Electronic Computers*, EC-12(3):233–243, 1963. 16

[30] D. Perrin and M. P. Schützenberger. Synchronizing prefix codes and automata and the road coloring problem. In *Symbolic dynamics and its applications*, volume 135 of *Contemporary Mathematics*, pages 295II–318. Amer. Math. Soc., 1992. 38

theasFogg:2002 [31] N. Pytheas Fogg. *Substitutions in dynamics, arithmetics and combinatorics*, volume 1794 of *Lecture Notes in Mathematics*. Springer-Verlag, 2002. Edited by V. Berthé, S. Ferenczi, C. Mauduit and A. Siegel. 27

Roman:2008 [32] A. Roman. A note on Černý conjecture for automata over 3-letter alphabet. *J. Automata, Languages, and Combinatorics*, 13(2):141–143, 2008. 16

zenberger:1956 [33] M.-P. Schützenberger. On an application of semi groups methods to some problems in coding. *IRE Transactions on Information Theory*, 2(3):47–60, 1956. 17, 18

Shannon:1948 [34] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948. 21

taStudies:1956 [35] C. E. Shannon and J. McCarthy, editors. *Automata Studies*. Princeton University Press, 1956. 8

Trahtman:2006 [36] A. Trahtman. An efficient algorithm finds noticeable trends and examples concerning the Černý conjecture. In R. Královič and P. Urzyczyn, editors, *31st Int. Symp. Math. Foundations of Comput. Sci.*, volume 4162 of *Lecture Notes in Comput. Sci.*, pages 789–800. Springer-Verlag, 2006. 16

Trahtman:2009 [37] A. Trahtman. The Road Coloring Problem. *Israel J. Math.*, 172(1):51–60, 2009. 38

Turing:1936 [38] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc., Ser. 2*, 42:230–265, 1936. 7