Černý's conjecture and the road coloring problem

Jarkko Kari[1], *Mikhail Volkov*[2]

[1]Department of Mathematics
FI-20014 University of Turku
Turku, Finland

[2]Department of Mathematics and Mechanics
620083 Ural State University
Ekaterinburg, Russia
email: jkari@utu.fi, Mikhail.Volkov@usu.ru

January 10, 2011 1 h 23

chapterKV

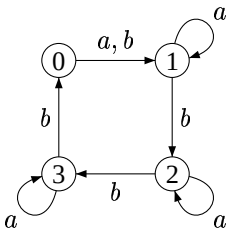
2010 Mathematics Subject Classification: 68Q45 68R10

Key words: Finite automata, Synchronizing automata, Reset words, Černý's conjecture, Road Coloring Problem

Contents

1 Synchronizing automata, their origins and importance

A complete deterministic finite automaton (DFA) $\mathcal{A} = (Q, A)$ (here and below Q stands for the state set and A for the input alphabet) is called *synchronizing* if there exists a word $w \in A^*$ whose action resets $\mathcal{A}$, that is, w leaves the automaton in one particular state no matter at which state in Q it is applied: $q \cdot w = q' \cdot w$ for all $q, q' \in Q$. Any word w with this property is said to be a *reset* word for the automaton.



KV:fig:C4

Figure 1. The automaton $\mathcal{C}_4$

KV:fig:C4

Figure 1 shows a synchronizing automaton with 4 states denoted by $\mathcal{C}_4$. The reader can easily verify that the word ab^3ab^3a resets the automaton leaving it in the state 1. With somewhat more effort one can also check that ab^3ab^3a is the shortest reset word for $\mathcal{C}_4$. The example in Figure 1 is due to KV:fig:C4 Černý, a Slovak computer scientist, in whose pioneering paper [8] the notion of a synchronizing automaton explicitly appeared for the first time. (Černý called such automata *directable*. The word *synchronizing* in this context was probably introduced by Hennie [21].) Implicitly, however, this concept has been around since the earliest days of automata theory. The very first synchronizing automaton that we were able to trace back in the literature appeared in Ashby's classic book [2, pp. 60–61], see [41, Section 1] for a discussion.

Needs double-checking!!

In [8] the notion of a synchronizing automaton arose within the classic framework of Moore's "Gedanken-experiments" [23]. 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 [23] 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. Ginsburg [18] considered more restricted experiments that he called *uniform*. A uniform experiment[1] 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.)

The original "Gedanken-experiments" motivation for studying synchronizing automata is still of importance, and reset words are frequently applied in model-based testing of reactive systems. See [10, 6] as typical samples of technical contributions to the area and [38] for a recent survey.

Another strong motivation comes from the coding theory. We refer to [4, Chapters 3 and 10] for a detailed account of profound connections between codes and automata; here

[1] After [17], the name *homing sequence* has become standard for the notion.

we restrict ourselves to a special (but still very important) case of maximal prefix codes. Recall that a *prefix code* over a finite alphabet A is a set X of words in A^* such that no word of X is a prefix of another word of X. A prefix code is *maximal* if it is not contained in another prefix code over the same alphabet. A maximal prefix code X over A is *synchronized* if there is a word $x \in X^*$ such that for any word $w \in A^*$, one has $wx \in X^*$. Such a word x 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 $v \in A^*$ contains a fixed factor x tends to 1 as the length of v increases, synchronized codes eventually resynchronize by themselves, after sufficiently many symbols being sent. (As shown in [7], the latter property in fact characterizes synchronized codes.) The following simple example illustrates these ideas: let $A = \{0, 1\}$ and $X = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$. Then X is a maximal prefix code and one can easily check that each of the words 010, 011110, 011111110, ... is a synchronizing word for X. For instance, if the code word 000 has been sent but, due to a channel error, the word 100 has been received, the decoder interprets 10 as a code word, and thus, loses synchronization. However, with a high probability this synchronization loss only propagates for a short while; in particular, the decoder definitely resynchronizes as soon as it encounters one of the segments 010, 011110, 011111110, ... in the received stream of symbols. A few samples of such streams are shown in Figure 2 in which vertical lines show the partition of each stream into code words and the boldfaced code words indicate the position at which the decoder resynchronizes.

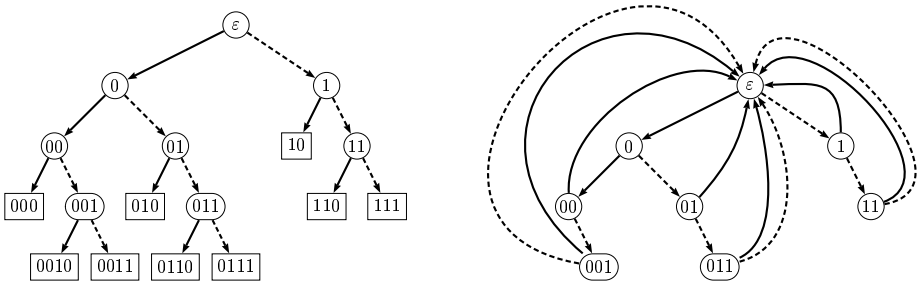
Sent	0 0 0 | 0 0 1 0 | **0 1 1 1** | ...
Received	1 0 | 0 0 0 | 1 0 | **0 1 1 1** | ...
Sent	0 0 0 | 0 1 1 1 | 1 1 0 | 0 0 1 1 | 0 0 0 | 1 0 | **1 1 0** | ...
Received	1 0 | 0 0 1 1 | 1 1 1 | 0 0 0 | 1 1 0 | 0 0 1 0 | **1 1 0** | ...
Sent	0 0 0 | 0 0 0 | 1 1 1 | **1 0** | ...
Received	1 0 | 0 0 0 | 0 1 1 1 | **1 0** | ...

Figure 2. Restoring synchronization

If X is a finite prefix code over an alphabet A, then its decoding can be implemented by a deterministic 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 $a \in A$, define

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

The resulting automaton $\mathcal{A}_X$ is complete whenever the code X is maximal and it is easy to see that $\mathcal{A}_X$ 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 $\mathcal{A}_X$ and sends all states in Q to the state ε. Figure 3 illustrates this construction for the code $X = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$ considered above. The solid/dashed lines correspond to (the action of) 0/1.



:fig:huffman

Figure 3. A synchronized code (on the left) and its automaton (on the right)

Thus, **(to be continued and supplied by some historical references).**

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 again rediscovered in the mid-1980s by Natarajan [24, 25] who showed how synchronizing automata can be used to design sensor-free orienters for polygonal parts, see [41, Section 1] for a transparent example illustrating Natarajan's approach in a nutshell. Since the 1990s synchronizing automata usage in the area of robotic manipulation has grown into a prolific research direction but it is fair to say that publications in this area deal mostly with implementation technicalities. However, amongst them there are papers of significant theoretical importance such as [13, 19, 9].

To be checked with Jarrko!!

Recently, it has been realized that a notion that arose in studying of *substitution systems* is also closely related to synchronizing automata. A *substitution* on a finite alphabet X is a map $\sigma : X \to X^+$; 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 m-th letter. For an example, consider the substitution τ on $X = \{0, 1, 2\}$ defined by $0 \mapsto 11,\ 1 \mapsto 12,\ 2 \mapsto 20$. Calculating the iterations of τ up to τ^4 (see Figure 4), we observe that τ satisfies the coincidence condition (with $k = 4, m = 7$).

kv:fig:substituion

$$
\begin{array}{lllllllll}
0 & \mapsto & 11 & \mapsto & 1212 & \mapsto & 12201220 & \mapsto & 12202011\mathbf{1}12202011 \\
1 & \mapsto & 12 & \mapsto & 1220 & \mapsto & 12202011 & \mapsto & 12202011\mathbf{2}0111212 \\
2 & \mapsto & 20 & \mapsto & 2011 & \mapsto & 20111212 & \mapsto & 20111212\mathbf{1}2201220
\end{array}
$$

:substituion

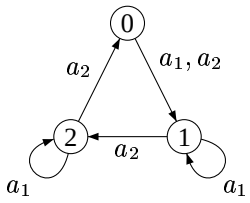
Figure 4. A substitution satisfying the coincidence condition

The importance of the coincidence condition comes from the crucial fact (established by Dekking [12]) that it is this condition that completely characterizes the constant length substitutions which give rise to dynamical systems measure-theoretically isomorphic to a translation on a compact Abelian group, see [30, 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 $\mathcal{A} = (Q, A)$ with $A = \{a_1, \ldots, a_\ell\}$ defines a length ℓ substitu-

tion on Q that maps every $q \in Q$ to the word $(q \cdot a_1) \dots (q \cdot a_\ell) \in Q^+$. (For instance, the automaton $\mathcal{C}_4$ in Figure 1 induces the substitution $0 \mapsto 11,\ 1 \mapsto 12,\ 2 \mapsto 23,\ 3 \mapsto 30$.) Conversely, each substitution $\sigma : X \to X^+$ 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 5.) It is clear that under the described bijection

KV:fig:C4

KV:fig:C3



KV:fig:C3

Figure 5. The automaton induced by the substitution $0 \mapsto 11,\ 1 \mapsto 12,\ 2 \mapsto 20$

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.

We mention in passing a purely algebraic framework within which synchronizing automata also appear in a natural way. One may treat DFAs as unary algebras since each letter of the input alphabet defines a unary operation on the state set. A *term* in the language of such unary algebras is an expression t of the form $x \cdot w$, where x is a variable and w is a word over an alphabet A. An *identity* is a formal equality between two terms. A DFA $\mathcal{A} = (Q, A)$ *satisfies* an identity $t_1 = t_2$, where the words involved in the terms t_1 and t_2 are over A, 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 from either $x \cdot u = x \cdot v$ (*homotypical* identities) or $x \cdot u = y \cdot v$ 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. See [5] 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.

If space permits!!

2 Algorithmic and complexity issues

It should be clear that not every DFA is synchronizing. Therefore, the very first question that we should address is the following one: *given an automaton $\mathcal{A}$, how to determine whether or not $\mathcal{A}$ is synchronizing?*

This question is in fact quite easy, and the most straightforward solution to it can be achieved via the classic subset construction by Rabin and Scott [31]. Given a DFA $\mathcal{A} = (Q, A)$, we define its *subset automaton* $\mathcal{P}(\mathcal{A})$ on the set of the non-empty subsets of

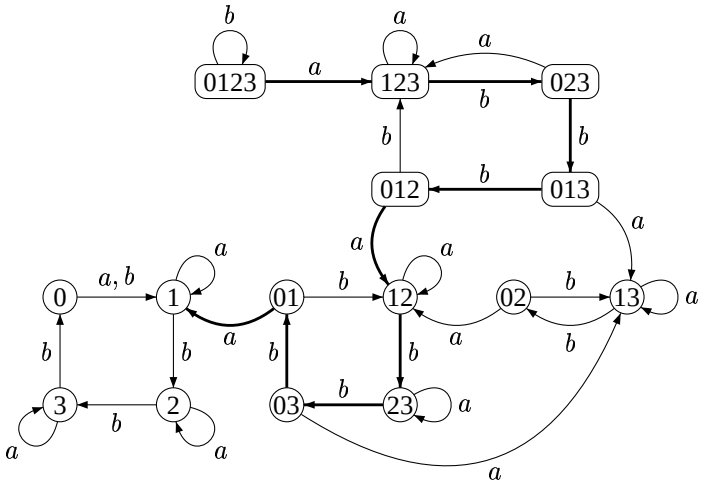


Figure 6. The power automaton $\mathcal{P}(\mathcal{C}_4)$

Q by setting $P \cdot a = \{p \cdot a \mid p \in P\}$ for each non-empty subset P of Q and each $a \in \mathcal{A}$. (Since we start with a deterministic automaton, we do not need adding the empty set to the state set of $\mathcal{P}(\mathcal{A})$.) Figure 6 presents the subset automaton for the DFA $\mathcal{C}_4$ shown in Figure 1.

Now it is obvious that a word $w \in A^*$ is a reset word for the DFA $\mathcal{A}$ if and only if w labels a path in $\mathcal{P}(\mathcal{A})$ starting at Q and ending at a singleton. (For instance, the bold path in Figure 6 represents the shortest reset word ab^3ab^3a of the automaton $\mathcal{C}_4$.) Thus, the question of whether or not a given DFA $\mathcal{A}$ is synchronizing reduces to the following reachability question in the underlying digraph of the subset automaton $\mathcal{P}(\mathcal{A})$: is there a path from Q to a singleton? The latter question can be easily answered by breadth-first search, see, e.g., [11, Section 22.2].

The described procedure is conceptually very simple but rather inefficient because the power automaton $\mathcal{P}(\mathcal{A})$ is exponentially larger than $\mathcal{A}$. However, the following criterion of synchronizability [8, Theorem 2] gives rise to a polynomial algorithm.

Proposition 2.1. *A DFA $\mathcal{A} = (Q, A)$ is synchronizing if and only if for every $q, q' \in Q$ there exists a word $w \in A^*$ such that $q \cdot w = q' \cdot w$.*

One can treat Proposition 2.1 as a reduction of the synchronizability problem to a reachability problem in the subautomaton $\mathcal{P}^{[2]}(\mathcal{A})$ of $\mathcal{P}(\mathcal{A})$ whose states are 2-element and 1-element subsets of Q. Since the subautomaton has $\dfrac{|Q|(|Q|+1)}{2}$ states, breadth-first search solves this problem in $O(|Q|^2 \cdot |A|)$ time. This complexity bound assumes that no reset word is explicitly calculated. If one requires that, whenever $\mathcal{A}$ turns out to be synchronizing, a reset word is produced, then the best of the known algorithms (which is basically due to Eppstein [13, Theorem 6], see also [38, Theorem 1.15]) has an implementation that consumes $O(|Q|^3 + |Q|^2 \cdot |A|)$ time and $O(|Q|^2 + |Q| \cdot |A|)$ working space, not counting the space for the output which is $O(|Q|^3)$.

For a synchronizing automaton, the subset automaton can be used to construct shortest reset words which correspond to shortest paths from the whole state set Q to a singleton. Of course, this requires exponential (of $|Q|$) time in the worst case. Nevertheless, there were attempts to implement this approach, see, e.g., [33, 40]. One may hope that, as above, a suitable calculation in the "polynomial" subautomaton $\mathcal{P}^{[2]}(\mathcal{A})$ may yield a polynomial algorithm. However, it is not the case, and moreover, as we will see, it is very unlikely that any reasonable algorithm may exist for finding shortest reset words in general synchronizing automata. In the following discussion we assume the reader's acquaintance with some basics of computational complexity (such as the definitions of the complexity classes NP and coNP) that can be found, e.g., in [15, 27].

Consider the following decision problem:

SHORT-RESET-WORD: *Given a synchronizing automaton $\mathcal{A}$ and a positive integer ℓ, is it true that $\mathcal{A}$ has a reset word of length ℓ?*

Clearly, SHORT-RESET-WORD belongs to the complexity class NP: one can nondeterministically guess a word $w \in A^*$ of length ℓ and then check if w is a reset word for $\mathcal{A}$ in time $\ell|Q|$. Several authors [35, 13, 20, 36, 37] have proved that SHORT-RESET-WORD is NP-hard by a polynomial reduction from SAT (the satisfiability problem for a system of *clauses*, that is, disjunctions of literals). We reproduce here Eppstein's reduction from [13].

Given an arbitrary instance ψ of SAT with n variables $x_1, \dots, x_n$ and m clauses $c_1, \dots, c_m$, we construct a DFA $\mathcal{A}(\psi)$ with 2 input letters a and b as follows. The state set Q of $\mathcal{A}(\psi)$ consists of $(n+1)m$ states $q_{i,j}$, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n+1$, and a special state z. The transitions are defined by

$$q_{i,j} \cdot a = \begin{cases} z \text{ if the literal } x_j \text{ occurs in } c_i, \\ q_{i,j+1} \text{ otherwise} \end{cases} \qquad \text{for } 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n;$$

$$q_{i,j} \cdot b = \begin{cases} z \text{ if the literal } \neg x_j \text{ occurs in } c_i, \\ q_{i,j+1} \text{ otherwise} \end{cases} \qquad \text{for } 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n;$$

$$q_{i,n+1} \cdot a = q_{i,n+1} \cdot b = z \qquad \text{for } 1 \leqslant i \leqslant m;$$

$$z \cdot a = z \cdot b = z.$$

Figure 7 shows two automata of the form $\mathcal{A}(\psi)$ build for the SAT instances

$$\psi_1 = \{x_1 \vee x_2 \vee x_3,\ \neg x_1 \vee x_2,\ \neg x_2 \vee x_3,\ \neg x_2 \vee \neg x_3\},$$
$$\psi_2 = \{x_1 \vee x_2,\ \neg x_1 \vee x_2,\ \neg x_2 \vee x_3,\ \neg x_2 \vee \neg x_3\}.$$

If at some state $q \in Q$ in Figure 7 there is no outgoing edge labelled $c \in \{a, b\}$, the edge $q \xrightarrow{c} z$ is assumed (those edges are omitted to improve readability). The two instances differ only in the first clause: in ψ_1 it contains the literal x_3 while in ψ_2 it does not. Correspondingly, the automata $\mathcal{A}(\psi_1)$ and $\mathcal{A}(\psi_2)$ differ only by the outgoing edge labelled a at the state $q_{1,3}$: in $\mathcal{A}(\psi_1)$ it leads to z (and therefore, it is not shown) while in $\mathcal{A}(\psi_2)$ it leads to the state $q_{1,4}$ and is shown by the dashed line.

Observe that ψ_1 is satisfiable for the truth assignment $x_1 = x_2 = 0, x_3 = 1$ while ψ_2 is not satisfiable. It is not hard to check that the word bba resets $\mathcal{A}(\psi_1)$ while $\mathcal{A}(\psi_2)$ is reset by no word of length 3 but by every word of length 4.

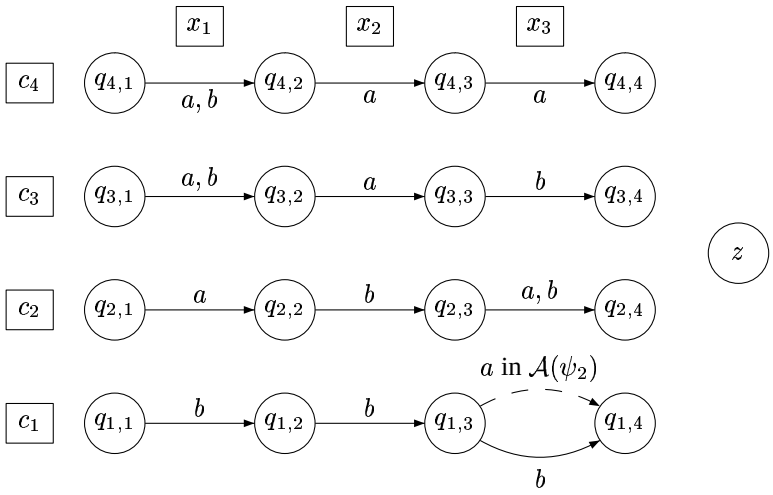


Figure 7. The automata $\mathcal{A}(\psi_1)$ and $\mathcal{A}(\psi_2)$

g:A2_example

In general, it is easy to see that $\mathcal{A}(\psi)$ is reset by every word of length $n+1$ and is reset by a word of length n if and only if ψ is satisfiable. Therefore assigning the instance $(\mathcal{A}(\psi), n)$ of SHORT-RESET-WORD to an arbitrary n-variable instance ψ of SAT, one obtains a polynomial reduction of the latter problem to the former. Since SAT is NP-complete and SHORT-RESET-WORD lies in NP, we obtain the following.

:complexity1 **Proposition 2.2.** *The problem* SHORT-RESET-WORD *is NP-complete.*

In fact, as observed by Samotij [37], the above construction yields slightly more[2]. Consider the following decision problem:

SHORTEST-RESET-WORD: *Given a synchronizing automaton $\mathcal{A}$ and a positive integer ℓ, is it true that the minimum length of a reset word for $\mathcal{A}$ is equal to ℓ?*

Assigning the instance $(\mathcal{A}(\psi), n+1)$ of SHORTEST-RESET-WORD to an arbitrary system ψ of clauses on n variables, one sees that the answer to the instance is "Yes" if and only if ψ is not satisfiable. Thus, we have a polynomial reduction from the negation of SAT to SHORTEST-RESET-WORD whence the latter problem is coNP-hard. As a corollary, SHORTEST-RESET-WORD cannot belong to NP unless NP = coNP which is commonly considered to be very unlikely. In other words, even non-deterministic algorithms cannot decide the minimum length of a reset word for a given synchronizing automaton in polynomial time.

The exact complexity of the problem SHORTEST-RESET-WORD has been recently determined by Gawrychowski [16] and, independently, by Olschewski and Ummels [26]. It turns out that the appropriate complexity class is DP (Difference Polynomial-Time) introduced by Papadimitriou and Yannakakis [28]; this class consists of languages of the form $L_1 \cap L_2$ where L_1 is a language from NP and a L_2 is a language in coNP.

[2] Actually, the reduction in [37] is not correct but the result claimed can be easily recovered as shown below.

A "standard" DP-complete problem is SAT-UNSAT whose instance is a pair of clause systems ψ, χ, say, and whose question is whether ψ is satisfiable and χ is unsatisfiable.

Proposition 2.3. *The problem* SHORTEST-RESET-WORD *is DP-complete.*

Proposition 2.3 follows from mutual reductions between SHORTEST-RESET-WORD and SAT-UNSAT obtained in [16, 26].

The complexity class $\mathsf{P}^{\mathsf{NP}[\log]}$ is defined as the class of all problems that can be solved by a deterministic polynomial-time Turing machine that has an access to an oracle for an NP-complete problem, with the number of queries being logarithmic in the size of the input. The class DP is contained in $\mathsf{P}^{\mathsf{NP}[\log]}$ (if fact, for every problem in DP two oracle queries suffice) and the inclusion is believed to be strict. Olschewski and Ummels [26] have shown that the problem of computing the minimum length of reset words (as opposed to deciding whether it is equal to a given integer) is complete for the functional analogue $\mathsf{FP}^{\mathsf{NP}[\log]}$ of the class $\mathsf{P}^{\mathsf{NP}[\log]}$ (see [39] for a discussion of functional complexity classes). Hence, this problem appears to be even harder than deciding the minimum length of reset words. Recently Berlinkov [3] has shown (assuming $\mathsf{P} \neq \mathsf{NP}$) that no polynomial algorithm can approximate the minimum length of reset words for a given synchronizing automaton within a constant factor.

The problem of finding a reset word of minimum length (as opposed to computing only the length without writing down the word itself) may be even more difficult. From the quoted result of [26] it follows that the problem is $\mathsf{FP}^{\mathsf{NP}[\log]}$-hard but its exact complexity is not known yet.

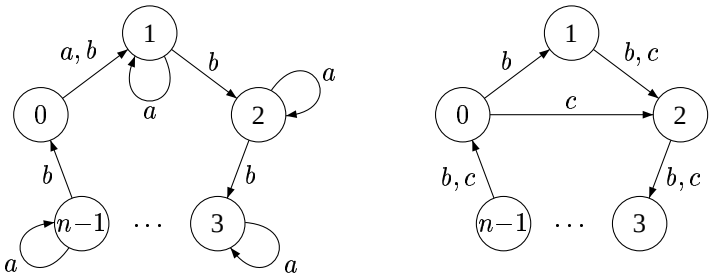
We mention that Pixley, Jeong and Hachtel [29] suggested an heuristic polynomial algorithm for finding short reset words in synchronizing automata that was reported to perform rather satisfactory on a number of benchmarks from [43]; further polynomial algorithms yielding short (though not necessarily shortest) reset words have been implemented by Trahtman [40] and Roman [34]. Some algorithms for finding reset words will be also discussed in the next section.

3 The Černý conjecture

A very natural question to ask is the following: *given a positive integer n, how long can be reset words for synchronizing automata with n states?* Černý [8] found a lower bound by constructing, for each $n > 1$, a synchronizing automaton $\mathcal{C}_n$ with n states and 2 input letters whose shortest reset word has length $(n-1)^2$. We assume that the state set of $\mathcal{C}_n$ is $Q = \{0, 1, 2, \ldots, n-1\}$ and the input letters are a and b, subject to the following action on Q:

$$i \cdot a = \begin{cases} i & \text{if } i > 0, \\ 1 & \text{if } i = 0; \end{cases} \qquad i \cdot b = i + 1 \pmod{n}.$$

Our first example of synchronizing automaton (see Figure 1) is, in fact, $\mathcal{C}_4$. A generic automaton $\mathcal{C}_n$ is shown in Figure 3 on the left.



:fig:cerny-n

Figure 8. The DFA $\mathcal{C}_n$ and the DFA $\mathcal{W}_n$ induced by the actions of b and $c = ab$

The series $\mathcal{C}_n$ was rediscovered many times (see, e.g., [22, 14, 13]). It is easy to see that the word $(ab^{n-1})^{n-2}a$ of length $n(n-2)+1 = (n-1)^2$ is a reset word for $\mathcal{C}_n$. There are several nice proofs for Černý's result [8, Lemma 1] that $\mathcal{C}_n$ has no shorter reset words. Here we present a recent proof from [1]; it is based on a transparent idea and reveals an interesting connection between Černý's automata $\mathcal{C}_n$ and an extremal series of digraphs discovered in Wielandt's classic paper [42] (see Section 4).

KV:sec:rcp

Let w be a reset word of minimum length for $\mathcal{C}_n$. Since the letter b acts on Q as a cyclic permutation, the word w cannot end with b. (Otherwise removing the last letter gives a shorter reset word.) Thus, we can write w as $w = w'a$ for some $w' \in \{a, b\}^*$ such that the image of Q under the action of w' is precisely the set $\{0, 1\}$.

Since the letter a fixes each state in its image $\{1, 2, \ldots, n-1\}$, every occurrence of a in w except the last one is followed by an occurrence of b. (Otherwise a^2 occurs in w as a factor and reducing this factor to just a results in a shorter reset word.) Therefore, if we let $c = ab$, then the word w' can be rewritten into a word v over the alphabet $\{b, c\}$. The actions of b and c induce a new DFA on the state set Q; we denote this induced DFA (shown in Figure 8 on the right) by $\mathcal{W}_n$. Since w' and v act on Q in the same way, the word vc is a reset word for $\mathcal{W}_n$ and brings the automaton to the state 2.

KV:fig:cerny-n

If $u \in \{b, c\}^*$, the word uvc also is a reset word for $\mathcal{W}_n$ and it also brings the automaton to 2. Hence, for every $\ell \geqslant |vc|$, there is a path of length ℓ in $\mathcal{W}_n$ from any given state i to 2. In particular, setting $i = 2$, we conclude that for every $\ell \geqslant |w|$ there is a cycle of length ℓ in $\mathcal{W}_n$. The underlying digraph of $\mathcal{W}_n$ has simple cycles only of two lengths: n and $n-1$. Each cycle of $\mathcal{W}_n$ must consist of simple cycles of these two lengths whence each number $\ell \geqslant |w|$ must be expressible as a non-negative integer combination of n and $n-1$. Here we invoke the following well-known and elementary result from arithmetics:

ma:sylvester

Lemma 3.1 ([32, Theorem 2.1.1]). *If k_1, k_2 are relatively prime positive integers, then $k_1k_2 - k_1 - k_2$ is the largest integer that is not expressible as a non-negative integer combination of k_1 and k_2.*

KV:lemma:sylvester

Lemma 3.1 implies that $|vc| > n(n-1) - n - (n-1) = n^2 - 3n + 1$. Suppose that $|vc| = n^2 - 3n + 2$. Then there should be a path of this length from the state 1 to the state 2. Every outgoing edge of 1 leads to 2, and thus, in the path it must be followed by a cycle of length $n^2 - 3n + 1$. No cycle of such length may exist by Lemma 3.1. Hence $|vc| \geqslant n^2 - 3n + 3$.

KV:lemma:sylvester

Since the action of b on any set S of states cannot change the cardinality of S and the action of c can decrease the cardinality by 1 at most, the word vc must contain at least $n-1$ occurrences of c. Hence the length of v over $\{b,c\}$ is at least n^2-3n+2 and v contain at least $n-2$ occurrences of c. Since each occurrence of c in v corresponds to an occurrence of the factor ab in w', we conclude that the length of w' over $\{a,b\}$ is at least $n^2-3n+2+n-2=n^2-2n$. Thus, $|w|=|w'a|\geqslant n^2-2n+1=(n-1)^2$.

4 The road coloring problem

KV:sec:rcp

5 Generalizations

References

[1] D. Ananichev, V. Gusev, and M. Volkov. Slowly synchronizing automata and digraphs. In P. Hliněný and A. Kučera, editors, *Mathematical Foundations of Computer Science, MFCS'10*, volume 6281 of *Lecture Notes in Comput. Sci.*, pages 55–64. Springer-Verlag, 2010. 10

[2] W. R. Ashby. *An introduction to cybernetics.* Chapman & Hall, 1956. 2

[3] M. Berlinkov. Approximating the minimum length of synchronizing words is hard. In F. Ablayev and E. W. Mayr, editors, *Computer Science in Russia, CSR'10*, volume 6072 of *Lecture Notes in Comput. Sci.* Springer-Verlag, 2010. 9

[4] J. Berstel, D. Perrin, and C. Reutenauer. *Codes and automata.* Number 129 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2009. 2

[5] 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. Proc. 8th Int. Conf. "Algebra & Logic" (Novi Sad, 1998). 5

[6] V. Boppana, S. Rajan, K. Takayama, and M. Fujita. Model checking based on sequential ATPG. In *Computer Aided Verification, Proc. 11th International Conference*, volume 1622 of *Lecture Notes in Comput. Sci.*, pages 418–430. Springer-Verlag, 1999. 2

[7] 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. 3

[8] J. Černý. Poznámka k homogénnym eksperimentom s konečnými automatami. *Matematicko-fyzikalny Časopis Slovenskej Akadémie Vied*, 14(3):208–216, 1964. (in Slovak). 2, 6, 9, 10

[9] Y.-B. Chen and D. J. Ierardi. The complexity of oblivious plans for orienting and distinguishing polygonal parts. *Algoritmica*, 14:367–397, 1995. 4

[10] H. Cho, S.-W. Jeong, F. Somenzi, and C. Pixley. Synchronizing sequences and symbolic traversal techniques in test generation. *J. Electronic Testing*, 4:19–31, 1993. 2

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press and McGraw-Hill, 2001. 6

[12] F. M. Dekking. The spectrum of dynamical systems arising from substitutions of constant length. *Z. Wahrsch. Verw. Gebiete*, 41:221–239, 1978. 4

[13] D. Eppstein. Reset sequences for monotonic automata. *SIAM J. Comput.*, 19:500–510, 1990. 4, 6, 7, 10

[14] M. A. Fischler and M. Tannenbaum. Synchronizing and representation problems for sequential machines with masked outputs. In *Proc. 11th Annual Symp. Foundations Comput. Sci*, pages 97–103. IEEE Press, 1970. 10

[15] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. Freeman, 1979. 7

[16] P. Gawrychowski. Complexity of shortest synchronizing word. Private communiction, 2008. 8, 9

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

[18] 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. 2

[19] K. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10:201–225, 1993. 4

[20] P. Goralčik and V. Koubek. Rank problems for composite transformations. *Internat. J. Algebra Comput.*, 5:309–316, 1995. 7

[21] F. C. Hennie. Fault detecting experiments for sequential circuits. In *Switching Circuit Theory and Logical Design, Proceedings of the Fifth Annual Symposium*, pages 95–110. IEEE Press, 1964. 2

[22] A. E. Laemmel and B. Rudner. Study of the application of coding theory. Technical Report PIBEP-69-034, Dept. Electrophysics, Polytechnic Inst. Brooklyn, Farmingdale, N.Y., 1969. 10

[23] E. F. Moore. Gedanken experiments on sequential machines. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 129–153. Princeton Universty Press, 1956. 2

[24] 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. 4

[25] B. K. Natarajan. Some paradigms for the automated design of parts feeders. *Internat. J. Robotics Research*, 8(6):89–109, 1989. 4

[26] J. Olschewski and M. Ummels. The complexity of finding reset words in finite automata. In P. Hliněný and A. Kučera, editors, *Mathematical Foundations of Computer Science, MFCS'10*, number 6281 in Lecture Notes in Comput. Sci., pages 568–579. Springer-Verlag, 2010. 8, 9

[27] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994. 7

[28] C. H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *J. Comput. System Sci.*, 28(2):244–259, 1984. 8

[29] C. Pixley, S.-W. Jeong, and G. D. Hachtel. Exact calculation of synchronization sequences based on binary decision diagrams. In *Proc. 29th Design Automation Conf.*, pages 620–623. IEEE Press, 1992. 9

[30] 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. 4

[31] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res. Develop.*, 3(2):114–125, 1959. 5

[32] J. L. Ramírez Alfonsín. *The diophantine Frobenius problem.* Oxford University Press, 2005. 10

[33] J.-K. Rho, F. Somenzi, and C. Pixley. Minimum length synchronizing sequences of finite state machine. In *Proc. 30th Design Automation Conf.*, pages 463–468. ACM, 1993. 7

[34] A. Roman. Synchronizing finite automata with short reset words. *Applied Mathematics and Computation*, 209(1):125–136, 2009. 9

[35] I. K. Rystsov. On minimizing length of synchronizing words for finite automata. In *Theory of Designing of Computing Systems*, pages 75–82. Institute of Cybernetics of Ukrainian Acad. Sci., 1980. (in Russian). 7

[36] A. Salomaa. Composition sequences for functions over a finite domain. *Theoret. Comput. Sci.*, 292:263–281, 2003. 7

[37] W. Samotij. A note on the complexity of the problem of finding shortest synchronizing words. In *Proc. AutoMathA 2007, Automata: from Mathematics to Applications.* Univ. Palermo, 2007. (CD). 7, 8

[38] S. Sandberg. Homing and synchronizing sequences. In M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors, *Model-Based Testing of Reactive Systems*, volume 3472 of *Lecture Notes in Comput. Sci.*, pages 5–33. Springer-Verlag, 2005. 2, 6

[39] A. L. Selman. A taxonomy of complexity classes of functions. *J. Comput. System Sci.*, 42(1):357–381, 1994. 9

[40] 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. 7, 9

[41] M. Volkov. Synchronizing automata and the Černý conjecture. In C. Martín-Vide, F. Otto, and H. Fernau, editors, *Language and Automata Theory and Applications*, volume 5196 of *Lecture Notes in Comput. Sci.*, pages 11–27. Springer-Verlag, 2008. 2, 4

[42] H. Wielandt. Unzerlegbare, nicht negative Matrizen. *Math. Z.*, 52:642–648, 1950. (in German). 10

[43] S. Yang. Logic synthesis and optimization benchmarks. Technical Report User Guide Version 3.0, Microelectronics Center of North Carolina, Research Triangle Park, NC, 1991. 9

Index