Complexity of Problems Concerning Carefully Synchronizing Words for PFA and Directing Words for NFA

P.V. Martyugin

Ural State University,
620083 Ekaterinburg, Russia
martugin@mail.ru

Abstract. We show that the problem of checking careful synchronizability of partial finite automata and the problem of finding the shortest carefully synchronizing word are PSPACE-complete. We show that the problem of checking D_1, D_2 and D_3-directability of nondeterministic finite automata and the problem of finding the shortest D_1, D_2 and D_3-directing word are PSPACE-complete. The restrictions of these problems to 2-letter automata remain PSPACE-complete.

Keywords: Synchronization, Automata, Directing Words, Computational Complexity.

1 Introduction

A *deterministic finite automaton (DFA)* is a triple $\mathscr{A} = (Q, \Sigma, \delta)$, where Q is a finite set of states, Σ is a finite alphabet and δ is a totally defined transition function. Denote by 2^Q the set of all subsets of the set Q, and by Σ^* the free Σ-generated monoid with the empty word λ. The function δ extends in a natural way to the action $Q \times \Sigma^* \to Q$. This extension is also denoted by δ. A DFA $\mathscr{A} = (Q, \Sigma, \delta)$ is called *synchronizing* if there exists a word $w \in \Sigma^*$ whose action resets $\mathscr{A}$, that is, leaves the automaton in one particular state no matter at which state in Q it started: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$. Any word w with this property is said to be a *reset* or *synchronizing* word for the automaton $\mathscr{A}$.

A conjecture proposed by Černý in [1] states that every synchronizing automaton with n states can be synchronized by a word of length at most $(n-1)^2$. There have been many attempts to prove it, but they all have failed so far. The conjecture has been proved only for some special cases of automata. The best known upper bound is $(n^3 - n)/6$ (see [2]). The corresponding lower bound has been proved by Černý [1]. Surveys of results concerning synchronizing words can be found in [4].

It is natural to consider the computational complexity of various problems arising from the study of automata synchronization. The main natural questions are as follows: given an automaton is it synchronizing or not, and what is the

F. Ablayev and E.W. Mayr (Eds.): CSR 2010, LNCS 6072, pp. 288–302, 2010.

length of the shortest synchronizing word for a given automaton? In [3], Eppstein presented an algorithm which checks whether a given DFA $\mathscr{A} = (Q, \Sigma, \delta)$ is synchronizing. This algorithm runs in $O(|\Sigma| \cdot |Q|^2) + |Q|^3$ time. Moreover, for a synchronizing automaton this algorithm finds some synchronizing word although not necessarily the shortest. In [3], it is proved that the following problem is NP-complete. Given a DFA $\mathscr{A}$ and an integer L, is there a word of length $\leq L$ resetting the automaton $\mathscr{A}$? This problem remains NP-complete even if the input automaton has a 2-letter alphabet.

The notion of a synchronizing word can be generalized to the case of DFA with a partial transition function (PFA) and to nondeterministic finite automata (NFA). *A partial finite automaton (PFA)* is a triple $\mathscr{A} = (Q, \Sigma, \delta)$, where Q is a finite set of states, Σ is a finite alphabet and δ is a partial function from $Q \times \Sigma$ to Q (the function δ can be undefined on some pairs from the set $Q \times \Sigma$). The function δ can be naturally extended to $2^Q \times \Sigma^*$ as follows. We put $\delta(q, \lambda) = q$ for every $q \in Q$. Let $q \in Q$, $a \in \Sigma$, $w \in \Sigma^*$. If both states $p = \delta(q, w)$ and $\delta(p, a)$ are defined, then we put $\delta(q, wa) = \delta(p, a)$. If $S \subseteq Q$, $w \in \Sigma^*$ and the values $\delta(q, w)$ are defined for all states $q \in S$, then we put $\delta(S, w) = \{\delta(q, w) | q \in S\}$.

A PFA $\mathscr{A} = (Q, \Sigma, \delta)$is called *carefully synchronizing*, if there is a word $w \in \Sigma^*$ such that the value $\delta(Q, w)$ is defined and $|\delta(Q, w)| = 1$. We say that such a word w is a *carefully synchronizing word (c.s.w.)* for the automaton $\mathscr{A}$. Clearly DFA is a partial case of PFA and in this case any c.s.w. is also synchronizing. Therefore careful synchronization of PFA is a natural generalization of the synchronization of DFA. The notion of careful synchronization was introduced in [6]. This paper is devoted to a generalization of the Černý problem to the case of PFA. Let $c(n)$ be the maximal length of the shortest c.s.w. for carefully synchronizing PFA $\mathscr{A}$ with n states. It follows from [6] and [7] that

$$\Omega(3^{n/3}) \leq c(n) \leq O(n^2 \cdot 4^{n/3}).$$

A *nondeterministic finite automaton (NFA)* is a triple $\mathscr{A} = (Q, \Sigma, \delta)$ such that Q is a finite set of states, Σ is a finite alphabet, and δ is a function from $Q \times \Sigma$ to 2^Q. The function δ can be naturally extended to the set $2^Q \times \Sigma^*$. Let $S \subseteq Q$, $a \in \Sigma$, then we put $\delta(S, a) = \bigcup_{q \in S} \delta(q, a)$. We also put $\delta(S, \lambda) = S$. Let $S \subseteq Q$, $w \in \Sigma^*$, $w = ua$ and the set $\delta(S, u)$ is defined, then we put $\delta(S, w) = \delta(\delta(S, u), a)$. Note, that PFA is a partial case of NFA, but the definitions of the functions $\delta : 2^Q \times \Sigma^* \to 2^Q$ for PFA and NFA are sufficiently different, because these function have different co-domain.

Let $\mathscr{A} = (Q, \Sigma, \delta)$ be an NFA and $w \in \Sigma^*$. The word w is D_1-*directing* if $\delta(q, w) \neq \emptyset$ for all $q \in Q$ and $|\delta(Q, w)| = 1$. The word w is D_2-*directing* if $\delta(q, w) = \delta(Q, w)$ for all $q \in Q$. The word w is D_3-*directing* if $\bigcap_{q \in Q} \delta(q, w) \neq \emptyset$. The NFA $\mathscr{A}$ is called D_1, D_2 or D_3-*directing* if there is a D_1, D_2 or D_3-directing word for it. The D_1 and D_3-directability is a generalization of careful synchronization in the case when an NFA is a PFA and the D_1, D_2 and D_3-directability is a generalization of the synchronization in the case that an NFA is a DFA. The D_1, D_2 and D_3-directing words were studied in [5,7]. Let $d_1(n)$, $d_2(n)$ and $d_3(n)$

be the maximal lengths of the shortest D_1, D_2 or D_3-directing words for NFA with n states respectively. The following lower and upper bounds of the length of the shortest directing words were obtained:

$$2^n - n - 1 \leq d_1(n), d_2(n) \leq O(2^n), \quad \Omega(3^{n/3}) \leq d_3(n) \leq O(n^2 \cdot 4^{n/3}).$$

There are two natural questions. How can we check whether a given PFA (NFA) is carefully synchronizing (D_1, D_2, D_3-directable)? How can we find the shortest carefully synchronizing (D_1, D_2, D_3-directing) word for a given PFA (NFA)?

These problems can be solved using simple algorithms. Let us consider an algorithm of finding a c.s.w. for a PFA (the problems of finding D_1, D_2, D_3-words can be solved using similar algorithms). The algorithm uses the *power automaton* $2^{\mathscr{A}} = (2^Q, \Sigma, \delta)$, where 2^Q is a set of all subsets of Q, and $\delta : 2^Q \times \Sigma \to 2^Q$ is an extension of the function δ. If there is a path from the set Q to some one-element set in the graph of the automaton $2^{\mathscr{A}}$, then there is a c.s.w. for $\mathscr{A}$. This word can be read along the path in the graph. The path in the graph of the automaton $2^{\mathscr{A}}$ can be found using breadth-first search. The breadth-first search finds the shortest c.s.w. Such algorithm uses an exponential time and an exponential memory in the size of PFA $\mathscr{A}$.

The natural questions are as follow: can we solve the problems described before using the memory of polynomial size? What is the complexity of checking careful syncronizability and D_1, D_2, D_3-directability? What is the complexity of finding carefully synchronizing (D_1, D_2, D_3-directing) words? In this paper we prove that all these problems are PSPACE-complete. These problems stated for automata over a k-letter alphabet for $k > 1$ (for instance for 2-letter alphabet) are PSPACE-complete too.

Let us consider one more natural subclass of PFA. A PFA $\mathscr{A} = (Q, \Sigma, \delta)$ is called a *PFA with a zero* if there is a state $z \in Q$ such that for any letter $a \in \Sigma$ it follows $\delta(z, a) = z$. We will show that checking careful syncronizability for PFA with a zero is PSPACE-complete even for 2-letter PFA.

2 Problems

Let us give formal definitions of the problems we have discussed.

Problem: CARSYN (ZERO CARSYN)
Input: A PFA (a PFA with a zero) $\mathscr{A} = (Q, \Sigma, \delta)$.
Question: Is the automaton $\mathscr{A}$ carefully synchronizing?

Problem: D1DIR (D2DIR, D3DIR)
Input: A NFA $\mathscr{A} = (Q, \Sigma, \delta)$.
Question: Is the automaton $\mathscr{A}$ D_1-directing (D_2-directing, D_3-directing)?

Problem: SHORT CARSYN (ZERO CARSYN) WORD
Input: A PFA (a PFA with a zero) $\mathscr{A} = (Q, \Sigma, \delta)$.
Task: To check whether the automaton $\mathscr{A}$ is carefully syncronizing and to find the shortest c.s.w. for the automaton $\mathscr{A}$ if $\mathscr{A}$ is carefully synchronizing.

Problem: SHORT D1DIR (D2DIR, D3DIR) WORD
Input: A NFA $\mathscr{A} = (Q, \Sigma, \delta)$.
Task: To check whether the automaton $\mathscr{A}$ is D_1-directing (D_2-directing, D_3-directing) and to find the shortest D_1-directing (D_2-directing, D_3-directing) word for automaton $\mathscr{A}$ if $\mathscr{A}$ is D_1-directing (D_2-directing, D_3-directing).

If the input of some PROBLEM contains only automata over an alphabet of size $\leq k$ for some fixed k, then we call such a problem k-PROBLEM (for example, k-CARSYN or k-SHORT D1DIR WORD).

We will prove that all the problems defined above are PSPACE-complete. We use polynomial-space reducibility of problems. Let the problem A be polynomial-space reducible to the problem A. We denote this $A \leq_p B$. This means that for any instance α of the problem A there is an instance β of the problem B such that the answer on α is true iff the answer on β is true and the size of β is polynomial in the size of α. The problem B does not have to be a decision problem, but can also be a search problem (like SHORT CARSYN WORD). To find a solution of such a problem includes the problem of determining the existence of the solution. Therefore, the relation $A \leq_p B$ is defined for all problems B defined in this section. The length of fining words can be not polynomial in the size of input automata. Therefore, then we calculate the computational space complexity of some search problem we do not take take in account the size of output. The size of search problem is the size of used memory without output tape. The next proposition contain some trivial relations among the problems.

Proposition 1. *Let PROBLEM* $\in$ {*CARSYN, ZERO CARSYN, D1DIR, D2DIR, D3DIR*}*, and let* $k \geq 2$ *be an integer, then*

1. *ZERO CARSYN* $\leq_p$ *CARSYN, D2DIR*
2. *k-ZERO CARSYN* $\leq_p$ *k-CARSYN, k-D2DIR*
3. *SHORT ZERO CARSYN WORD* $\leq_p$ *SHORT CARSYN (D2DIR) WORD;*
4. *k-SHORT ZERO CARSYN WORD* $\leq_p$ *k-SHORT CARSYN (D2DIR) WORD.*
5. *CARSYN* $\leq_p$ *D1DIR, D3DIR;*
6. *k-CARSYN* $\leq_p$ *k-D1DIR, k-D3DIR;*
7. *SHORT CARSYN WORD* $\leq_p$ *SHORT D1DIR (D3DIR) WORD;*
8. *k-SHORT CARSYN WORD* $\leq_p$ *k-SHORT D1DIR (D3DIR) WORD.*
9. *PROBLEM* $\leq_p$ *SHORT PROBLEM WORD;*
10. *k-PROBLEM* $\leq_p$ *k-SHORT PROBLEM WORD;*
11. *k-PROBLEM* $\leq_p$ $k+1$*-PROBLEM;*
12. *k-SHORT PROBLEM WORD* $\leq_p$ $k+1$*-SHORT PROBLEM WORD;*
13. *k-PROBLEM* $\leq_p$ *PROBLEM;*
14. *k-SHORT PROBLEM WORD* $\leq_p$ *SHORT PROBLEM WORD;*

Proof. See Appendix.

Now, let us formulate the main theorem.

Theorem 1. *For any integer* $k \geq 2$

1. *The problems CARSYN, SHORT CARSYN WORD, k-CARSYN and k-SHORT CARSYN WORD are PSPACE-complete;*
2. *The problems ZERO CARSYN, ZERO SHORT CARSYN WORD, k- ZERO CARSYN and k-SHORT ZERO CARSYN WORD are PSPACE-complete;*
3. *The problems D1DIR, SHORT D1DIR WORD, k-D1DIR and k-SHORT D1DIR WORD are PSPACE-complete;*
4. *The problems D2DIR, SHORT D2DIR WORD, k-D2DIR and k-SHORT D2DIR WORD are PSPACE-complete;*
5. *The problems D3DIR, SHORT D3DIR WORD, k-D3DIR and k-SHORT D3DIR WORD are PSPACE-complete.*

Proof. Our proof will consist of three steps. Firstly we reduce the problem FINITE AUTOMATA INTERSECTION to the problem ZERO CARSYN and prove that the problem CARSYN is PSPACE-hard (Proposition 2). Secondly we reduce the problem ZERO CARSYN to the problem 2-ZERO CARSYN with the help of automata obtained in Proposition 3. This proves that the problem 2-ZERO CARSYN is PSPACE-hard (Proposition 3). Thirdly we show that the problems SHORT D1, D2 and D3DIR WORD belong to the class PSPACE (Proposition 4). Then Proposition 1 gives us PSPACE-completeness of all considered problems.

Now we give some auxiliary definitions and notations. Let $w \in \Sigma^*$. Denote by We let $|w|$ denote the length of the word w. Let $i, j \in \{1, \dots, |w|\}$. We let $w[i]$ denote the i-th letter of the word w. We let $w[i, j]$ denote the word $w[i]w[i+1]\cdots w[j]$. Let n be a natural number. We denote by Σ^n the set of all words of length n over the alphabet Σ.

3 The Problems Are PSPACE-Hard

Let us prove that the problem ZERO CARSYN is PSPACE-hard. We use the classical PSPACE-complete problem FINITE AUTOMATA INTERSECTION (see [8]). We consider deterministic finite automata of the form $\mathscr{A} = (Q, \Sigma, \delta, s, F)$ as *recognizers*, where Q is a set of states, Σ is an alphabet, δ is a totally-defined transition function, $s \in Q$ is an initial state and $F \subseteq Q$ is a set of final states. Let w be a word Σ^*. The automaton $\mathscr{A}$ *accepts* the word w if and only if $\delta(s, w) \in F$.

Problem: FINITE AUTOMATA INTERSECTION
Input: The recognizers $\mathscr{A}_1 = (Q_1, \Sigma, \delta_1, s_1, F_1), \dots, \mathscr{A}_k = (Q_k, \Sigma, \delta_k, s_k, F_k)$, for $k \geq 2$.
Question: Is there a word $w \in \Sigma^*$ such that
$\delta_1(s_1, w) \in F_1, \dots, \delta_k(s_k, w) \in F_k$?

Proposition 2. *The problem ZERO CARSYN is PSPACE-hard.*

Proof. We reduce the problem FINITE AUTOMATA INTERSECTION to the problem ZERO CARSYN. Let recognizers $\mathscr{A}_1 = (Q_1, \Sigma', \delta_1, s_1, F_1), \dots$
$\dots, \mathscr{A}_k = (Q_k, \Sigma', \delta_k, s_k, F_k)$, $k \geq 2$ be an input of the problem FINITE AUTOMATA INTERSECTION. We construct a PFA $\mathscr{B} = (Q, \Sigma, \delta)$ such that there

is a c.s.w. for the automaton $\mathscr{B}$ if an only if the recognizers $\mathscr{A}_1, \ldots, \mathscr{A}_k$ have a common accepting word.

Let $\mathscr{B} = (Q, \Sigma, \delta)$ be a PFA with $Q = Q_1 \cup \ldots \cup Q_k \cup \{beg, end\}$, $\Sigma = \Sigma' \cup \{x, y\}$ and the following transition function. Let $c \in \Sigma'$ and $q \in Q_i$ for some $i \in \{1, \ldots, k\}$. We put

$$\delta(q, c) = \delta_i(q, c),\ \delta(q, x) = s_i,\ \delta(q, y) = \begin{cases} end, & q \in F_i \\ \text{undefined}, & \text{otherwise} \end{cases}$$

$$\delta(end, x) = \delta(end, y) = \delta(end, c) = end.$$

$$\delta(beg, x) = s_1, \delta(beg, y), \delta(beg, c) \text{ are undefined.}$$

The automaton $\mathscr{B}$ is represented by Figure 2 which can be found in Appendix. The state *end* is a zero in the PFA $\mathscr{B}$.

Let $w \in \Sigma'^*$ be a a word accepted by all the recognizers $\mathscr{A}_1, \ldots, \mathscr{A}_k$. Let us prove that the word xwy is a c.s.w. for the PFA $\mathscr{B}$. Indeed, $\delta(Q, x)$ is defined and $\delta(Q, x) = \{s_1, \ldots, s_k, end\}$. We have $\delta_1(s_1, w) \in F_1, \ldots, \delta_k(s_k, w) \in F_k$. Therefore the function $\delta(\bullet, w)$ is defined on the set $\{s_1, \ldots, s_k\}$ and $\delta(\{s_1, \ldots, s_k\}, w) \subseteq F_1 \cup \cdots \cup F_k$. Hence the letter y is defined on the set $\delta(Q, xw) = \delta(\{s_1, \ldots, s_k\} \cup \{end\}, w)$ and $\delta(Q, xwy) = \{end\}$. Thus the word xwy is a c.s.w. for the PFA $\mathscr{B}$.

Lemma 1. *Each shortest c.s.w. $u \in \Sigma^*$ for the automaton $\mathscr{B}$ is of the form $u = xwy$ for some $w \in \Sigma'^*$.*

Proof. See Appendix.

Let $u \in \Sigma^*$ be some shortest c.s.w. for the automaton $\mathscr{B}$. By Lemma 1 we get $u = xwy$ for some $w \in \Sigma'^*$. The letter y is defined on the set $\delta(Q, xw)$. Therefore $\delta(Q, xw) \subseteq F_1 \cup \cdots \cup F_k \cup \{end\}$. This means that $\delta_1(s_1, w) \in F_1, \ldots, \delta_k(s_k, w) \in F_k$ and the word w is a common accepting word for the recognizers $\mathscr{A}_1, \ldots, \mathscr{A}_k$.

We have reduced the problem FINITE AUTOMATA INTERSECTION to the problem ZERO CARSYN. The size of the PFA $\mathscr{B}$ is polynomial in the sum of the sizes of automata $\mathscr{A}_1, \ldots, \mathscr{A}_k$. Thus the problem ZERO CARSYN is PSPACE-hard. The proposition is proved.

A PFA $\mathscr{B}$ is called *simple* if it can be constructed from some recognizers using the procedure from the proof of Proposition 2. Note that the problem ZERO CARSYN is PSPACE-hard not only for the class of arbitrary PFA with a zero but for the class of simple PFA. We denote such problem by SIMPLE CARSYN. It will be used in the proof of the next proposition. Proposition 3 states the PSPACE-hardness for the problem ZERO CARSYN for two-letter automata.

Proposition 3. *The problem 2-ZERO CARSYN is PSPACE-hard.*

Proof. We reduce the problem SIMPLE CARSYN to the problem 2-ZERO CARSYN. Let a PFA $\mathscr{B}$ be an input of the problem SIMPLE CARSYN. This means that $\mathscr{B} = (Q, \Sigma, \delta)$, $Q = \{q_1, \ldots, q_n\}$, $\Sigma = \{c_1, \ldots, c_m, x, y\}$ and there

is a state $end \in Q$ such that every shortest c.s.w. u for PFA $\mathscr{B}$ is equal to xwy for some $w \in \{c_1, \ldots, c_m\}^*$, $\delta(Q, xwy) = end$ and the action of letters $y, c_1, \ldots, c_m$ is undefined on the set Q. We also suppose $end = q_n$. We construct a PFA $\mathscr{C} = (P, \{a, b\}, \gamma)$ with a zero state z such that there is a c.s.w. for the automaton $\mathscr{C}$ if an only if there is a c.s.w. for the automaton $\mathscr{B}$.

Let $\mathscr{C} = (P, \{a, b\}, \gamma)$ be a PFA, $P = \{p_{i,k} | i \in \{1, ..., n-1\}, k \in \{0, ..., m+1\}\} \cup \{z\}$. Denote $c_0 = y$, $c_{m+1} = x$. We also denote $p_{n,1} = \ldots = p_{n,m+1} = z$ Let $i \in \{1, ..., n\}$, $k \in \{0, ..., m+1\}$. Define the function γ.

$$\text{for } j \le m, \gamma(p_{i,k}, a) = p_{i,k+1},\ \gamma(p_{i,m+1}, a) = p_{i,m+1},$$

$$\gamma(p_{i,k}, b) = \begin{cases} p_{t,0}, & \text{if } \delta(q_i, c_k) = q_t \\ \text{undefined}, & \text{if } \delta(q_i, c_k) \text{ is undefined.} \end{cases}$$

The state $z = p_{n,1} = \ldots = p_{n,m+1}$ is a zero state in the automaton $\mathscr{C}$.

Figure 1 represents an example of reduction according to the proofs of Propositions 2 and 3. At first, the set of recognizers $\{\mathscr{A}_1, \mathscr{A}_2\}$ reduces to the corresponding

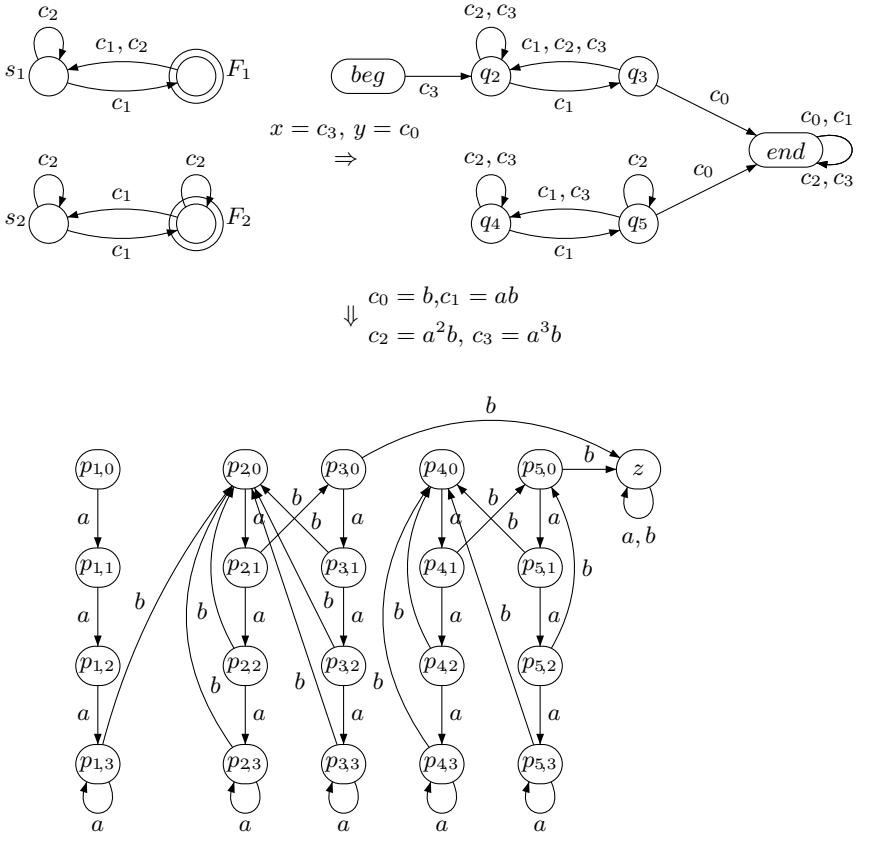

Fig. 1. The example of automata $\mathscr{A}_1, \mathscr{A}_2$, $\mathscr{B}$ and $\mathscr{C}$

PFA $\mathscr{B}$ using the proof of Proposition 2. Then the PFA $\mathscr{B}$ reduces to the corresponding PFA $\mathscr{C}$.

Let $i \in \{1, \ldots, n\}$. We call the set $Q_i = \{p_{i,0}, \ldots, p_{i,m+1}\}$ the *i-th column* of the set P. Let $k \in \{0, \ldots, m+1\}$. We call the set $R_k = \{p_{1,k}, \ldots, p_{n,k}\}$ the *k-th row* of the set P. Note, that n-th column is one state z, and on the over hand, the state z belongs to any row. The following statements are evident corollaries of the definition of the function γ.

Lemma 2. *1.* $\gamma(Q_i, a) \subseteq Q_i$ *for* $i \in \{1, \ldots, n\}$;
2. $\gamma(R_k, a) = R_{k+1}$ *for* $k \in \{0, \ldots, m\}$, $\gamma(R_{m+1}, a) = R_{m+1}$;
3. $\gamma(P, b) \subseteq R_0$.

Let us formulate more complicated properties about the function γ.

Lemma 3. *Let* $v \in \{a, b\}^* \backslash \{a\}^*$ *and the value* $\gamma(P, v)$ *is defined then*

1. for any $i \in \{1, \ldots, n\}$, *we have* $|Q_i \cap \gamma(P, v)| \leq 1$;
2. $|\gamma(P, va)| = |\gamma(P, v)|$;
3. $\gamma(P, vb) \subseteq R_0$.

Proof. See Appendix

Let us consider a morphism $\phi : \{a, b\}^* b \to \Sigma^*$, where $\Sigma = \{c_0, \ldots, c_{m+1}\}$ and the set $\{a, b\}^* b$ is the set of all words over $\{a, b\}$ ended by b. By definition put $\phi(a^k b) = c_k$ for $k \in \{0, \ldots, m+1\}$ and $\phi(a^k b) = c_{m+1}$ for $k \geq m+2$. Every word from the set $\{a, b\}^* b$ is a product of words from the set $\{a^k b | k = 0, 1, 2, ...\}$. Therefore ϕ can be extended to the set $\{a, b\}^* b$. We also consider $\phi^{-1} : \Sigma^* \to \{a, b\}^* b$. Put $\phi^{-1}(c_k) = a^k b$. Of course, the morphism ϕ^{-1} is not a inverse relation to the function ϕ, but ϕ^{-1} is a convenient notation. The morphism ϕ^{-1} also can be extended to Σ^*. Note that for any word $u \in \Sigma^*$ we have $\phi(\phi^{-1}(u)) = u$. We are going to prove that the word u is a c.s.w. for PFA $\mathscr{B}$ if and only if the word $\phi^{-1}(u)$ is a c.s.w. for PFA $\mathscr{C}$.

Let $v \in \{a, b\}^* b$ and the value $\gamma(P, v)$ is defined. By Lemma 2 we get $\gamma(P, b) \subseteq R_0$. Therefore $\gamma(P, v) \subseteq R_0$. We define the value $I(v) = \{i | p_{i,0} \in \gamma(P, v)\}$. Let $u \in \Sigma^*$ and the value $\delta(Q, u)$ is defined. Denote $J(u) = \{i | q_i \in \delta(Q, u)\}$.

Lemma 4. *If* $v \in \{a, b\}^* b$ *and the value* $\gamma(P, v)$ *is defined, then* $I(v) = J(\phi(v))$. *If* $u \in \Sigma^*$ *and the value* $\delta(Q, u)$ *is defined, then* $J(u) = I(\phi^{-1}(u))$.

Proof. Let $v \in \{a, b\}^* b$, then $v = a^{h_1} b a^{h_1} b \ldots a^{h_r} b$ for some integer $h_1, \ldots, h_r$. Let $v_k = a^{h_1} b \ldots a^{h_k} b$ for some $k \in \{1, \ldots, r\}$. The value $\gamma(P, v)$ is defined. This means that the value $\gamma(P, a^{h_1} b) = \gamma(P, v_1)$ is defined too. It is possible only in the case of $h_1 \geq m+1$, because if $h_1 < m+1$ then $p_{m,n} \in \gamma(P, a^{h_1})$ and the letter b is undefined on the set $\gamma(P, a^{h_1})$. Therefore $\phi(v_1) = \phi(a^{h_1} b) = c_{m+1} = x$. It can be easily verified that $I(v_1) = I(a^{h_1} b) = J(c_{m+1})$.

Let $k > 1$. Let $I(v_{k-1}) = J(\phi(v_{k-1}))$. We prove that $I(v_k) = J(\phi(v_k))$. Let $i \in I(v_{k-1}) = J(\phi(v_{k-1}))$. This means that $p_{i,0} \in \gamma(P, v_{k-1})$ and $q_i \in \delta(Q, \phi(v_{k-1}))$. The value $\gamma(p_{i,0}, a^k b)$ is defined. This means that $\gamma(p_{i,0}, a^k b) = p_{m,0}$ for some

$m \in I(v_k)$. Therefore, from the definition of the function γ we obtain that the value $\delta(q_i, \phi(a^{h_k}b))$ is defined and $\delta(q_i, \phi(a^{h_k}b)) = q_m$. Hence $m \in J(\phi(v_k))$ and $I(v_k) \subseteq J(\phi(v_k))$. The proof of the inclusion $I(v_k) \supseteq J(\phi(v_k))$ is analogous. Therefore $I(v_k) = J(\phi(v_k))$. This means that $I(v) = J(\phi(v))$.

The second statement of the lemma can be proved similarly using an induction and the definition of the function γ. The lemma is proved.

Let u be some shortest c.s.w. for PFA $\mathscr{B}$. By Lemma 1 we get $u = xwy$ for some $w \in \Sigma'^*$. Therefore $u = c_{m+1}wc_0 = \phi(a^{m+1}b)w\phi(b)$. Hence for any $k \in \{1, \ldots, |u|\}$ the last letter of the word $\phi^{-1}(u[1,k])$ is b and $\phi^{-1}(u[1,k]) \in \{a,b\}^*b$. By Lemma 4 we obtain $J(u[1,k]) = I(\phi^{-1}(u[1,k]))$. In particular $J(u) = I(\phi^{-1}(u))$. Hence $|I(\phi^{-1}(u))| = |J(u)| = 1$. Therefore the word $\phi^{-1}(u)$ is a c.s.w. for the automaton C.

Let v be some shortest c.s.w. for PFA $\mathscr{C}$. It is easy to see that any word form the set a^* is not carefully synchronizing. Therefore $v \in \{a,b\}^*\backslash\{a\}^*$. By Lemma 3 we obtain $|\gamma(P, v[1,|v|-1]a)| = |\gamma(P, v[1,|v|-1])|$. Therefore $v[|v|] = b$ and $v \in \{a,b\}^*b$. Whence $I(v) = J(\phi(v))$. The word v is a c.s.w. for the automaton $\mathscr{C}$ and hence $|I(v)| = 1$. Therefore $|J(\phi(v))| = 1$. This means that the word $\phi(v)$ is a c.s.w. for the automaton $\mathscr{B}$. The proposition is proved.

4 The Problems Are in PSPACE

Proposition 4. *The problems SHORT D1, D2 and D3DIR WORD belong to the complexity class PSPACE.*

Proof. At first, recall the following property.

Lemma 5. *There is a constant C such that for $D \in \{D_1, D_2, D_3\}$ and for every D-directing NFA with n states the length of the shortest D-directing word is less than $C \cdot 2^n$.*

Proof. The statement follows from [7].

Using Lemma 5 and the Savitch's theorem (which states that PSPACE= NPSPACE) it can be proved that the problems CARSYN, D1DIR, D2DIR and D3DIR are belong to PSPACE, because any carefully synchronizing or directing word can be nondeterministically applied to a given PFA or NFA using $O(n)$ bits of memory. Similar but more complicated ideas can be used to prove that the corresponding search problems are also belong to PSPACE. We do not discuss here such approach because there can be some conceptual problems with a using of Savitch's theorem for a search problem and for a model with out taking in account of the size of output.

Instead of using nondeterministic calculations we introduce here an deterministic searching algorithm. Let $\mathscr{A} = (Q, \Sigma, \delta)$ be an NFA with $Q = \{q_1, \ldots, q_n\}$ and $\Sigma = \{a_1, \ldots, a_k\}$. The automaton $\mathscr{A}$ can be stored using $O(nk)$ bits of memory. We are going to describe an algorithm for searching the shortest D_1, D_2 or D_3-directing words for the automaton $\mathscr{A}$. Our purpose is to construct

an algorithm that uses space that is a polynomial in $O(nk)$. We do not take in account the size of output word. Our algorithm examines different word and finds the first appropriate one. A simple corollary of Lemma 5 is that the length of the shortest directing word can be kept using $O(n)$ bits of memory. The same fact is true for all numbers used in our algorithm.

Our algorithm is a procedure $FindWord(D, \mathscr{A})$ which takes some $D \in \{D_1, D_2, D_3\}$ and some NFA $\mathscr{A}$ as input parameters and returns one of the shortest D_1, D_2 or D_3-directing word letter by letter. The procedure *FindWord* uses the function $GetLetter(L, N)$ where N and L are integers and $N \leq L$. The parameters D and $\mathscr{A}$ are global variables for the function *GetLetter*. This function returns either the N-th letter of a D-directing word of length L for the NFA $\mathscr{A}$ or the value *no* if there is no D-directing word of length L for $\mathscr{A}$. The function calls $GetLetter(L, N_1)$ and $GetLetter(L, N_2)$ return different letters of one word. Let us describe the procedure $FindWord(D, \mathscr{A})$. If the NFA $\mathscr{A}$ is D-directing then the procedure finds the length of the shortest D-directing word and writes this word letter by letter.

```
Procedure FindWord(D, A)
    L = 1
    While L ≤ C · 2^n and GetLetter(L, 1) = no Do
        L = L + 1
    If (L > C · 2^n) Then
        Return "There is no D-directing word"
    Else
        For N = 1 To L Do
            Write GetLetter(L, N)
End Procedure
```

Now we describe the function $GetLetter(L, N)$. Let $\mathscr{A} = (Q, \Sigma, \delta)$ and $Q = \{q_1, \ldots, q_n\}$. A *position* of the NFA $\mathscr{A}$ is an array $P = (P^1, \ldots, P^n)$ where $P^1, \ldots, P^n \subseteq Q$. Let $Pos(\mathscr{A})$ be the set of all positions of the NFA $\mathscr{A}$. The function δ extends to the function acting from $Pos(\mathscr{A}) \times \Sigma$ to $P(\mathscr{A})$. Let $P \in Pos(\mathscr{A})$ and $a \in \Sigma$. We put $\delta(P, a) = (\delta(P^1, a), \ldots, \delta(P^n, a))$. The *starting position* of the NFA $\mathscr{A}$ is the array $S_0 = (\{q_1\}, \ldots, \{q_n\})$. Let $w \in \Sigma^*$. The array $P = (P^1, \ldots, P^n)$ is a D_1-*position* of $\mathscr{A}$ if $P^1 = \cdots = P^n = \{q\}$ for some $q \in Q$. The array P is a D_2-*position* of $\mathscr{A}$ if $P^1 = \cdots = P^n$. The array P is a D_3-*position* of $\mathscr{A}$ if $P^1 \cap \cdots \cap P^n \neq \emptyset$. The position $\delta(S_0, w)$ describes the images of all states of $\mathscr{A}$ under the action of w. The following lemma is a trivial consequence of the definitions of D_1, D_2, D_3-directing words and D_1, D_2, D_3-positions.

Lemma 6. *Let $D \in \{D_1, D_2, D_3\}$. The word $w \in \Sigma^*$ is D-directing for the NFA $\mathscr{A}$ iff $\delta(S_0, w)$ is a D-position.*

Recall $\mathscr{A} = (Q, \Sigma, \delta)$, $|Q| = n$, $|\Sigma| = k$. Let us calculate without proof how much memory is sufficient to keep some objects and to do some operations which we need for the algorithm.

Lemma 7. *1. Every subset $T \subseteq Q$ can be kept using n bits of memory.*

2. *Every position $P \in Pos(\mathscr{A})$ can be kept using n^2 bits of memory.*
3. *Let $P, R \in Pos(\mathscr{A})$ and we need to check whether there is a letter $a \in \Sigma$ such that $\delta(P, a) = R$. It can be done using $O(n)$ of memory.*
4. *For $D \in \{D_1, D_2, D_3\}$ every position $P \in Pos(\mathscr{A})$ can be checked to be a D-position using $O(n)$ of memory.*

Let us describe the function $GetLetter(L, N)$. We denote $S[0] = S_0$.

```
Function GetLetter(L, N)
    m = ⌊log(L)⌋
    P[m + 1] = S[0]
    Return InPath(m, 2^m, L, N)
End Function
```

The function $GetLetter(L, N)$ uses the recursive function $InPath$. The function $InPath$ can return yes, no or some letter $a \in \Sigma$. Define the relation $\leq$ on the set $\Sigma \cup \{yes, no\}$. Put $no < yes < a$ for any $a \in \Sigma$. The sense of every returned letter shall be described later.

Let $D \in \{D_1, D_2, D_3\}$. By Lemma 5 we have the length of the shortest D-directing word for the NFA $\mathscr{A}$ is less than $C \cdot 2^n$ for some constant C. Let $\bar{m} = \lfloor \log(C \cdot 2^n) \rfloor + 1$. Our algorithm uses an additional memory of size at most $\bar{m} \cdot n^2 + O(n^2) = O(n^3)$. We keep at most $\bar{m} + 1$ positions $P[0], \ldots, P[\bar{m}]$ of the automaton $\mathscr{A}$. The NFA $\mathscr{A}$, the directing type $D \in \{D_1, D_2, D_3\}$ and the positions $P[0], \ldots, P[\bar{m}]$ are global variables for the function $InPath$.

```
Function InPath(t, M, L, N)
    res = no
    For Each P[t] ∈ Pos(A)
        If (M ≠ L) or (M = L and P[t] is a D-position)
            If t > 0 Then
                res_1 = InPath(t − 1, M − 2^(m−1), N)
                res_2 = InPath(t − 1, M + 2^(m−1), N)
            If t = 0 Then
                r =the number of the last nonzero bit in the number M − 1
                l =the number of the last nonzero bit in the number M + 1
                If M − 1 ≥ L Then
                    res_1 = yes
                Else
                    If there is a ∈ Σ such that δ(P[r], a) = P[0] Then
                        If M = N Then res_1 = a
                        Else res_1 = yes
                    If there is b ∈ Σ such that δ(P[0], b) = P[l] Then
                        If M = N − 1 Then res_2 = b
                        Else res_1 = yes
            If res_1 > no and res_2 > no and res ≤ yes Then
                    res = max(res, res_1, res_2)
    Return res
End Function
```

Due to space limitations we do not give a detailed proof of the algorithm correctness, but we attempt to describe how the algorithm works in Appendix.

The algorithm $GetLetter(L, N)$ returns the N-th letter of the first found D-directing word or *no* if there is no such word. It follows from Lemma 7 that all operations of the algorithm use polynomial space. It can be calculated that the algorithm uses $O(n^3)$ of additional memory i.e., additional memory of polynomial size. The proposition is proved.

By Proposition 3 we have that the problem 2-ZERO CURSYN is PSPACE-hard. From Proposition 4 we have that the problems SHORT D1, D2 and D3DIR WORD belong to PSPACE. Let $k \geq 2$. From Proposition 1 we have that all problems from the statement of Theorem 1 are PSPACE-complete. Theorem 1 is proved.

Acknowledgement. The author is grateful to Dr. D. Ananichev and Prof. M. Volkov for valuable help. He also acknowledges support from the Federal Education Agency of Russia, project 2.1.1/3537, and from the Russian Foundation for Basic Research, grant 09-01-12142.

References

1. Černý, J.: Poznámka k homogénnym eksperimentom s konecnými avtomatami. Mat.-Fyz. Cas. Slovensk. Akad. Vied. 14, 208–216 (1964) (in Slovak)
2. Pin, J.-E.: On two combinatorial problems arising from automata theory. Ann. Discrete Math. 17, 535–548 (1983)
3. Eppstein, D.: Reset sequences for monotonic automata. SIAM J. Comput. 19, 500–510 (1990)
4. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer, Heidelberg (2008)
5. Ito, M.: Algebraic Theory of Automata and Languages. World Scientific, Singapore (2004)
6. Martyugin, P.V.: A Lower Bound for the Length of the Shortest Carefully Synchronizing Words. Russian Mathematics (Iz. VUZ) 54(1), 46–54 (2010)
7. Gazdag, Z., Ivan, S., Nagy-Gyorgy, J.: Improved upper bounds on synchronizing non-deterministic automata. Information Processing Letters 109(17), 986–990 (2009)
8. Kozen, D.: Lower bounds for natural proof systems. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, pp. 254–266 (1977)

Appendix

The Proof of Proposition 1
1,2,3,4. The class of PFA with a zero is a subclass of the class of all PFA. Therefore ZERO CARSYN $\leq_p$ CARSYN, k-ZERO CARSYN $\leq_p$ k-CARSYN e.t.c. Let $\mathscr{A} = (Q, \Sigma, \delta)$ and the state $z \in Q$ be a zero in $\mathscr{A}$. For any word $w \in \Sigma$ we have $\delta(z, w) = z$ Therefore, the PFA $\mathscr{A}$ can be carefully synchronized only to the zero state z. We can consider the PFA $\mathscr{A}$ as a NFA. If the word $w \in \Sigma^*$ is D_2-directing for the NFA $\mathscr{A}$, then for any state $q \in Q$, we have $\delta(q, w) = \delta(z, w) = z \neq \emptyset$. Hence every D_2-directing word is carefully synchronizing for $\mathscr{A}$. Therefore ZERO CARSYN $\leq_p$ D2DIR, k-ZERO CARSYN $\leq_p$ k-D2DIR e.t.c.

5,6,7,8. Note that if some NFA is a PFA then any D_1 and D_3-directing word is a carefully synchronizing word. Therefore the problem CARSYN is a partial case of the problems D1DIR, D3DIR and the problem SHORT CARSYN WORD is a partial case of the problems SHORT D1 and D3DIR WORD e.t.c.

9,10. Checking the careful synchronizability (careful synchronizability for PFA with zero, D_1, D_2 or D_3 -directability) is a part of the problem SHORT CARSYN (ZERO CARSYN, D1DIR, D2DIR, D3DIR) WORD. Therefore the relations are evident.

11, 12. For any $k > 0$ it follows that $k+1$-PROBLEM is more complicated then k-PROBLEM. To prove this fact we can add one letter with identical action to any k-letter PFA (NFA) and obtain a $k+1$-letter automaton with the same properties of the synchronization.

13, 14. k-PROBLEM is a partial case of PROBLEM for any integer $k > 0$. Therefore, PROBLEM is more complicated then k-PROBLEM for any fixed k and k-PROBLEM $\leq_p$ PROBLEM. The proposition is proved.

The Proof of Lemma 1
Only the letter y can merge some states from Q_1 and Q_2. Therefore $u[|u|] = y$. The word u is defined on the state beg. On the other hand letters from Σ and the letter y are undefined on the state beg. Therefore $u[1] = x$. Hence $u = xwy$.

The image of the letter y contains only one element. Hence if $u[m] = y$ for some $m > 0$, then the word $w[1, m]$ is also a c.s.w. for PFA $\mathscr{B}$ and the word w is not the shortest. On the other hand only the letter y can merge some states from Q_1 and Q_2. Therefore the shortest c.s.w. u for automaton $\mathscr{B}$ contains only once the letter y and $u[|u|] = y$.

We have $u[1] = x$. Hence for any $i \in 1...k$, we have $|\delta(Q, u[1]) \cap Q_i| = 1$. All the letters from Σ and the letter x are defined on the set Q_i and map Q_i to Q_i. Therefore for any $m \in 1...|u| - 1$ we have $|\delta(Q, u[1, m]) \cap Q_i| = 1$. Hence if $u[m] = x$ then $\delta(Q, u[1, m]) = \{s_1, \ldots, s_k\} = \delta(Q, u[1])$ and the word $u[1]u[m+1, |u|]$ is also a c.s.w. This means that there is the only one letter x in the shortest c.s.w. u and $u[1] = x$. Thus $u = xwy$ for some $w \in \Sigma'^*$. The lemma is proved.

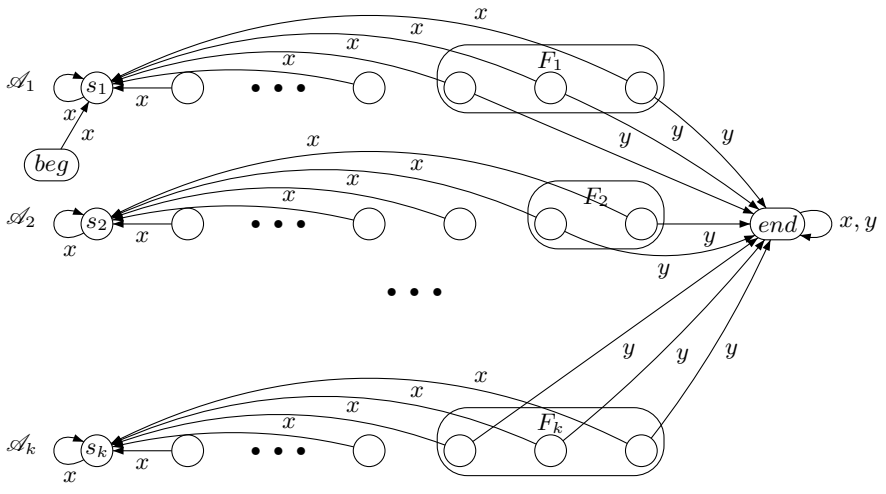

Fig. 2. Automaton $\mathscr{B}$

The Proof of Lemma 3

1. We have $v \in \{a,b\}^*\backslash\{a\}^*$. Therefore a letter b occurs in the word v. Let $v[h_1]$ be the first occurrence of the letter b in the word v. By Lemma 2 obtain $\gamma(P,b) \subseteq R_0$. Therefore $\gamma(P, v[1,h_1]) \subseteq R_0$. Therefore $|Q_i \cap \gamma(P, v[1,h_1])| \leq 1$. Let for some $j \in \{1,\ldots,|v|-1\}$ it holds $|Q_i \cap \gamma(P, v[1,j])| \leq 1$. If $v[j+1] = a$ then from Lemma 2 we have for any $i \in \{1,\ldots,n\}$ $\gamma(Q_i, a) \subseteq Q_i$. Therefore $|\gamma(P, v[1,j]a) \cap Q_i| \subseteq |\gamma(P, v[1,j]) \cap Q_i| \leq 1$. If $v[j+1] = b$ then $Q_i \cap \gamma(P, v[j+1]) \subseteq R_0$. This means $|Q_i \cap \gamma(P, v[j+1])| \leq 1$. Whence $|Q_i \cap \gamma(P,v)| \leq 1$.

2. Let $i \in \{1,\ldots,n\}$. We have $|Q_i \cap \gamma(P,v)| \leq 1$. By Lemma 2 we obtain $\gamma(Q_i, a) \subseteq Q_i$ for $i \in \{1,\ldots,n\}$. The letter a is defined on every state from the set P. Therefore if $|Q_i \cap \gamma(P,v)| = 1$, then $|Q_i \cap \gamma(P,va)| = 1$. Therefore

$$|\gamma(P,v)| = \sum_{i=1}^{n} |\gamma(P,v) \cap Q_i| = \sum_{i=1}^{n} |\gamma(P,va) \cap Q_i| = |\gamma(P,va)|.$$

3. By Lemma 2 we obtain $\gamma(P,b) \subseteq R_0$. Therefore $\gamma(P,vb) \subseteq \gamma(P,b) \subseteq R_0$. The lemma is proved.

The Description of the Function $InPath$**.**

Our goal is to find the set of positions $\{S[0],\ldots,S[L]\}$ such that there is a word $w \in \Sigma^*$ such that for $i \in \{1,\ldots,L\}$, we have $\delta(S[i-1], w[i]) = S[i]$ and $S[L]$ is a D-position. In such case the word w is a D-directing word for the NFA $\mathscr{A}$. Every position from the set $\{S[0],\ldots,S[L]\}$ will appear as a value of the one of the variables $P[0],\ldots,P[m+1]$. The position $P[m+1]$ is always equal to $S[0]$. The position $S[2^m]$ appears as a value of the variable $P[m]$. The positions $S[2^m - 2^{m-1}]$ and $S[2^m + 2^{m-1}]$ can appear as a values of $P[m-1]$.

The positions $S[2^m - 2^{m-1} - 2^{m-2}]$, $S[2^m - 2^{m-1} + 2^{m-2}]$, $S[2^m + 2^{m-1} - 2^{m-2}]$ and $S[2^m + 2^{m-1} + 2^{m-2}]$ can appear as values of $P[m-2]$ and so on. If there are exactly t zero bits at the end of the binary record of some number $M \in \{0, \ldots, L\}$, then the position $S[M]$ can appear as a value of $P[t]$. The length of the shortest D-directing word for $\mathscr{A}$ is L, therefore all the found sets $S[1], \ldots, S[L]$ are different.

When we call $InPath(s, M, L, N)$ we search a position $S[M]$. If $M = L$ then algorithm searches only D-positions because the last position should be a D-position. The number M has exactly t zero bits at the end of its binary record. We look over all the positions from $Pos(\mathscr{A})$ and put each of them into the variable $P[t]$. Let M be even. This means $t > 0$. Firstly, for any set $P[t]$ (which can be $S[M]$) we want to find a position $P[t-1]$ which can be $S[M - 2^{m-1}]$. If $P[t] = S[M]$ and $P[t-1] = S[M - 2^{m-1}]$ then there is a word v of length 2^{m-1} such that $\delta(P[t-1], v) = P[t]$. Secondly, we want to find a position $P[t-1]$ which can be $S[M + 2^{m-1}]$. If $P[t] = S[M]$ and $P[t-1] = S[M + 2^{m-1}]$ then there is a word v of length 2^{m-1} such that $\delta(P[t], v) = P[t-1]$. The function $InPath$ calls itself recursively to find the positions $S[M - 2^{m-1}]$ and $S[M + 2^{m-1}]$. If $L > M + 2^{m-1}$ then the position $S[M + 2^{m-1}]$ shell not be searched, but the algorithm shell search sets $S[M], \ldots, S[L]$ anyway.

If the sets $S[M - 2^{m-1}]$ and $S[M + 2^{m-1}]$ (or $S[L]$) are not found then the function $InPath(s, M, L, N)$ returns *no*. If the sets are found, then if $M + 2^{m-1} \geq N > M - 2^{m-1}$, then the function $InPath(s, M, L, N)$ returns the $N - M + 2^{m-1}$-th letter of the first found word v such that $\delta(S[M - 2^{m-1}], v) = S[M + 2^{m-1}]$ (or $\delta(S[M - 2^{m-1}], v) = S[L]$), if not $N \leq M - 2^{m-1}$ or $N > M + 2^{m-1}$ then the function $InPath(s, M, L, N)$ returns *yes*.

If M is odd, then $t = 0$. The variables r and l are the numbers of the last nonzero bit in the numbers $M - 1$ and $M + 1$. In this case the position $P[r]$ is suspected to be the position $S[M-1]$ and the position $P[l]$ is suspected to be the position $S[M+1]$. The variables $P[r]$ and $P[l]$ already contain some positions. The algorithm looks over all positions, puts each of them to a variable $P[0]$ and checks whether there are letters $a, b \in \Sigma$ such that $\delta(P[r], a) = P[0]$ and $\delta(P[0], a) = P[l]$. If the letters are found then $P[0] = S[M]$ and the function $InPath(s, M, L, N)$ returns *yes*, or the letter a if $M = N$, or the letter b if $M = N - 1$.