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`

March 17, 2010 1 h 33

chapterKV

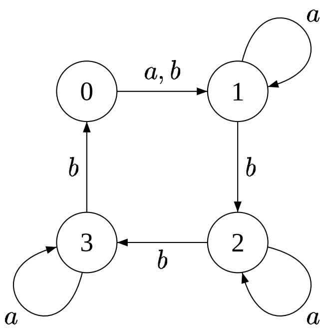
2010 Mathematics Subject Classification: 68Q45

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) $\mathcal{A}$ with input alphabet A and state set Q 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. A synchronizing automaton

Figure 1 shows an example of a synchronizing automaton with 4 states. 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 this automaton. The example in Figure 1 is due to Černý, a Slovak computer scientist, in whose pioneering paper [31] 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 [16].) 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 [1, pp. 60–61].

Needs double-checking!!

In [31] the notion of a synchronizing automaton arose within the classic framework of Moore's "Gedanken-experiments" [17]. 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 [17] 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 [13] 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 [7, 4] as typical samples of technical contributions to the area and

[1]After [12], the name *homing sequence* has become standard for the notion.

[29] for a recent survey.

Another stong motivation comes from the coding theory. We refer to [3, Chapters 3 and 10] for a detailed account of profound connections between codes and automata; here we restrict ourselves to a brief introduction into 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 word x as a factor tends to 1 when the length of v 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 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, $\ldots$ 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, $\ldots$ 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 a finite 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.

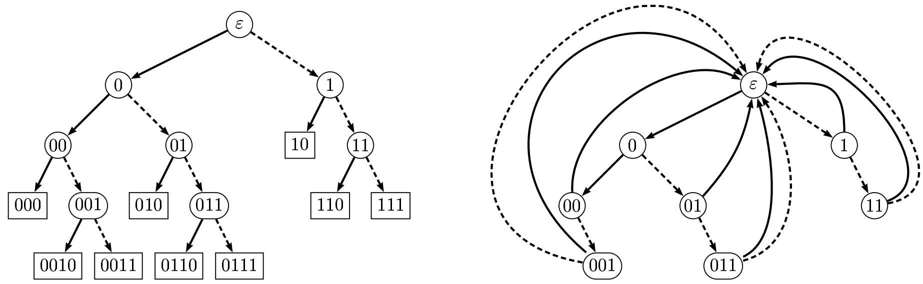


Figure 3. A synchronized code (on the left) and its automaton (on the right)

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 [18, 19] who showed how synchronizing automata can be used to design sensor-free orienters for polygonal parts, see [32, 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 [9, 14, 6].

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 [23]. Given a DFA $\mathcal{A}$ with input alphabet A and state set Q, we define its *subset automaton* $\mathcal{P}(\mathcal{A})$ on the set of the non-empty subsets of 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 4 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

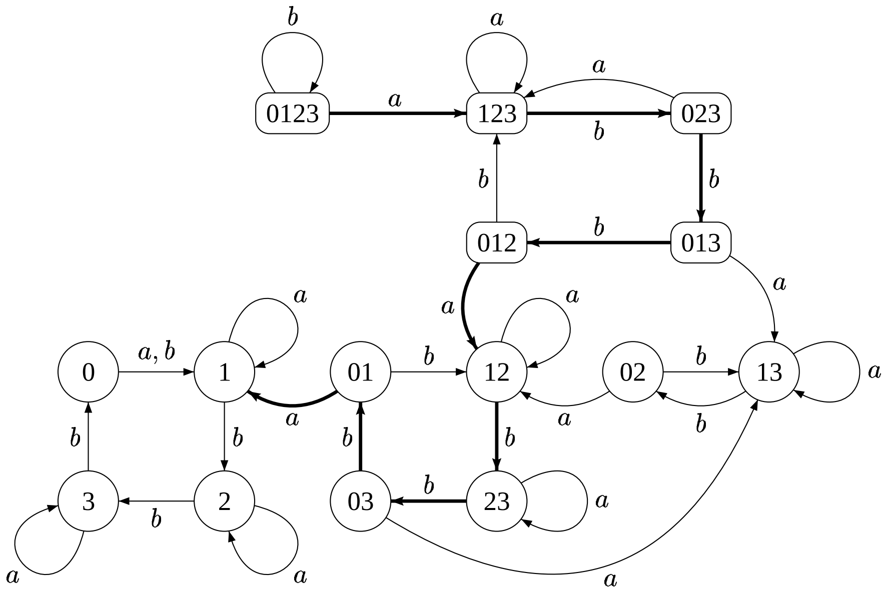


Figure 4. The power automaton $\mathcal{P}(\mathcal{C}_4)$

path in Figure 4 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., [8, 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 [31, Theorem 2] gives rise to a polynomial algorithm.

Proposition 2.1. *A DFA $\mathcal{A}$ with input alphabet A and state set Q 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 [9, Theorem 6], see also [29, 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 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., [24, 30]. 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 [10, 20].

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 [26, 9, 15, 27, 28] 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 [9].

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

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

Figure KV:fig:A2_example 5 shows two automata of the form $\mathcal{A}(\psi)$ build for the SAT instances

$$\begin{aligned}
\psi_1 &= \{x_1 \vee x_2 \vee x_3,\ \neg x_1 \vee x_2,\ \neg x_2 \vee x_3,\ \neg x_2 \vee \neg x_3\}, \\
\psi_2 &= \{x_1 \vee x_2,\ \neg x_1 \vee x_2,\ \neg x_2 \vee x_3,\ \neg x_2 \vee \neg x_3\}.
\end{aligned}$$

If at some state $q \in Q$ in Figure KV:fig:A2_example 5 there is no outgoing arrow labelled $c \in \{a, b\}$, the arrow $q \xrightarrow{c} z$ is assumed (those arrows 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 arrow 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.

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 instance ψ of SAT, one obtains a

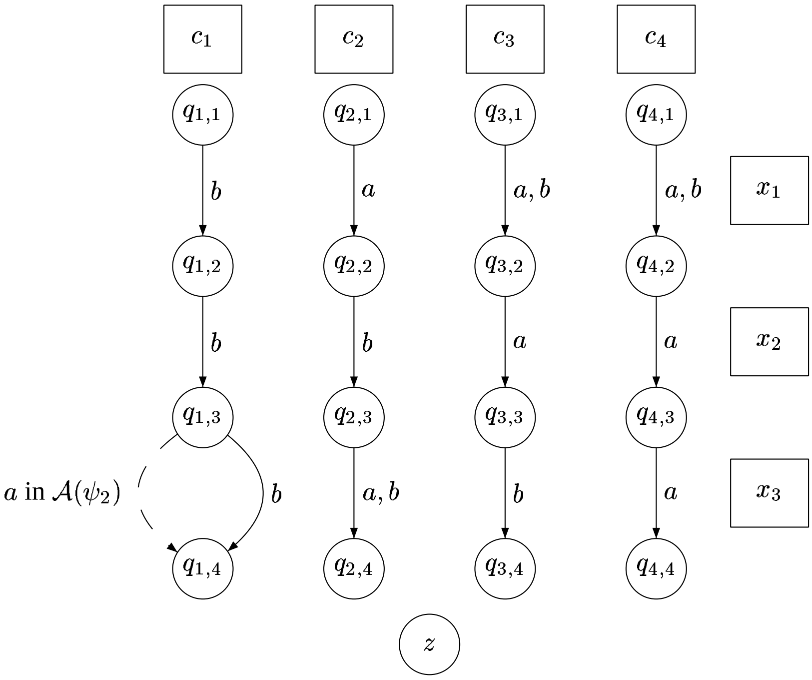


Figure 5. The automata $\mathcal{A}(\psi_1)$ and $\mathcal{A}(\psi_2)$

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.

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

In fact, as observed by Samotij [28], 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?*

Clearly, SHORT-RESET-WORD reduces to SHORTEST-RESET-WORD and by Proposition 2.2 the latter problem is NP-hard. Moreover, assigning the instance $(\mathcal{A}(\psi), n+1)$ of SHORTEST-RESET-WORD to an arbitrary system ψ of clauses, 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 also 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 find the minimum length of a reset word for a given synchronizing automaton in polynomial time.

[2] Actually, the reduction proposed in [28] is not correct but the result claimed in that note can be easily recovered as shown below.

As for exact complexity of the problem SHORTEST-RESET-WORD, it has been recently determined by Gawrychowski [11]. It turns out that the appropriate complexity class is DP (Difference Polynomial-Time introduced by Papadimitriou and Yannakakis [21]; 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. 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 [11].

Recently Berlinkov [2] has shown (assuming P $\neq$ NP) that no polynomial algorithm can approximate the minimum length of reset words for a given synchronizing automaton within a constant factor. We mention that Pixley, Jeong and Hachtel [22] suggested an heuristic algorithm for finding short reset words in synchronizing automata that was reported to perform rather satisfactory on a number of benchmarks from [33]; further algorithms yielding short (though not necessarily shortest) reset words have implemented by Trahtman [30] and Roman [25].

3 The Černý conjecture

4 The road coloring problem

5 Generalizations

References

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

[2] 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*, number 6072 in Lecture Notes in Comput. Sci. Springer-Verlag, 2010. 8

[3] J. Berstel, D. Perrin, and C. Reutenauer. *Codes and automata.* Number 129 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2009. 3

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

[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. 3

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

[7] 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

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms.* MIT Press and McGraw-Hill, 2001. 5

[9] D. Eppstein. Reset sequences for monotonic automata. *SIAM J. Comput.*, 19:500–510, 1990. 4, 5, 6

[10] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness.* Freeman, 1979. 6

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

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

[13] 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

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

[15] P. Goralčik and V. Koubek. Rank problems for composite transformations. *Internat. J. Algebra Comput.*, 5:309–316, 1995. 6

[16] 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

[17] 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

[18] 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

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

[20] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994. 6

[21] 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

[22] 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. 8

[23] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res. Develop.*, 3(2):114–125, 1959. 4

[24] 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. 6

[25] A. Roman. Synchronizing finite automata with short reset words. *Applied Mathematics and Computation*, 209(1):125–136, 2009. 8

[26] 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). 6

[27] A. Salomaa. Composition sequences for functions over a finite domain. *Theoret. Comput. Sci.*, 292:263–281, 2003. 6

[28] 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). 6, 7

[29] 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. 3, 5

[30] 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.*, number 4162 in Lecture Notes in Comput. Sci., pages 789–800. Springer-Verlag, 2006. 6, 8

[31] J. Černý. Poznámka k homogénnym eksperimentom s konečnými automatami. *Matematicko-fyzikalny Časopis Slovenskej Akadémie Vied*, 14(3):208–216, 1964. (in Slovak). 2, 5

[32] 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. 4

[33] S. Yang. Logic synthesis and optimization benchmarks. Technical Report User Guide Version 3.0, Microelectronics Center of North Carolina, Research Triangle Park, NC, 1991. 8

Index