Complexity Analysis: Finite Transformation Monoids

Christian Brandl and Hans Ulrich Simon

Department of Theoretical Computer Science,
Faculty of Mathematics,
Ruhr-University Bochum,
44780 Bochum, Germany
{christian.brandl,hans.simon}@rub.de

Abstract. We examine the computational complexity of some problems from algebraic automata theory and from the field of communication complexity: testing Green's relations (relations that are fundamental in monoid theory), checking the property of a finite monoid to have only Abelian subgroups, and determining the deterministic communication complexity of a regular language. By well-known algebraizations, these problems are closely linked with each other. We show that all of them are PSPACE-complete.

Keywords: Green's relations, finite monoids, regular languages, communication complexity, PSPACE-completeness.

1 Introduction

The Green's relations $\mathcal{L}, \mathcal{R}, \mathcal{J}, \mathcal{H}$ are ubiquitous tools for studying the buildup of a (finite) monoid M. For example, the maximal subgroups of M can be characterized as $\mathcal{H}$-classes of M containing an idempotent element (e.g. [7]). Such $\mathcal{H}$-classes are called *regular*. As illustrated in [3], there are also important applications in automata theory: star-free languages, factorization forests, and automata over infinite words. The former application, due to Schützenberger, characterizes the star-free languages as regular languages with syntactic monoids having only trivial subgroups [8]. Finite monoids having only trivial subgroups are called *aperiodic* and form a variety denoted as **A**. In [2], Cho and Huynh prove Stern's conjecture from [9] that testing for aperiodicity (star-freedom, alternatively) is PSPACE-complete if the regular language is given as a minimum DFA. **A** is contained in the variety $\overline{\mathbf{Ab}}$ of monoids having only Abelian subgroups (regular $\mathcal{H}$-classes, alternatively). $\overline{\mathbf{Ab}}$ plays a decisive role for the communication complexity of a regular language [10]. Since its introduction by Yao [11], communication complexity has developed to one of the major complexity measures with plenty of applications (e.g., listed in [6]). In [10], Tesson and Thérien categorize the communication complexity of an arbitrary regular language L by some properties of its underlying syntactic monoid $M(L)$. This algebraic classification can be achieved in the following models of communication: deterministic, randomized (bounded error), simultaneous, and Mod_p-counting. For the deterministic model, the authors of [10] show that the communication complexity of L can only be constant, logarithmic, or linear (in the sense of Θ-notation). Thereby, the condition $M(L) \notin \overline{\mathbf{Ab}}$ is a sufficient (but not necessary) condition for the linear case.

In this paper, we focus on monoids which are generated by mappings with domain and range Q for some finite set Q (like the syntactic monoid where these mappings are viewed as state transformations). We primarily analyze the computational complexity of problems related to Green's relations, the monoid-variety $\overline{\mathbf{Ab}}$, and the deterministic communication complexity of regular languages. Our main contributions are summarized in the following theorem:

Theorem 1. *(Main results) Let M be a finite monoid given by the generators (=state transformations) $f_1, \ldots, f_l$. Let g, h be 2 elements of M. Let $\mathcal{G} \in \{\mathcal{L}, \mathcal{R}, \mathcal{J}, \mathcal{H}\}$ be one of Green's relations w.r.t. the monoid M. Let A be a minimum DFA with syntactic monoid $M(A)$. Let N be an NFA recognizing language L. Let $cc(L)$ be the deterministic communication complexity of L. With these notations, the following problems are PSPACE-complete:*

Problem	**Input**	**Question**
(i) $\mathcal{G}$-TEST	$g, h \in M$ and generators $f_1, \ldots, f_l$	$g \; \mathcal{G} \; h$?
(ii) DFA-$\overline{\mathbf{Ab}}$	minimum DFA A	$M(A) \in \overline{\mathbf{Ab}}$?
(iii) NFA-CC	NFA N	Is $cc(L)$ const., log., or lin.?

This paper is organized as follows. In Section 2, we introduce the necessary background and notations. In Section 3, we show the PSPACE-hardness of deciding Green's relations. In Section 4, we show the PSPACE-hardness of the problem DFA-$\overline{\mathbf{Ab}}$. In Section 5.1, we show that all problems listed in Theorem 1 are members of PSPACE even when the underlying monoid is a syntactic monoid of a regular language that is given by an NFA. In Section 5.2, we show the PSPACE-hardness of the problem NFA-CC. Moreover, this problem remains PSPACE-hard even when the language L is given by a regular expression β at the place of the NFA N.

2 Preliminaries

We assume that the reader is familiar with the basic concepts of computational complexity (e.g. [4]), communication complexity (e.g. [6]), regular languages (e.g. [4]), and algebraic automata theory (e.g. [7]). In the sequel, we will briefly recapitulate some definitions and facts from these fields and thereby fix some notation.

A Deterministic Finite Automaton (DFA) is formally given as a 5-tuple $A = (Q, \Sigma, \delta, s, F)$ where Q denotes the finite set of states, Σ the input alphabet,

$\delta : Q \times \Sigma \to Q$ the transition function, $s \in Q$ the initial state and $F \subseteq Q$ the set of final (accepting) states. As usual, the mapping δ can be extended by morphism to $\delta^* : Q \times \Sigma^* \to Q$. Throughout the paper, we will make use of the notation $q \cdot w := \delta^*(q, w)$. Intuitively, $q \cdot w$ is the state that is reached after A, started in state q, has completely processed the string w. The change from state q to state $q \cdot w$ is sometimes called the *w-transition from q*. The language recognized by A is given by $L(A) = \{w \in \Sigma^* : s \cdot w \in F\}$. Languages recognizable by DFA are called *regular*. The DFA with the minimal number of states that recognizes a regular language L is called the *minimum DFA* for L. DFA-minimization can be executed efficiently.

Let $L \subseteq \Sigma^*$ be a formal language. We write $w_1 \sim_L w_2$ iff the equivalence $uw_1v \in L \Leftrightarrow uw_2v \in L$ holds for every choice of $u, v \in \Sigma^*$. $\sim_L$ defines a congruence relation on Σ^* named *syntactic congruence*. For every $w \in \Sigma^*$, $[w]_{\sim_L}$ denotes the equivalence class represented by w. The quotient monoid $\Sigma^*/\sim_L$, denoted as $M(L)$, is called the *syntactic monoid* of L. $M(L)$ is finite iff L is regular. Moreover, $M(L)$ coincides with the monoid consisting of the state transformations $f_w(q) := q \cdot w$ of a minimum DFA for L. Clearly, this monoid is generated by $\{f_a : a \in \Sigma\}$. If $L = L(A)$ for some DFA A, we often simply write $M(A)$ instead of $M(L(A))$. The analogous convention applies to NFA.

Let M be a monoid and $a, b \in M$ two arbitrary elements. $a\mathcal{J}b :\Leftrightarrow MaM = MbM$; $a\mathcal{L}b :\Leftrightarrow Ma = Mb$; $a\mathcal{R}b :\Leftrightarrow aM = bM$; $a\mathcal{H}b :\Leftrightarrow a\mathcal{L}b \wedge a\mathcal{R}b$ define four equivalence relations on M named *Green's relations*. An element $e \in M$ is called *idempotent* iff $e^2 = e$. A *subgroup* of M is a subsemigroup of M that is a group.

We denote by PSPACE the class of all problems that can be solved by a Deterministic Turing-Machine (DTM) in polynomial space. We use the symbol $\mathcal{M}$ for such a DTM. PSPACE is closed under complement. The following two decision problems FAI [5] and RENU [1] are known to be complete problems for PSPACE. FAI: Given DFAs $A_1, \ldots, A_k$ with a common input alphabet and unique final states, is there an input word accepted by all of $A_1, \ldots, A_k$? RENU: Given a regular expression β over Σ (i.e., an expression with operands from $\Sigma \cup \{\varepsilon\}$ and operations "+" (for union), "·" (for concatenation) and "$*$" (for Kleene closure)), do we have $L(\beta) \neq \Sigma^*$?[1]

Let $L \subseteq \Sigma^*$ be a formal language. Let ε denote the empty string. In the so-called communication game, 2 parties, say X and Y, exchange bits in order to decide if the string $x_1y_1 \ldots x_ny_n$ belongs to L. Thereby, X (resp. Y) only knows $(x_1, \ldots, x_n) \in (\Sigma \cup \{\varepsilon\})^n$ (resp. $(y_1, \ldots, y_n) \in (\Sigma \cup \{\varepsilon\})^n$). The communication may depend only on the input of the bit-sending party and the bits already exchanged. The minimal number of communication bits needed to decide membership is called *deterministic communication complexity* and is denoted as $cc(L)$. By [10], a regular language L has constant communication complexity iff $M(L)$ is commutative.

The definition of an NFA $N = (Q, \Sigma, \delta, s, F)$ is similar to the definition of a DFA with the notable exception that $\delta(q, w)$ is not an element but a subset of Q, i.e., an element of the powerset 2^Q. Again the mapping δ can be extended by

[1] The language $L(\beta)$ induced by a regular expression is defined in the obvious manner.

morphism to a mapping $\delta : Q \times \Sigma^* \to 2^Q$ or even to a mapping $\delta^* : 2^Q \times \Sigma^* \to 2^Q$ by setting

$$\delta^*(R, w) = \cup_{z \in R} \delta^*(z, w) \ . \quad (1)$$

3 Testing Green's Relations

In this section, we prove the PSPACE-hardness of testing Green's relations (see Theorem 1,(i)). To this end, we design 2 logspace-reductions that start from the problem FAI. Recall that an instance of FAI is given by DFA $A_1, \ldots, A_k$ with a common input alphabet $\Sigma = \{a_1, \ldots, a_l\}$ and unique final states.
$\mathcal{L}$-,$\mathcal{H}$-TEST: The instance of FAI is transformed to the mappings

$$f_{a_1}, \ldots, f_{a_l}, h_0, g_+, h_+ : Z \uplus \{\sigma_0, \sigma_1, \sigma_2, \tau_0, \tau_1\} \to Z \uplus \{\sigma_0, \sigma_1, \sigma_2, \tau_0, \tau_1\} \ , \quad (2)$$

which we view as state transformations. Here, $Z = \uplus_{j=1}^k Z(A_j)$ denotes the disjoint union of the state sets of the DFAs, and $\sigma_0, \sigma_1, \sigma_2, \tau_0, \tau_1$ are five additional *special states*. In the sequel, the notion *state diagram* refers to the total diagram that is formed by the disjoint union of the state diagrams for all DFAs whereas the diagram for a particular DFA A_i is called *sub-diagram*. On the *ordinary states* (as opposed to the special states), the state transformations act as follows:

$$\forall a \in \Sigma : f_a(z) = z \cdot a \ \text{ and } \ h_0(z) = z_0, g_+(z) = h_+(z) = z_+ \ .$$

Here, z_0 denotes the unique initial state in the sub-diagram containing z. Likewise, z_+ denotes the unique accepting state in this sub-diagram. The special states $\sigma_0, \sigma_1, \sigma_2$ are transformed as follows:

$$\sigma_0 \stackrel{g_+,h_+}{\longmapsto} \sigma_1 \stackrel{g_+,h_+}{\longmapsto} \sigma_2 \quad (3)$$

Moreover, σ_0, σ_1 are fix-points for $f_{a_1}, \ldots, f_{a_l}, h_0$, and σ_2 is a fix-point for every transformation. The analogous interpretation applies to

$$\tau_0 \stackrel{h_0,h_+}{\longmapsto} \tau_1, \tau_1 \stackrel{h_0,h_+}{\longmapsto} \tau_0 \quad (4)$$

Concerning the $\mathcal{L}$-TEST (resp. the $\mathcal{H}$-TEST), we ask whether $g_+ \mathcal{L} h_+$ (resp. $g_+ \mathcal{H} h_+$) w.r.t. the monoid generated by the mappings in (2). We claim that the following equivalences are valid:

$$\bigcap_{j=1}^{k} L(A_j) \neq \emptyset \Leftrightarrow g_+ \mathcal{H} h_+ \Leftrightarrow g_+ \mathcal{L} h_+ \ . \quad (5)$$

To prove this claim, we first suppose $\bigcap_{j=1}^k L(A_j) \neq \emptyset$. Now, pick a word w from $\bigcap_{j=1}^k L(A_j)$, and observe that the following holds:

$$h_+ = f_w \circ h_0 \circ g_+, \qquad g_+ = f_w \circ h_0 \circ h_+ \quad (6)$$

$$h_+ = g_+ \circ h_0, \qquad g_+ = h_+ \circ h_0 \quad (7)$$

Thus, we have $g_+ \mathcal{L} h_+$ by (6) and $g_+ \mathcal{R} h_+$ by (7). Hence, $g_+ \mathcal{H} h_+$, as required. The implication from $g_+ \mathcal{H} h_+$ to $g_+ \mathcal{L} h_+$ holds for trivial reasons. Now, suppose $g_+ \mathcal{L} h_+$. Certainly, this implies that h_+ can be written as $h_+ = P \circ g_+$ where P is a product of generators, i.e., a composition of functions from (2). Since h_+ and g_+ are the only generators that do not leave states of type σ fixed and act on them according to (3), it follows that P neither contains h_+ nor g_+. For ease of later reference, we call this kind of reasoning the *σ-argument*. Since h_+, h_0 are the only generators that do not leave states of type τ fixed and act on them according to (4), the product P must contain h_0 an odd number of times. Focusing on the leftmost occurrence of h_0, P can be written as $P = P' \circ h_0 \circ P''$ where P' does not contain h_0, and P'' contains h_0 an even number of times. It is easily verified that $h_0 = h_0 \circ P''$, so $P = P' \circ h_0$ and $h_+ = P' \circ h_0 \circ g_+$ where product P' contains exclusively generators from $\{f_{a_1}, \ldots, f_{a_l}\}$. Thus, there exists a word $w \in \Sigma^*$ such that $P' = f_w$ and $h_+ = f_w \circ h_0 \circ g_+$. Now, we are done with the proof of (5) since this implies $w \in \bigcap_{j=1}^{k} L(A_j)$. (5) directly implies the desired hardness result for the $\mathcal{L}$- and the $\mathcal{H}$-TEST, respectively.

$\mathcal{R}$-,$\mathcal{J}$-TEST: This time, we map $A_1, \ldots, A_k$ to the following list of generators:

$$f_{a_1}, \ldots, f_{a_l}, f_0, f, g, g_+ : Z \uplus Z' \uplus \{\sigma_0, \sigma_1, \sigma_2\} \rightarrow Z \uplus Z' \uplus \{\sigma_0, \sigma_1, \sigma_2\} \qquad (8)$$

Here, Z is chosen as in (2), $Z' = \{z' : z \in Z\}$ contains a marked state z' for every ordinary state z, and $\sigma_0, \sigma_1, \sigma_2$ are special states (put into place to apply the σ-argument). The marked states are fix-points for every mapping. Mappings g, g_+ act on states of type σ according to (3) but now with g in the role of h_+. The remaining mappings leave states of type σ fixed. For every $a \in \Sigma$, $f_a(z) = z \cdot a$ is defined as in the previous logspace-reduction. Mappings f_0, f, g, g_+ act on ordinary states (with the same notational conventions as before) as follows:

$$f_0(z) = z_0, f(z) = g(z) = z', g_+(z) = z'_+$$

Concerning the $\mathcal{R}$-TEST (resp. the $\mathcal{J}$-TEST), we ask whether $g \mathcal{R} g_+$ (resp. $g \mathcal{J} g_+$) w.r.t. the monoid generated by the mappings in (8). We claim that the following equivalences are valid:

$$\bigcap_{j=1}^{k} L(A_j) \neq \emptyset \Leftrightarrow g \mathcal{R} g_+ \Leftrightarrow g \mathcal{J} g_+ \qquad (9)$$

To prove this claim, we first suppose $\bigcap_{j=1}^{k} L(A_j) \neq \emptyset$. Now, pick a word w from $\bigcap_{j=1}^{k} L(A_j)$, and observe that the following holds:

$$g = g_+ \circ f \ , \ g_+ = g \circ f_w \circ f_0$$

Thus, $g \mathcal{R} g_+$, as required. The implication from $g \mathcal{R} g_+$ to $g \mathcal{J} g_+$ holds for trivial reasons. Now, suppose $g \mathcal{J} g_+$. Certainly, this implies that g_+ can be written as $g_+ = P \circ g \circ Q = g \circ Q$ where P and Q are products of generators, respectively. The second equation is valid simply because g marks ordinary states and marked states are left fixed by all generators (so that P is redundant). It follows from the σ-argument that neither g nor g_+ can occur in Q (or P). We may furthermore assume that f does not occur in Q because a decomposition of $g \circ Q$ containing f could be simplified according to $g \circ Q = g \circ Q' \circ f \circ Q'' = g \circ Q''$. The last equation holds because f (like g) marks ordinary states which are then kept fixed by all generators. We may conclude that $g_+ = g \circ Q$ for some product of Q that does not contain any factor from $\{g, g_+, f\}$. Because of the simplification $Q' \circ f_0 \circ Q'' = Q' \circ f_0$, we may furthermore assume that either Q does not contain f_0, or it contains f_0 as the rightmost factor only. Thus, there exists some word $w \in \Sigma^*$ such that either $g_+ = g \circ Q = g \circ f_w$ or $g_+ = g \circ Q = g \circ f_w \circ f_0$. In both cases, this implies that $w \in \bigcap_{j=1}^{k} L(A_j)$ so that the proof of (9) is now complete. (9) directly implies the desired hardness result for the $\mathcal{R}$- and the $\mathcal{J}$-TEST, respectively. □

4 Finite Monoids: Testing for a Non-Abelian Subgroup

Recall from Section 1 that **A** denotes the variety of finite monoids with only trivial subgroups (the so-called aperiodic monoids). Let DFA-**A** be defined in analogy to the problem DFA-$\overline{\mathbf{Ab}}$ from Theorem 1. In [2], Cho and Huynh show the PSPACE-hardness of DFA-**A** by means of a generic reduction that proceeds in two stages with the first one ending at a special version of FAI. We will briefly describe this reduction and, thereafter, we will modify it so as to obtain a generic reduction to the problem DFA-$\overline{\mathbf{Ab}}$.

Let $\mathcal{M}$ be an arbitrary but fixed polynomially space-bounded DTM with input word x. In a first stage, Cho and Huynh efficiently transform $(\mathcal{M}, x)$ into a collection of prime p many minimum DFAs $A_1, \ldots, A_p$ with aperiodic syntactic monoids $M(A_i)$, initial states s_i, unique accepting states f_i, and unique (non-accepting) dead states such that $L(A_1) \cap \ldots \cap L(A_p)$ coincides with the strings that describe an accepting computation of $\mathcal{M}$ on x. Consequently, $L(A_1) \cap \ldots \cap L(A_p)$ is either empty or the singleton set that contains the (representation of the) unique accepting computation of $\mathcal{M}$ on x. In a second stage, Cho and Huynh connect $A_1, \ldots, A_p$ in a cyclic fashion by using a new symbol # that causes a state-transition from the accepting state f_i of A_i to the initial state s_{i+1} of A_{i+1} (or, if $i = p$, from f_p to s_1). This construction of a single DFA A (with $A_1, \ldots, A_p$ as sub-automata) is completed by amalgamating the p dead states, one for every sub-automaton, to a single dead state, and by declaring s_1 as the only initial state and f_1 as the only accepting state. (All #-transitions different from the just described ones end up in the dead state.) By construction, A is a minimum DFA. Moreover, $M(A)$ is not aperiodic iff $\mathcal{M}$ accepts x. The latter result relies on the following general observation:

Lemma 1 ([2]). *Let B be a minimum DFA: $M(B)$ is not aperiodic iff there is a state q and an input word u such that u* defines a non-trivial cycle starting at *q, i.e., $q \cdot u \neq q$ and, for some positive integer r, $q \cdot u^r = q$.*

For ease of later reference, we insert the following notation here:

$$r(q,u) := \min\left(\{r \in \mathbb{Z} : r \geq 1, q \cdot u^r = q\}\right)$$

with the convention that $\min(\emptyset) = \infty$.

Our modification of the reduction by Cho and Huynh is based on the following general observation:

Lemma 2. *Let B be a minimum DFA with state set Q and alphabet Σ: If $M(B)$ contains a non-Abelian subgroup G, then there exists a state q and a word u with $r(q,u) \geq 3$.*

Proof. Since every subgroup whose elements are of order at most 2 is Abelian, G contains an element $u \in \Sigma^+$ (identified with the element in $M(B)$ that it represents) of order r at least 3. Because u^r fixes the states from $Q' := Q \cdot u$, for every $q' \in Q'$, u defines a cycle starting at q'. Therefore, we obviously get $r = \mathrm{lcm}\{r(q',u) : q' \in Q'\}$. Because of $r \geq 3$, this directly implies the claim. □

We modify the first stage of the reduction by Cho and Huynh by introducing 2 new symbols $\vdash, \dashv$ (so-called *endmarkers*). Moreover, each sub-automaton A_i gets s'_i as its new initial state and f'_i as its new unique accepting state. We set $s'_i \cdot \vdash = s_i$ and $f_i \cdot \dashv = f'_i$. All other transitions involving s'_i, f'_i or $\vdash, \dashv$ end into the dead state of A_i. It is obvious that A_i still satisfies the conditions that are valid for the construction by Cho and Huynh: it has a unique accepting state and a unique (non-accepting) dead state; it is a minimum DFA whose syntactic monoid, $M(A_i)$, is aperiodic so that, within a single sub-automaton A_i, a word can define a trivial cycle only. In an intermediate step, we perform a duplication and obtain $2p$ sub-automata, say $A'_1, A'_2, \ldots, A'_{2p-1}, A'_{2p}$ such that A'_{2i-1} and A'_{2i} are state-disjoint duplicates of A_i.

In stage 2, we build a DFA A' by concatenating the sub-automata $A'_1, A'_2, \ldots, A'_{2p-1}, A'_{2p}$ in a cyclic fashion in analogy to the original construction (using symbol #) but now with s'_i, f'_i in the role of s_i, f_i. Again in analogy, we amalgamate the $2p$ dead states to a single dead state denoted REJ, and we declare s'_1 as the initial state and f'_{2p} as the unique accepting state of A'. The most significant change to the original construction is the introduction of a new symbol $\underline{\text{swap}}$ that, as indicated by its naming, causes transitions from s'_{2i-1} to s'_{2i} and vice versa, and transforms any other state into the unique dead state. The following result is obvious:

Lemma 3. *A' is a minimum DFA.*

The following two results establish the hardness result from Theorem 1,(ii).

Lemma 4. *If $\mathcal{M}$ accepts x, then $M(A')$ contains a non-Abelian subgroup.*

Proof. Let α denote the string that describes the accepting computation of $\mathcal{M}$ on x. Then, for every $i = 1, \ldots, 2p-1$ and for every state $q \notin \{s'_1, \ldots, s'_{2p}\}$,

$$s'_i \cdot \vdash \alpha \dashv \# = s'_{i+1}, s'_{2p} \cdot \vdash \alpha \dashv \# = s'_1, q \cdot \vdash \alpha \dashv \# = \mathrm{REJ}\ .$$

Thus, string $\vdash \alpha \dashv \#$ represents the cyclic permutation $\langle s'_1, s'_2, \ldots, s'_{2p-1}, s'_{2p}\rangle$ in $M(A')$. A similar argument shows that the letter $\underline{\text{swap}}$ represents the permutation $\langle s'_1, s'_2\rangle \ldots \langle s'_{2p-1}, s'_{2p}\rangle$ in $M(A')$. The strings $\vdash \alpha \dashv \#$ and $\underline{\text{swap}}$ generate a non-Abelian subgroup of $M(A')$. □

Lemma 5. *If $M(A')$ contains a non-Abelian subgroup, then $\mathcal{M}$ accepts x.*

Proof. According to Lemma 2, there exists a state q and a word u such that u defines a cycle C starting at q and $r := r(q, u) \geq 3$. Clearly, q must be different from the dead state. Let $S := \{s'_1, \ldots, s'_{2p}\}$. Let $C(q, u)$ be the set of states occurring in the computation that starts (and ends) at q and processes u^r letter by letter. $C(q, u) \cap S$ cannot be empty because, otherwise, the cycle C would be contained in a single sub-automaton A'_i which, however, is impossible because A'_i is aperiodic. By reasons of symmetry, we may assume that $s'_1 \in C(q, u)$. After applying an appropriate cyclic permutation to the letters of u, we may also assume that u defines a cycle C starting (and ending) at s'_1 and $r = r(s'_1, u) \geq 3$ (the same r as before). Since $C(q, u)$ does not contain the dead state, u must decompose into segments of two types:

Type 1: segments of the form $\vdash \alpha \dashv \#$ with no symbol from $\{\underline{\text{swap}}, \vdash, \dashv, \#\}$ between the endmarkers

Type 2: segments consisting of the single letter $\underline{\text{swap}}$

Since $r \geq 3$, there must be at least one segment of type 1. Applying again the argument with the cyclic permutation, we may assume that the first segment in u, denoted $\bar{u}_1$ in what follows, is of type 1. Every segment of type 1 transforms s'_i into s'_{i+1}.[2] Every segment of type 2 transforms s'_{2i-1} into s'_{2i} and vice versa. Now, consider the computation path, say P, that starts at s'_1 and processes u letter by letter. Let k be the number of segments of type 1 in u, let k' be the number of occurrences of $\underline{\text{swap}}$ in u that hit a state $s'_i \in P$ for an odd index i, and finally let k'' be the number of occurrences of $\underline{\text{swap}}$ in u that hit a state $s'_i \in P$ for an even index i. Thus, $s'_{2i-1} \cdot u = s'_{2i-1+k+k'-k''}$ and $s'_{2i} \cdot u = s'_{2i+k-k'+k''}$. Let $d := k + k' - k''$.

Case 1: d is even.

Note that $d \not\equiv 0 \pmod{2p}$ (because, otherwise, $s'_1 \cdot u = s'_1$ - a contradiction to $r \geq 3$). Since the sequence $s'_1, s'_1 \cdot u, s'_1 \cdot u^2, \ldots$ exclusively runs through states of odd index from S and there are p (prime number) many of them, the sequence runs through all states of odd index from S. It follows that at some point every sub-automaton A'_{2i-1} will process the first segment $\bar{u}_1 = \vdash \alpha \dashv \#$ of u (which is of type 1) and so it will reach state f'_{2i-1}. We conclude that $L(A'_1) \cap L(A'_3) \cap \ldots \cap L(A'_{2p-1})$ is not empty (as witnessed by $\bar{u}_1$). Thus, α represents an accepting computation of $\mathcal{M}$ on input x.

Case 2: d is odd.

Note that, for every $i = 1, \ldots, 2p$, $s'_i \cdot u^2 = s'_{i+2k}$. Thus, $2k \not\equiv 0 \pmod{2p}$ (because, otherwise, $s'_1 \cdot u^2 = s'_1$ - a contradiction to $r \geq 3$). Now, the sequence

[2] Throughout this proof, we identify an index of the form $2pm + i, 1 \leq i \leq 2p$, with the index i. For example, s'_{2p+1} is identified with s'_1.

$s'_1, s'_1 \cdot u^2, s'_1 \cdot u^4, \ldots$ exclusively runs through states of odd index from S, and we may proceed as in Case 1. □

5 Complexity of Communication Complexity

5.1 Space-efficient Algorithms for Syntactic Monoids

Let $N = (Z, \Sigma, \delta, z_1, F)$ be an NFA with states $Z = \{z_1, \ldots, z_n\}$, alphabet Σ, initial state z_1, final states $F \subseteq Z$, transition function $\delta : Z \times \Sigma \to 2^Z$, and let $\delta^* : 2^Z \times \Sigma^* \to 2^Z$ be the extension of δ as defined in Section 2. Let $L = L(N)$ be the language recognized by N, and let $\mathcal{A}$ be the minimum DFA for L. It is well-known that the syntactic monoid $M := M(L)$ of L coincides with the transformation monoid of $\mathcal{A}$, and that $\mathcal{A}$ may have up to 2^n states. We aim at designing space-efficient algorithms that solve questions related to M. These algorithms will never store a complete description of $\mathcal{A}$ (not to speak of M). Instead, they will make use of the fact that *reachable sets* $R \subseteq Z$ represent states of $\mathcal{A}$ in the following sense:

- R is called *reachable (by w)* if there exists $w \in \Sigma^*$ such that $R = \delta^*(z_1, w)$.
- Two sets Q, R are called *equivalent*, denoted as $Q \equiv R$, if, for all $w \in \Sigma^*$, $\delta^*(Q, w) \cap F \neq \emptyset \Leftrightarrow \delta^*(R, w) \cap F \neq \emptyset$, which is an equivalence relation.
- For reachable $R \subseteq Z$, $[R]$ denotes the class of reachable sets Q such that $Q \equiv R$.

The following should be clear from the power-set construction combined with DFA-minimization (e.g. [4]): the states of $\mathcal{A}$ are in bijection with the equivalence classes $[R]$ induced by reachable sets. Moreover, the transition function $\delta_{\mathcal{A}}$ of $\mathcal{A}$ satisfies $\delta_{\mathcal{A}}([R], a) = [\delta^*(R, a)]$ for every $a \in \Sigma$ (and this is well-defined). The extension $\delta^*_{\mathcal{A}}$ is related to δ^* according to $\delta^*_{\mathcal{A}}([R], w) = [\delta^*(R, w)]$ for every $w \in \Sigma^*$.

We now move on and turn our attention to M. Since M coincides with the transformation monoid of $\mathcal{A}$, it precisely contains the mappings

$$T_w([R]) := \delta^*_{\mathcal{A}}([R], w) = [\delta^*(R, w)] \ , \text{ reachable } R \subseteq Z \tag{10}$$

for $w \in \Sigma^*$. M is generated by $\{T_a | a \in \Sigma\}$. Because of (1) and (10), every transformation T_w is already determined by $A^w := (A^w_1, \ldots, A^w_n)$ where

$$A^w_i := \delta^*(z_i, w) \subseteq Z, i = 1, \ldots, n \ . \tag{11}$$

In particular, the following holds for $A := A^w$, $A_i := A^w_i$, and $T_A := T_w$:

$$T_A([R]) = \left[\bigcup_{i: z_i \in R} \delta^*(z_i, w) \right] = \left[\bigcup_{i: z_i \in R} A_i \right] \ , \text{ reachable } R \subseteq Z \tag{12}$$

Thus, given a reachable R and $A = A^w$, one can time-efficiently calculate a representant of $T_A([R]) = T_w([R])$ without knowing w. In order to emphasize this, we prefer the notation T_A to T_w in what follows. We call A a *transformation-vector* for T_A.

The next lemma presents a list of problems some of which can be solved in polynomial time (p.t.), and all of which can be solved in polynomial space (p.s.):

Lemma 6. *The NFA N is part of the input of all problems in the following list.*

1. *Given $R \subseteq Z$, the reachability of R can be decided in p.s..*
2. *Given a reachable set $R \subseteq Z$ and a transformation-vector A (for an unknown T_w), a representant of $T_A([R]) = T_w([R])$ can be computed in p.t..*
3. *Given $a \in \Sigma$, a transformation-vector for T_a can be computed in p.t..*
4. *Given transformation-vectors A (for an unknown T_w), B (for an unknown $T_{w'}$), a transformation-vector for $T_B \circ T_A$, denoted as $B \circ A$, can be computed in p.t..*
5. *Given a transformation-vector A, its* validity *(i.e., does there exist $w \in \Sigma^*$ such that $A = A^w$) can be decided in p.s..*
6. *Given $Q, R \subseteq Z$, it can be decided in p.s. whether $Q \equiv R$.*
7. *Given valid transformation-vectors A, B, their* equivalence *(i.e., $T_A = T_B$) can be decided in p.s..*
8. *It can be decided in p.s. whether M is commutative.*
9. *Given a valid transformation-vector A, it can be decided in p.s. whether T_A is idempotent.*
10. *Given valid transformation-vectors A, B, it can be decided in p.s. whether $T_A \in MT_BM$ (similarly for $T_A \in MT_B$, or for $T_A \in T_BM$).*

Proof. By Savitch's Theorem, membership in PSPACE can be proved by means of non-deterministic procedure. We shall often make use of this option.

1. Initialize Q to $\{z_1\}$. While $Q \neq R$ do
   (a) Guess a letter $a \in \Sigma$.
   (b) Replace Q by $\delta^*(Q, a)$.
2. Apply formula (12).
3. Apply formula (11) for $i = 1, \ldots, n$ and $w = a$ (so that δ^* collapses to δ).
4. For $i = 1, \ldots, n$, apply the formula $C_i = \cup_{j:z_j \in A_i} B_j$. Then $T_C = T_{ww'}$.
5. Initialize B to $(\{z_1\}, \ldots, \{z_n\})$ which is a transformation-vector for T_ε. While $B \neq A$ do
   (a) Guess a letter $c \in \Sigma$. Compute the (canonical) transformation-vector C for T_c.
   (b) Replace B by the (canonical) transformation-vector for $T_C \circ T_B$.
6. It suffices to present a non-deterministic procedure that recognizes inequivalence: While $Q \cap F \neq \emptyset \Leftrightarrow R \cap F \neq \emptyset$ do
   (a) Guess a letter $a \in \Sigma$.
   (b) Replace Q by $\delta^*(Q, a)$ and R by $\delta^*(R, a)$, respectively.
7. It suffices to present a non-deterministic procedure that recognizes inequivalence:
   (a) Guess $Q \subseteq Z$ and verify that Q is reachable.
   (b) Compute a representant R of $T_A([Q])$.

(c) Compute a representant S of $T_B([Q])$.
(d) Verify that $R \not\equiv S$.

8. The syntactic monoid is commutative iff its generators commute. It suffices to present a non-deterministic procedure that recognizes non-commutativity:
   (a) Guess two letters $a, b \in \Sigma$.
   (b) Compute transformation-vectors A for T_{ab} and B for T_{ba}.
   (c) Given these transformation-vectors, verify their inequivalence.
9. Compute $A \circ A$ and decide whether A and $A \circ A$ are equivalent.
10. Guess two transformations-vectors C, D and verify their validity. Compute the transformation-vector $D \circ B \circ C$ and accept iff it is equivalent to A.

□

Note that the 10th assertion of Lemma 6 is basically saying that Green's relations w.r.t. the syntactic monoid of $L(N)$ can be decided in polynomial space.

Corollary 1. *Given NFA N, the deterministic communication complexity $cc(L)$ of the language $L = L(N)$ can be determined in polynomial space. Moreover, the membership of the syntactic monoid $M(L)$ in $\overline{\mathbf{Ab}}$ can be decided in polynomial space.*

Proof. The following facts are known from [10]: $cc(L)$ is constant iff $M(L)$ is commutative. If $cc(L)$ is not constant, it is either logarithmic or linear. The linear case occurs iff there exist $a, b, c, d, e \in M(L)$ such that (i) $a\mathcal{H}b\mathcal{H}c, a^2 = a, bc \neq cb$ or (ii) $a\mathcal{J}b, a^2 = a, b^2 = b, (ab)^2 \neq ab \vee a \not\mathcal{J} ab$. Condition (i) is equivalent to the condition $M(L) \notin \overline{\mathbf{Ab}}$. The assertion of the corollary is now immediate from Lemma 6. □

5.2 Hardness Result for Regular Expressions

Definition 1. *Let L be a formal language over an alphabet Σ. Let $w = a_1 \ldots a_m$ be an arbitrary Σ-word of length m. We say that L is* invariant under permutation *if*

$$w = a_1 \ldots a_m \in L \Longrightarrow \pi(w) := a_{\pi(1)} \ldots a_{\pi(m)} \in L$$

holds for every permutation π of $1, \ldots, m$.

The following result is folklore:

Lemma 7. *Let L be a formal language over an alphabet Σ. Then $M(L)$ is commutative iff L is invariant under permutation.*

We are now ready for the main result in this section:

Theorem 2. *For every $f(n) \in \{1, \log n, n\}$, the following problem is PSPACE-hard: given a regular expression β over an alphabet Σ, decide whether $L(\beta)$ has deterministic communication complexity $\Theta(f(n))$.*

Proof. We know from [10] (see Section 2) that a regular language has constant deterministic communication complexity iff its syntactic monoid is commutative. In [1], the authors show (by means of a generic reduction) that the problem of deciding whether $L(\beta) \neq \Sigma^*$ is PSPACE-hard, even if either $L(\beta) = \Sigma^*$ or $L(\beta) = \Sigma^* \setminus \{w\}$ for some word $w \in \Sigma^*$ that contains at least two distinct letters. Clearly, Σ^* is invariant under permutation whereas $\Sigma^* \setminus \{w\}$ is not. According to Lemma 7, the syntactic monoid of Σ^* is commutative whereas the syntactic monoid of $\Sigma^* \setminus \{w\}$ is not. It readily follows that deciding "$cc(L(\beta)) = O(1)$?" is PSPACE-hard. It is easy to show that the deterministic communication complexity of $\Sigma^* \setminus \{w\}$ is $\Theta(\log n)$. Thus, deciding "$cc(L(\beta)) = \Theta(\log n)$?" is PSPACE-hard, too. It is easy to modify the proof of [1] so as to obtain the PSPACE-hardness of the problem "$L(\beta) \neq \Sigma^*$?" even when either $L(\beta) = \Sigma^*$ or $L(\beta) = \Sigma^* \setminus w^*$ for some word w that contains at least two distinct letters. It is easy to show that the deterministic communication complexity of $\Sigma^* \setminus w^*$ is $\Theta(n)$. Thus, deciding "$cc(L(\beta)) = \Theta(n)$?" is PSPACE-hard, too. □

As is well-known, a regular expression can be transformed into an equivalent NFA in polynomial time. Thus, the decision problems from Theorem 2 remain PSPACE-hard when the language L is given by an NFA.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
2. Cho, S., Huynh, D.T.: Finite-Automaton Aperiodicity is PSPACE-Complete. Theor. Comput. Sci. 88(1), 99–116 (1991)
3. Colcombet, T.: Green's Relations and Their Use in Automata Theory. In: Dediu, A.H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 1–21. Springer, Heidelberg (2011)
4. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading (1979)
5. Kozen, D.: Lower Bounds for Natural Proof Systems. In: 18th Annual Symposium on Foundations of Computer Science, pp. 254–266. IEEE Computer Society, Washington (1977)
6. Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press, Cambridge (1997)
7. Pin, J.E.: Varieties of Formal Languages. Plenum Publishing, New York (1986)
8. Schützenberger, M.P.: On Finite Monoids Having Only Trivial Subgroups. Information and Control 8(2), 190–194 (1965)
9. Stern, J.: Complexity of Some Problems from the Theory of Automata. Information and Control 66(3), 163–176 (1985)
10. Tesson, P., Thérien, D.: Complete Classifications for the Communication Complexity of Regular Languages. Theory Comput. Syst. 38(2), 135–159 (2005)
11. Yao, A.C.: Some Complexity Questions Related to Distributive Computing. In: 11th Annual Symposium on Theory of Computing, pp. 209–213. ACM, New York (1979)