Checking if an Automaton is Monotonic is NP-complete

Marek Szykuła*

Institute of Computer Science, University of Wrocław, Joliot-Curie 15, PL-50-383 Wrocław, Poland
{msz@cs.uni.wroc.pl}

Abstract. An automaton is monotonic if its states can be arranged in a linear order that is preserved by the action of every letter. We prove that the problem of deciding if a given automaton is monotonic is NP-complete. This also holds under the restriction that the input alphabet is binary. The same result is obtained for oriented automata, whose states can be arranged in a cyclic order.

Keywords: automaton, monotonic, oriented, linear order, NP-complete, partial order, order-preserving, semigroup

1 Introduction

We deal with complete deterministic finite (semi)automata \( A = \langle Q, \Sigma, \delta \rangle \), where \( Q \) is the set of states, \( \Sigma \) is the input alphabet, and \( \delta: Q \times \Sigma \to Q \) is the transition function defining the action of \( \Sigma \) on \( Q \). This action naturally extends to the action of \( \delta(q, w) \) words for any \( q \in Q, w \in \Sigma^* \).

Monotonic automata are those that admit a linear order of the states. The same qualification is applied to transformation semigroups. Formally, an automaton \( A \) is monotonic if there exists a linear order \( \leq \) of \( Q \) such that if \( p \leq q \) then \( \delta(p, a) \leq \delta(q, a) \), for all \( p, q \in Q \) and \( a \in \Sigma \). We call such an order \( \leq \) an underlying order of \( A \). It is clear that if the actions of all letters preserve the order, then also the actions of all words do so.

The class of monotonic automata is a subclass of aperiodic ones [17], which recognize precisely star-free languages, and form one fundamental classes in the theory of formal languages. An automaton is aperiodic if no transformation of any word has a nontrivial cycle. Checking if an automaton is aperiodic is known to be PSPACE-complete [6]. Curiously, on the other hand, checking if an automaton is nonpermutational, where no transformation acts like a permutation of a nontrivial subset of \( Q \), can be easily done in \((|\Sigma| \times |Q|^2)\) time [11]. Such results may be useful in improving algorithms recognizing star-free languages to work better in particular cases.

Monotonic semigroups were studied by Gomes and Howie [9] for their maximum size (they use the term order-preserving. These semigroups play an important role as building-blocks in constructions of the largest aperiodic semigroups known so far [115]).

Monotonic automata have been considered, in particular, in connection with the problems of synchronizing automata. An automaton is said to be synchronizing if there is a word \( w \) such that \(|Qw| = 1\); such a word is called a reset word. The Černý conjecture, which is considered one of the most longstanding open problem in automata theory, states that every synchronizing automaton has a reset word of length at most \((|Q| - 1)^2\). Ananichev and Volkov [11] have proved that a synchronizing monotonic automaton has a reset word of length at most \(|Q| - 1\). They have also proved the same

* Supported in part by Polish NCN grant DEC-2013/09/N/ST6/01194.
bound for a larger class of generalized monotonic automata \[2\]. Volkov have introduced still larger class of weakly monotonic automata \[24\], which contains all aperiodic ones, and proved that strongly connected automata in this class possess a synchronizing word of length \(|Q|(|Q|+1)/6\). Finally, Grech and Kisielewicz have generalized this to the class of automata respecting intervals of a directed graph, and they have proved that the Černý conjecture holds for each automaton in this class, provided it holds for smaller quotient automata. These results could be also useful in computational verification of the conjecture for automata of limited size, provided we could efficiently recognize and skip from computations automata that belong to a class for which the conjecture has been proven \[13,14\]. Therefore it is important to consider computational complexity of the related problems.

The term monotonic was also used by Eppstein \[7\] for automata whose states can be arranged in a cyclic order that is preserved by the actions of the letters. Following \[1\] we call such automata oriented automata. They form a broader class, containing monotonic automata, which has certain applications in robotics (part-orienters, see Natarajan \[18\]). Eppstein has established the tight upper bound for the length of the shortest reset words of an oriented automaton \(|Q|−1)^2\), and provided an algorithm working in \((|Σ| × |Q|^2)\) time for finding such a word. However, this algorithm requires the cyclic order to be given.

Note that the problem of finding the shortest reset word is hard in general \[19\] (also for approximation \[18\] and some restricted classes \[13\]). But due to possible practical applications, there are many exponential algorithms that can deal with fairly large automata and polynomial heuristics (e.g. \[20,21,23,15,12\]). Also, hardness does not exclude a possibility of using a polynomial algorithm for some easily tractable classes (cf. slowly synchronizing \[12\]).

Here we prove that the problem of checking if a given automaton is monotonic is \(NP\)-complete, even under restriction to binary alphabets (Section 2). We also obtain that checking if an automaton is oriented is \(NP\)-complete under the same conditions (Section 3). It follows that, unfortunately, they are hardly recognizable, and it is hard to find a preserved linear (cyclic) order of a monotonic (oriented) automaton. In particular, we cannot efficiently apply the polynomial Eppstein algorithm \[7\] to compute a shortest reset word in the cases oriented automata, without knowing a cyclic order. On the other hand, checking if an automaton admits a nontrivial partial order is easy (Section 4).

2 Monotonic Automata

The problem MONOTONIC can be formulated as follows: given an automaton \(A\), decide if \(A\) is monotonic. This is the unrestricted version, where the alphabet can be arbitrary large. For a given \(k \geq 1\), the restricted problem to \(k\)-letter alphabets of the input automaton we call MONOTONIC\(_k\).

We show that MONOTONIC is \(NP\)-complete, as well as MONOTONIC\(_k\) for any \(k \geq 2\). The problem is easy if the alphabet is unary.

**Proposition 1.** A unary automaton is monotonic if and only if the transformation of the single letter does not contain a cycle of length \(\geq 2\). MONOTONIC\(_1\) can be solved in \(|Q|\) time, and a monotonic order can be found in \(|Q|\) time if it exists.

**Proof.** We simply check if the transformation of the single letter of \(A\) contains a cycle of length \(\geq 2\), that is \(δ(q_1,a) = q_2, δ(q_2,a) = q_3, \ldots , δ(q_t,a) = q_1\) for some distinct states \(q_1, \ldots , q_t\). If so, then from \(q_1 < q_2\) (or dually \(q_1 > q_2\)) it follows that \(q_2 < q_3, \ldots , q_t < q_1\)—a contradiction with that \(<\) is an order. Thus the automaton is not monotonic.

Otherwise we have an acyclic digraph of the transformation, and we can fix some order on the connected components (sometimes called clusters). Each such a component form a rooted tree. We
can perform an inverse depth-first search (DFS) starting from the root. Then \( p \leq q \) if \( p \) is in a component before that of \( q \), or they are in the same component but \( p \) was visited later than \( q \) during the inverse DFS in this component. So if \( p \leq q \) from the same component, then \( \delta(p, a) \) was visited later than \( \delta(q, a) \), or \( \delta(p, a) = \delta(q, a) \). Thus the order is preserved. These operations can be done in \((|Q|)\) time.

Clearly, MONOTONIC is in NP, as we can guess an underlying order and check if the action of each letter preserves it (this can be done in \((|\Sigma| \times |Q|)\) time).

**Proposition 2.** MONOTONIC is in NP.

### 2.1 MONOTONIC is NP-complete

We reduce MONOTONE-NAE-3SAT to MONOTONIC.

NAE-3SAT (NOT-ALL-EQUAL) is a variant of 3SAT, where a clause is satisfied if it contains at least one true and one false literal. The variant MONOTONE-NAE-3SAT additionally restricts instances so that every literal is a positive occurrence of a variable (negations are not allowed). From Schaefer's Theorem [22], we have that NAE-3SAT is NP-complete as well as MONOTONE-NAE-3SAT.

As an instance \( I \) of MONOTONE-NAE-3SAT we get a set of \( n \) boolean variables \( \mathcal{V} = \{v_1, \ldots, v_n\} \), and a set of \( m \) clauses \( \mathcal{C} = \{C_1, \ldots, C_m\} \), each one with exactly 3 literals. A literal is a positive occurrence of a variable \( v_i \). The problem is to decide if there exists a satisfying assignment \( \sigma: \mathcal{V} \rightarrow \{0, 1\} \) for \( I \), that is, for each clause \( C_i \in \mathcal{C}, C_i \) contains at least one true literal \( v_j \in C_i \) with \( \sigma(v_j) = 1 \) and at least one false literal \( v_j \in C_i \) with \( \sigma(v_j) = 0 \). We can assume that each variable occurs at least one time, and no variable appears more than once in a clause. Note that the complement of a satisfying assignment for \( I \) is also satisfying.

**Definition of \( \mathcal{A}_I \).** We construct \( \mathcal{A}_I = (Q, \Sigma, \delta) \) as follows. For each variable \( v_i \in \mathcal{V} \) we create a pair of states \( p_i, q_i \). We also add a unique state \( s \) (sink).

For a \( j \)-th clause \( C_j = (v_f, v_g, v_h) \) (we fix the order of variables in clauses), we create the clause gadget as follows. We add three states \( x_j, y_j, z_j \) and three letters \( a_j, b_j, c_j \), which correspond to the three occurrences of the variables \( v_f, v_g, v_h \), respectively. The action of these letters is defined as follows:

- \( \delta(p_f, a_j) = x_j \) and \( \delta(q_f, a_j) = y_j \);
- \( \delta(p_g, b_j) = y_j \) and \( \delta(q_g, b_j) = z_j \);
- \( \delta(p_h, c_j) = z_j \) and \( \delta(q_h, c_j) = x_j \);
- \( \delta(p_i, a_j) = p_i \) and \( \delta(q_i, a_j) = q_i \), for \( i < f \);
- \( \delta(p_i, b_j) = p_i \) and \( \delta(q_i, b_j) = q_i \), for \( i < g \);
- \( \delta(p_i, c_j) = p_i \) and \( \delta(q_i, c_j) = q_i \), for \( i < h \);
- \( \delta(u, e) = s \), for the other states \( u \) and each \( e \in \{a_j, b_j, c_j\} \).

So the actions of letters \( a_j, b_j, c_j \) send every state from \( Q \setminus \{p_i, q_i\} \) either to itself or to \( s \). The clause gadget is presented in Figure[1]

In Figure[2] the construction of \( \mathcal{A}_I \) is presented, with the action of \( a_1 \) as an example, in the case when variable \( v_2 \) is the first literal in clause \( C_1 \).

In summary, we have \(|Q| = 2n + 3m + 1\) states and \(|\Sigma| = 3m\) letters.
Correctness of the Reduction.

**Theorem 1.** \mathcal{A}_I is monotonic if and only if \mathcal{I} has a satisfying assignment.

**Proof.** Suppose that \mathcal{A}_I is monotonic with the underlying order \leq. We define an assignment \sigma for \mathcal{I}: \sigma(v_i) = 0 if \( p_i < q_i \), and \sigma(v_i) = 1 otherwise. We show that \sigma is satisfying for \mathcal{I}.

Assume for the contrary that there is a clause \( C_j = (v_f, v_g, v_h) \), where all the three variables evaluates to 0. This means that \( p_f < q_f \), \( p_g < q_g \), and \( p_h < q_h \). From that \leq is preserved, we have:

- \( \delta(p_f, a_j) = x_j < y_j = \delta(q_f, a_j) \);
- \( \delta(p_g, b_j) = y_j < z_j = \delta(q_g, b_j) \);
- \( \delta(p_h, c_j) = z_j < x_j = \delta(q_h, c_j) \).

Thus \( x_j < y_j < z_j < x_j \), a contradiction with that \leq is an order. The argument holds in the dual way in the case with all the three variables evaluated to 1. Hence, \sigma must be satisfying.

Now, suppose that there is a satisfying assignment \sigma. We define a linear order \leq and show that it is preserved. To do so, we define \( \tau: Q \to \mathbb{N} \), which for states \( q \in Q \) assigns pairwise distinct natural numbers that will determine \leq.

First, for any \( 1 \leq i \leq n \) let:
• $\tau(p_i) = 2i - 1$ and $\tau(q_i) = 2i$ if $\sigma(v_i) = 0$;
• $\tau(p_i) = 2i$ and $\tau(q_i) = 2i - 1$ if $\sigma(v_i) = 1$.

For $u \in \{x_j, y_j, z_j\}$ we define $\tau(u) \in \{2n + 3j - 2, 2n + 3j - 1, 2n + 3j\}$, depending on the assignment of the variables in $C_j = (v_f, v_g, v_h)$. Assignment $\sigma$ uniquely determines the relation between $x_j, y_j, z_j$ in an underlying order. Each of the six satisfying combinations of $\sigma(v_f), \sigma(v_g), \sigma(v_h)$ defines an acyclic relation between $x_j, y_j, z_j$, which is enforced by the action of the letters $a_j, b_j, c_j$.

For instance, if $\sigma(v_f) = 0$, then $p_f < q_f$, which implies $\delta(p_f, a_j) = x_j < y_j = \delta(q_f, a_j)$. If $\sigma(v_g) = 0$ then $y_j < z_j$. Then it must be $\sigma(v_h) = 1$ and $z_j > x_j$. If $\sigma(v_g) = 1$ then $y_j > z_j$, and we have either $z_j < x_j < y_j$ if $\sigma(v_h) = 0$, or $x_j < z_j < y_j$ otherwise. This is dual for $\sigma(v_f) = 1$.

Finally we define $\tau(s) = 3n + 3m + 1$. Hence, in our order $\leq$, first there are states $p_i, q_i$ sorted increasingly by $i$. The order between $p_i$ and $q_i$ depends on the assignment. Next, there are states from clause gadgets $x_j, y_j, z_j$ sorted by $j$. The order is shown in Figure 3 (from left to right).

Now we show that $\leq$ is indeed underlying. Consider a letter $a_j$ for any $1 \leq j \leq k$, and let $C_j = (v_f, v_g, v_h)$. We show that for every pair of distinct states the order $\leq$ is preserved.

• For the pair $p_f, q_f$, if $p_f < q_f$ then also $\delta(p_f, a_j) = x_j < y_j = \delta(q_f, a_j)$, and if $p_f > q_f$ then $\delta(p_f, a_j) = x_j > y_j = \delta(q_f, a_j)$, since we have chosen the order of $x_j, y_j, z_j$ to be consistent with $\sigma$, as described above.

• For $p_f$ (or $q_f$) and $u \in Q \setminus \{p_f, q_f\}$, if $p_f < u$ then $\delta(p_f, a_j) = x_j < \delta(u, a_j) = s$. If $u < p_f$ then $\delta(u, a_j) = u < x_j = \delta(p_i, a_i^r)$.

• For distinct states $u, v \in Q \setminus \{p_f, q_f\}$ with $u < v$, if $\delta(u, a_j) = s$ then either $\delta(v, a_j) = s$ or $\delta(v, a_j) = s < u$.

The same arguments work for letters $b_j$ and $c_j$. It follows that any letter preserves $\leq$, so $\leq$ is an underlying order of $A_I$.

We can state our main

**Theorem 2.** The problem of checking if a given automaton is monotonic is NP-complete.

### 2.2 Reduction from MONOTONIC to MONOTONIC$_2$

Let $A = (Q, \Sigma, \delta)$ be an automaton with $Q = \{v_1, \ldots, v_n\}$ and $\Sigma = \{a_1, \ldots, a_k\}$ with $k \geq 3$. We construct a binary automaton $B_A = (Q_B, \{a, b\}, \delta_B)$ such that $A$ is monotonic if and only if $B_A$ is monotonic.

$Q_B$ consists of $kn$ states $q_j^i$ for $1 \leq i \leq k, 1 \leq j \leq n$, and a unique state $s$ (sink). Now we define the action of $a$. For each state $q_j^i$ with $1 \leq i \leq k - 1$ and $1 \leq j \leq n$, we define $\delta_B(q_j^i, a) = \delta_B(q_j^{i+1})$. For each $q_k^i$ we define $\delta_B(q_k^i, a) = s$. Finally $\delta(s, a) = s$. The action of $b$ in each set $\{q_1^i, \ldots, q_n^i\}$ corresponds to the action of the $i$-th letter of $\Sigma$ on $Q$: For $1 \leq i \leq k$ and $1 \leq j \leq n$, if $\delta(v_j, a_i) = v_g$ then we define $\delta_B(q_j^i, b) = q_g$. Finally $\delta(s, b) = s$. The construction of $B_A$ is shown in Figure 3.

**Theorem 3.** $B_A$ is monotonic if and only if $A$ is monotonic.

**Proof.** Suppose that $A$ is monotonic with the underlying order $\leq_A$. We define the linear order $\leq_B$ on the states of $B_A$. For $1 \leq i, f \leq k$ and $1 \leq j, g \leq n$, let $q_j^i \leq_B q_g^f$ if and only if $i < f$, or $i = f$
and \( v_j <_A v_g \). Also, let \( q^j \in \mathcal{B} \subseteq s \) for each \( i, j \). The order \( \leq_B \) is linear, since \( \leq_A \) is linear. We show that \( \leq_B \) is an underlying order of \( B_A \).

Clearly, the actions of both letters preserve \( \leq_B \) on states \( q^j \) and \( s \). Consider a pair \( q^j, q^g \) with \( q^j \leq q^g \). Then \( i \leq f \) by definition. Consider the following cases:

- If \( i < f \), then \( \delta_B(q^j, a) = \delta_B(q^{j+1}, a) <_B \delta_B(q^g, a) \), since \( \delta_B(q^j, a) \) is either \( q^{f+1} \) or \( s \). Also, for some \( x, y \), \( \delta_B(q^j, b) = q^i <_B q^f = \delta_B(q^g, b) \), since \( i < f \).
- If \( i = f \), then \( v_j <_A v_g \) by definition. If \( i = k \) then \( \delta_B(q^j, a) = \delta_B(q^g, a) = s \); otherwise \( \delta_B(q^j, a) = q^{i+1} \leq_B q^{i+1} = \delta_B(q^j, a) \) from \( v_j <_A v_g \). Also, \( v_j <_A v_g \) implies \( \delta(v_j, a) = v_x \leq \delta_B(q^j, b) \).

Thus \( \leq_B \) is an underlying order of \( B_A \).

Now, suppose that \( B_A \) is monotonic with an underlying order \( \leq_B \). We define \( \leq_A \) on the states of \( A \): for \( 1 \leq j, g \leq n, v_j <_A v_g \) if and only if \( q^j \leq q^g \). Observe that for any \( j \neq g \), \( q^j \leq q^g \) implies \( q^j \leq q^k \) for each \( 2 \leq i < k \) due to the action of \( a \). Consider two states \( v_j, v_g \) with \( v_j < v_g \) and the \( i \)-th letter \( a_i \). By definition \( q^{j_i} \leq q^{g_i} \), and so \( q^j \leq q^g \). This implies \( \delta_B(q^j, b) = q^i <_B q^k = \delta_B(q^g, b) \) for some \( x, y \), and it follows that \( q^x \leq q^y \). Thus \( \delta(v_j, a_i) = v_x \leq v_g = \delta(v_g, a_i) \), and the order \( \leq_A \) is an underlying order of \( A \).

As a corollary we obtain that \( \text{MONOTONIC}_2 \) is also NP-complete. We can reduce an instance of \( \text{MONOTONE-NAE-3SAT} \) with \( n \) variables and \( m \) clauses to a binary automaton with \( 3m(2n + 3m + 1) + 1 \) states.

**Corollary 1.** The problem of checking if a given binary automaton \( A \) is monotonic is NP-complete.

### 3 Oriented Automata

The following definition of oriented automata is due to Eppstein [2] (who used the monotonic term). An automaton is *oriented* if there is a cyclic order of the states preserved by the action of the letters. Formally, there is a cyclic order \( q_1, \ldots, q_n \) such that for every \( a \in \Sigma \), the sequence \( \delta(q_1, a), \ldots, \delta(q_n, a) \), after removal of possibly adjacent duplicate states (the last is also adjacent with the first), is a subsequence of a cyclic permutation \( q_1, \ldots, q_n, q_1, \ldots, q_{n-1} \) of the cyclic order.
for some $1 \leq i \leq n$. Note that if $q_1, \ldots, q_n$ is a cyclic order then also $q_i, q_{i+1}, q_{i+2}, \ldots, q_{i+n-1}$ is for every $1 \leq i \leq n$. Every monotonic automaton is oriented, since if a linear order is preserved, then it is also preserved as a cyclic order. But the converse does not necessarily hold.

Let ORIENTED be the problem of deciding if a given automaton is oriented. As before, we consider $\text{ORIENTED}_k$ with restriction to $k$-letter alphabets. Again, $\text{ORIENTED}_1$ can be easily solved in $(n)$ time due to the following

**Proposition 3.** A unary automaton is oriented if and only if all cycles in the transformation of the single letter have the same length. There is an algorithm solving the problem $\text{ORIENTED}_1$ and finding a cyclic order if it exists, which works in $(|Q|)$ time.

**Proof.** Let $a$ be the single letter of the alphabet and $n = |Q|$. Suppose that $(c_1, \ldots, c_k)$ and $(d_1, \ldots, d_{\ell})$ are two cycles in the transformation of $a$, with $1 \leq k < \ell$. Then, the transformation of $a^k$ has the cycle $(d_1, d_{\ell}, \ldots, d_{(m-1)k \mod \ell})$ of length $m$, for some $2 \leq m \leq \ell$. On the other hand it has the fixed point $c_1$. Let $q_1, \ldots, q_{n-1}, q_n = c_1$ be a cyclic order of the states of the automaton. Since the transformation of $a^k$ has a cycle of length $\geq 2$ (which does not involve $c_1$), there are two states $q_i, q_j$ with $i < j < n$, $\delta(q_i, a^k) = q_j$, and $\delta(q_j, a^k) = q_i$, such that $f > g$. It follows that $\delta(q_i, q_j) = q_j$, $\delta(q_j, c_1) = q_i$, $\delta(c_1, a^k) = c_1$ violates the cyclic order $q_1, \ldots, q_{n-1}, q_n = c_1$, since $(q_j, q_i, c_1)$ is not a subsequence of any cyclic permutation of the cyclic order—a contradiction.

Assume now that we have $m$ cycles of the same length $k$:

$$(c_1^1, \ldots, c_k^1), (c_1^2, \ldots, c_k^2), \ldots, (c_1^m, \ldots, c_k^m),$$

so $c_i^j$ is the $i$-th state of the $j$-th cycle, and $\delta(c_i^j) = c_i^{j \mod k+1}$. We can compute a cyclic order by a breadth-first search (BFS) in the inverse digraph of the transformation of $a$. The constructed cyclic order will have the form

$$Q_1^1, c_1^1, \ldots, Q_m^1, c_k^m, \ldots, Q_1^k, c_1^k, \ldots, Q_k^m, c_k^m,$$

where $Q_i^j$ are sequences of states that do not lie on a cycle. Let $\ell(q)$ (level) be the smallest integer $i$ such that $\delta(q, a^i)$ is a state on a cycle. To simplify the notation, let $i \oplus j = (i - 1 + j) \mod k + 1$.

The algorithm starts from the list $(c_1^1, \ldots, c_k^m, \ldots, c_1^1, \ldots, c_k^m)$ of all cycle states; they are considered as visited in the 0-th step in this order. In the $i$-th step ($i \geq 1$), the algorithm processes the list of visited states from the $(i - 1)$-th step in the order in which they were visited. For each state $p$ from the list, the algorithm computes all states $q$ such that $\delta(q, a^i) = p$ and $q$ is not a cycle state; so it visits precisely all the states $q$ with $\ell(q) = i$. For every visited state $q$, it appends $q$ to the end of the new list of visited states in the current step. For a visited $q$, we have the corresponding cycle state $c_i^{q_0} = \delta(q, a^{\ell(q)})$, from which $q$ was reached (possibly indirectly). The algorithm appends $q$ to the beginning of $Q_j^i$ with $j = (nk + g - 1 - i) \oplus 1$; for example, if $g = 1$, then for $i = 1, 2, 3, \ldots, k - 1, k, k + 1, \ldots$ we have $j = k, k - 1, \ldots, 2, 1, k, \ldots$, respectively.

To illustrate the algorithm, consider the automaton from Figure 4. We start from the list $(c_1, c_2, c_3) = (v_2, v_6, v_9)$ of the one cycle, and empty $Q_1, Q_2, Q_3$. In the first step, from state $v_2$ we reach $v_8$, from $v_4$ we do not reach any state, and from $v_9$ we reach $v_5$. Hence, $Q_3 = (v_8)$ as $v_2 = c_1$, and $Q_2 = (v_9)$ as $v_9 = c_3$. Then, in the second step, from $v_8$ we reach $v_4$ and $v_3$, and from $v_5$ we do not reach any state; hence, we append $v_4$ and $v_3$ to the beginning of $Q_2$, obtaining $Q_2 = (v_3, v_4, v_5)$. In the third step, from $v_4$ we reach $v_1$, so $Q_1$ becomes $(v_1)$. Finally, in the last fourth step, from $v_1$ we reach $v_7$, obtaining $Q_3 = (v_7, v_8)$. The final order is so

$$Q_1, c_1, Q_2, c_2, Q_3, c_3 = v_1, v_2, v_3, v_4, v_5, v_7, v_8, v_9.$$
We can show that the resulted cyclic order is indeed preserved by the action of a. Observe that \( \delta(c_i^j, a) = c_i^{j+1} \), and if \( q \in Q_i^j \) then \( \delta(q, a) \in Q_i^{j+1} \) or \( \delta(q, a) = c_i^{j+1} \). Hence, the sequence \( Q_i^j \), \( c_i^j \) is mapped into \( Q_i^{j+1}, c_i^{j+1} \), and it remains to show that for each \( Q_i^j = (p_1, \ldots, p_s) \), the sequence \( \delta(p_1, a), \ldots, \delta(p_s, a), \delta(c_i^j, a) \) is a subsequence of \( Q_i^{j+1}, c_i^{j+1} \). Consider \( < \) as the order in these sequences, and let \( u, v \) be two states from \( Q_i^{j+1} \cup \{ c_i^{j+1} \} \) with \( u < v \). If \( v = c_i^{j+1} \) then we have \( \delta(u, a) \leq \delta(v, a) = c_i^{j+1} \). If \( u = p_f, v = p_g \), then \( u < v \) means that the algorithm appended \( u \) after \( v \), so \( u \) was visited after \( v \). They were directly reached from \( \delta(u, a) \) and \( \delta(v, a) \), respectively. If \( \delta(v, a) = c_i^j \) then \( \delta(u, a) \leq \delta(v, a) \) clearly holds, and if \( \delta(u, a) = c_i^j \) then also \( \delta(v, a) = c_i^j \). Otherwise, \( \delta(u, a), \delta(v, a) \in Q_i^{j+1} \) and it follows that \( \delta(u, a) \) was visited after \( \delta(v, a) \) by the algorithm, so \( \delta(u, a) > \delta(v, a) \). As usual breadth-first search, this procedure works in \(|Q|\) time.

To show hardness, we can simply reduce reduce the NP-complete problems MONOTONIC and MONOTONIC\(_k\) (with \( k \geq 2 \)) to ORIENTED and ORIENTED\(_k\), respectively.

**Proposition 4.** Let \( A_{n+1} \) be an automaton obtained from \( A = (Q, \Sigma, \delta) \) by adding a unique state \( s \) with \( \delta(s, a) = s \) for every \( a \in \Sigma \). Then the following are equivalent:

- \( A \) is monotonic;
- \( A_{n+1} \) is monotonic;
- \( A_{n+1} \) is oriented.

**Proof.** Clearly \( A_{n+1} \) is monotonic if and only if \( A \) is monotonic, and if \( A_{n+1} \) is monotonic then it is also oriented. It remains to show that if \( A_{n+1} \) is oriented then \( A_{n+1} \) is monotonic.

Assume that \( A_{n+1} \) is not monotonic but is oriented, and let \( q_1, \ldots, q_n, s \) be a preserved cyclic order of the states of \( A \). Since no state is mapped to \( s \), except \( s \), and \( s \) is mapped to itself under the action of any letter, \( q_1, \ldots, q_n \) is a preserved cyclic order of the states of \( A \). Since \( A \) is not monotonic, \( q_1, \ldots, q_n \) is not an underlying order of \( A \). So there are two states \( q_i, q_j \in Q \) and \( a \in \Sigma \), with \( i < j \), \( \delta(q_i, a) = q_f \), and \( \delta(q_j, a) = q_g \), such that \( f > g \). It follows that \( \delta(q_i, a) = q_f, \delta(q_j, a) = q_g, \delta(s, a) = s \) violates the cyclic order \( q_1, \ldots, q_n \), of the states of \( A \), since \( (q_f, q_g, s) \) is not a subsequence of any cyclic permutation of the cyclic order. Thus \( A_{n+1} \) cannot be oriented and not monotonic. □
Corollary 2. The problem of checking if a given automaton is oriented is NP-complete, even under restriction to binary alphabets.

4 Discussion

We have proved that checking if an automaton is monotonic or oriented is NP-complete. We leave open the problems of checking if an automaton is generalized [22] or weakly monotonic [24]. Also it remains open what is the complexity of checking if an automaton respects intervals of a directed graph [10].

It can be observed that if the alphabet is unary then the classes of generalized and weakly monotonic automata are precisely the class of monotonic automata. However, it is not difficult to check that automata $A_I$ from the construction from Subsection 2.1 are generalized, and so weakly monotonic, regardless of the instance $I$.

On the other hand, for a wider class of automata preserving a nontrivial partial order, the membership problem can be easily solved in polynomial time. An automaton preserves a partial order $\leq$, if $p \leq q$ implies $\delta(p, a) \leq \delta(q, a)$ for every $p, q \in Q$, $a \in \Sigma$. A partial order is nontrivial if at least one pair of states is comparable. In contrast to monotonic automata, not all pairs of states must be comparable, but at least one. This class contains monotonic, generalized, and weakly monotonic automata (but not oriented), and is a subclass of automata respecting intervals of a directed graph. From [10] it follows that if the Černý conjecture is true for all automata outside this class, it is true for all automata.

Proposition 5. Checking if an automaton preserves a nontrivial partial order and finding it if exists can be done in $(|\Sigma| \times |Q|^6)$ time and $(|Q|^2)$ working space.

Proof. For each pair of distinct states $p, q \in Q$, we try to construct a partial order $<_{\Sigma}$ with $p < q$. So at the beginning of constructing, all states incomparable and we order $p < q$. When ordering a pair $x, y \in Q$ with $x < y$, we take all the consequences $\delta(x, a) < \delta(y, a)$ for every $a \in \Sigma$ with $\delta(x, a) \neq \delta(y, a)$. Of course, this also involves that $x' < y'$ for every $x' < \delta(x, a)$ and $y' > \delta(y, a)$. For each newly ordered pair we repeat the procedure of taking consequences. If a contradiction is found, that is, if we need to order $x < y$ but they have been already ordered so that $x > y$, the construction fails and we start from another pair $p, q$. If for some pair $p, q$ all the consequences are taken without a contradiction, we have found a preserved partial order with $p < q$.

Clearly, if the algorithm finds a partial order, then $x \leq y$ implies $\delta(x, a) \leq \delta(y, a)$ as it has taken all the consequences, so the order is preserved. Conversely, if there exists a preserved nontrivial partial order $\leq$, then $p < q$ for some pair of states, and the consequences cannot lead to a contradiction. Hence, the algorithm will find the minimal partial order with $p < q$ that is preserved and is contained in $\leq$.

Concerning the complexity, we need to process $(|Q|^2)$ pairs. The constructed partial order can be simply stored as a directed acyclic graph. For every $p, q$, we start from the empty digraph with one edge $(p, q)$. For each ordered pair $(x, y)$ we need to take or check $(|\Sigma|)$ consequences, and we order $(|Q|^2)$ pairs. Taking a consequence and updating the constructed partial order takes $(|Q|^2)$ time, due to the possibly quadratic size of $\{z \in Q \mid z < x\} \times \{z \in Q \mid z > y\}$. These together yield in $(|\Sigma| \times |Q|^6)$ time, and the need of storing digraphs yields in $(|Q|^2)$ space. □

Remark 1. The algorithm from Proposition 5 may be modified for finding an underlying order of the given automaton. To do so, after finding a partial order that is not linear, we need to order
another pair that is not yet comparable, say \( \{x, y\} \). Here we must consider both possibilities \( x < y \) and \( x > y \), to check if one of them finally leads to a linear order. Hence, this results in exponential running time. However, despite having similar worst-case complexity, based on some experimental evidence this algorithm is practically much more efficient than the naive checking of all linear orderings.

References

1. D. Ananichev and M. Volkov. Synchronizing monotonic automata. In *Developments in Language Theory*, volume 2710 of *LNCS*, pages 111–121. Springer, 2003.
2. D. Ananichev and M. Volkov. Synchronizing generalized monotonic automata. *Theoretical Computer Science*, 330(1):3–13, 2005.
3. M. V. Berlinkov. Approximating the minimum length of synchronizing words is hard. In *Computer Science – Theory and Applications*, volume 6072 of *LNCS*, pages 37–47. Springer, 2010.
4. J. Brzozowski, B. Li, and D. Liu. Syntactic complexities of six classes of star-free languages. *Journal of Automata, Languages and Combinatorics*, 17(2-4):83–105, 2012.
5. J. Brzozowski and M. Szykuła. Large Aperiodic Semigroups. In *Implementation and Application of Automata*, volume 0 of *LNCS*, pages 0–0. Springer, 2014.
6. S. Cho and D. T. Huynh. Finite-automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88(1):99–116, 1991.
7. D. Eppstein. Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19:500–510, 1990.
8. M. Gerbush and B. Heeringa. Approximating minimum reset sequences. In *Implementation and Application of Automata*, volume 6482 of *LNCS*, pages 154–162. Springer, 2011.
9. G. Gomes and J. Howie. On the ranks of certain semigroups of order-preserving transformations. *Semigroup Forum*, 45:272–282, 1992.
10. M. Grech and A. Kisielewicz. The Černý conjecture for automata respecting intervals of a directed graph. *Discrete Mathematics and Theoretical Computer Science*, 15(3):61–72, 2013.
11. S. Iván and J. Nagy-György. On nonpermutational transformation semigroups with an application to syntactic complexity. [http://arxiv.org/abs/1402.7289](http://arxiv.org/abs/1402.7289), 2014.
12. A. Kisielewicz, J. Kowalski, and M. Szykuła. Computing the shortest reset words of synchronizing automata. *Journal of Combinatorial Optimization*, 2013.
13. A. Kisielewicz and M. Szykuła. Generating Small Automata and the Černý Conjecture. In *Implementation and Application of Automata*, volume 7982 of *LNCS*, pages 340–348. Springer, 2013.
14. A. Kisielewicz and M. Szykuła. Synchronizing Automata with Large Reset Lengths. [http://arxiv.org/abs/1404.3311](http://arxiv.org/abs/1404.3311), 2014.
15. R. Kudlacek, A. Roman, and H. Wagner. Effective synchronizing algorithms. *Expert Systems with Applications*, 39(14):11746–11757, 2012.
16. P. V. Martyugin. Complexity of problems concerning reset words for some partial cases of automata. *Acta Cybernetica*, 19:517–536, 2009.
17. R. McNaughton and S. A. Papert. *Counter-Free Automata*, volume 65 of *MIT Research Monographs*. The MIT Press, 1971.
18. B. K. Natarajan. An algorithmic approach to the automated design of parts orienters. In *Foundations of Computer Science, 27th Annual Symposium on*, 1986.
19. J. Olschewski and M. Ummels. The complexity of finding reset words in finite automata. In *Mathematical Foundations of Computer Science*, volume 6281 of *LNCS*, pages 568–579. Springer, 2010.
20. J.-K. Rho, Somenzi F., and C. Pixley. Minimum length synchronizing sequences of finite state machine. In *Proceedings of the 30th ACM/IEEE Design Automation Conference*, DAC ’93, pages 463–468, 1993.
21. S. Sandberg. Homing and synchronizing sequences. In *Model-Based Testing of Reactive Systems*, volume 3472 of *LNCS*, pages 5–33. Springer, 2005.
22. T. J. Schaefer. The Complexity of Satisfiability Problems. In Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, STOC, pages 216–226. ACM, 1978.
23. E. Skvortsov and E. Tipikin. Experimental study of the shortest reset word of random automata. In Implementation and Application of Automata, volume 6807 of LNCS, pages 290–298. Springer, 2011.
24. M. Volkov. Synchronizing automata preserving a chain of partial orders. Theoretical Computer Science, 410(37):3513–3519, 2009.