On the Complexity of Decidable Cases of The Commutation Problem of Languages

Juhani Karhumäki * **Wojciech Plandowski** † **Wojciech Rytter** ‡

Abstract

We investigate the complexity of basic decidable cases of the commutation problem for languages: testing the equality $XY = YX$ for two languages X and Y. We show that it varies from co-$NEXPTIME$ complete through $PSPACE$ complete and co-NP complete to deterministic polynomial time, when Y is an explicitly given finite language and X is given by a CF grammar generating a finite language, a nondeterministic finite automaton (or a regular expression), an acyclic nondeterministic finite automaton or an explicitly given finite language, respectively. Interestingly in most cases the complexity status does not change if instead of explicitly given finite Y we consider general Y of the same type as X. For deterministic finite automata the problem remains open, due to the asymmetry of the catenation.

1 Introduction

Research on word equations during the past few decades has revealed several amazing results, most notably the Makanin's algorithm and its extensions [16, 4, 17]. The situation changes completely when equations on languages (or even on finite languages) are considered. Very little, or in fact almost nothing, is known about their solutions [11]. The exception being the case when such restricted equations with two operations – union and concatenation – are considered where certain fixed points results become applicable. An example is the method to compute the rational expression for a given finite automaton, see [14] for a general survey. Such a theory, however, does not have a counterpart in word equations.

The equation corresponding to the *commutation problem* is: $XY = YX$. Even this deceptively simple equation proposes several natural and combinatorially interesting problems.

*Department of Mathematics, Turku University, Finland and Turku Centre for Computer Science, Turku, 20014 Finland. Supported by Academy of Finland under grant 44087. Email:`karhumak@cs.utu.fi`.

†Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02–097 Warszawa, Poland. Supported in part by KBN grant 8T11C03915 and in part by Academy of Finland under grant 44087. Email:`wojtekpl@mimuw.edu.pl`.

‡Instytut Informatyki, Uniwersytet Warszawski Banacha 2, 02–097 Warszawa, Poland and NJIT-CCS, Computer Science Department, GITC 4400, University Heights, Newark, NJ 07102, USA. Supported in part by KBN grant 8T11C03915. Email:`rytter@mimuw.edu.pl`

Recently there was a renewed interest in commutation problem due to the partial solution to the problem posed more than 30 years ago by Conway [2]:

> Assume X is the maximal solution to $XY = YX$ for a given finite set Y.
> Is the language X regular?

An affirmative answer for a special case of prefix sets was shown in [18]. Then in [1] a simple solution was shown when $Y \subseteq \Sigma^+$ is a binary set, and recently this was extended to all three element sets Y, see [13] and [12]. Interestingly, in all these cases the above maximal set is Y^+, and also complete characterization for sets commuting with Y were obtained. On the other hand, even for sets of cardinality four the Conway's Problem is unanswered. Something about the intriguing nature of the commutation was revealed in [7], where it was shown that it is undecidable whether a given two element set and a given context-free language commute. This is the reason we restrict here only CF grammars generating only finite languages.

It seems that further studies on the commutation with finite sets is needed. In this paper we study the following problem:

Instance: *Given descriptions of two languages X and Y,*

Question: *What is the complexity of testing the equality $X \cdot Y = Y \cdot X$?*

We can change the way we formulate a problem instance by changing the type of description: deterministic and nondeterministic finite automata (*dfa*'s and *nfa*'s, in short) with and without cycles, context-free grammars (*cfg*'s, in short) and regular expressions. Even the exact complexity of the case of an explicitly given finite X is nontrivial. The set Y is in most cases an explicitly given finite language, unless it is specified otherwise. Then the size of Y, in symbols $||Y||$, is to be understood as the total length of its words. Sometimes a secondary size is considered: the cardinality of Y, in symbols $|Y|$. By sizes of regular expressions, *nfa*'s and *cfg*'s we mean the numbers of symbols in their descriptions.

We assume throughout the paper that Σ is an alphabet containing at least two symbols. Our next proposition is a special case of a theorem in [18].

Proposition 1.1 *If $X \cap \Sigma^m \neq \emptyset$, $\epsilon \notin X$ and $X\Sigma = \Sigma X$, then $\Sigma^m \subseteq X$.*

Proof: Let $Z = \Sigma^m \cap X$. We show by induction the following statement for each $k \leq m$:

$$\forall (y \in \Sigma^k)\ \exists (x \in Z)\ :\ y \text{ is a prefix of } x.$$

This is clearly true for $k = 1$. Then, since $\Sigma Z \subseteq Z\Sigma$, we have that all words of length $k+1$, as they are prefixes of ΣZ by inductive assumption, are also prefixes of words in Z. $\Box$

We denote

$$L^{\leq n} \ = \ L^1 \cup L^2 \cup L^3 \ldots \cup L^n$$

and conclude the following useful facts.

Lemma 1.2 **(i)** *If $L = \Sigma^*$ or $L = \Sigma^* - \{u\}$ for some word u, then $L\Sigma = \Sigma L$ if and only if $L = \Sigma^*$.*

(ii) *If for all $k = 1, \ldots, m$, $L \cap \Sigma^k \neq \emptyset$, $\epsilon \notin L$ and $L \subseteq \Sigma^{\leq n}$, then $L\Sigma = \Sigma L \Leftrightarrow L = \Sigma^{\leq n}$.*

(iii) *If $L \subseteq \Sigma^m$ and $L \neq \emptyset$, then $L\Sigma = \Sigma L \Leftrightarrow L = \Sigma^m$.*

2 Regular expressions and nondeterministic automata

Regular languages can be represented by regular expressions or *nfa*'s. Unfortunately, the representations are not polynomially equivalent. For each regular expression R there is polynomial size *nfa* (with respect to the size of R) that accepts the language represented by R, and moreover such an *nfa* can be constructed in polynomial time [9, 8, 10]. The opposite is not true: there is an *nfa* accepting a regular language L such that the smallest regular expression for L is of exponential size with respect to the size of the *nfa* [5]. Nevertheless, independently of the representation we prove that the commutation problem for a regular language and a finite language is $PSPACE$ complete. As a consequence we have that the commutation problem for two regular languages is also $PSPACE$ complete.

Theorem 2.1 *Let X be a regular language given by a regular expression. Then the commutation problem for X and $Y = \Sigma$ is $PSPACE$ hard.*

Proof: Let M be any deterministic Turing machine working in polynomial space. We assume that M stops on every input. Let $p_M(n)$ be the size of the maximal space used by M during the computation for a word of size n. Clearly, $p_M(n) \leq cn^k$ for some constants c and k. Let w be any word over $\{a, b\}$. A history of a computation of M on the word w is a word $history(w)$ which is of the form $\#w_0\#w_1\#\ldots\#w_m\#$, where $|w_i| = c|w|^k + 1$ and w_i is a configuration of M at step i of the computation of M on the word w. Then $w_0 = q_0wB^j$, where q_0 is the starting state of M and B is a blank symbol of the tape of M and $j = cn^k - 1$. Moreover, we assume that after configuration w_m the machine M stops.

Let $\Sigma = \{a, b, \#, B\} \cup Q$, where Q is the set of states of M. Now we construct a regular expression R of polynomial size on $|w|$ such that, $R = \Sigma^*$ if w is not accepted by M, and $R = \Sigma^* - \{history(w)\}$ if w is accepted by M. Then the result is a consequence of Lemma 1.2.

The expression R is the union of four expressions: $R = R_1 \cup R_2 \cup R_3 \cup R_4$. R_1 describes words which are not of the form $\#u_0\#u_1\#\ldots\#u_l\#$, where u_i's are of length $c|w|^k + 1$, and they contain exactly one state of M. R_2 describes words which are neither of the form $\Sigma^*qxu\#u'x'q'\Sigma^*$, where $q,\ q' \in Q$, $x, x' \in \{a, b\}$, $(q, x, q', x', 1)$ is a transition of M, and $|xu\#u'| = cn^k - 1$, nor of the form $\Sigma^*qxu\#u'q'x'\Sigma^*$, where $q,\ q' \in Q$, $x, x' \in \{a, b\}$, $(q, x, q', x', -1)$ is a transition of M, and $|xu\#u'| = cn^k - 1$. R_3 describes the words which are not of the form $\#uau'\#wa$ where $|u| = |w|$ and the symbol a is not immediately preceded by a state q in uau'. R_4 describes the words which do not contain w_0 as a prefix or do not contain an accepting configuration as a suffix. Intuitively expression R_1 includes all words which are not of a correct form to be computations, R_2 includes those words where a computation step is not correct, R_3 includes those words where other than the scanned letter is changed, and finally R_4 those words which starts or ends wrongly. Clearly, as the disjunction all words except the potentially halting computation on w are included in R. We leave it to the reader to check that all R_i's indeed can be constructed and that they are of the required sizes. □

Corollary 2.2 *Let X and Y be regular languages given by* nfa*s or regular expressions. Then the commutation problem for X and Y is $PSPACE$ complete.*

Proof: By Theorem 2.1 and the above discussion it is enough to prove that the problem is in $PSPACE$ when X and Y are given by two *nfa*s. Since X and Y are regular, the languages XY and YX are regular, too. Moreover their representations as two *nfa*s can be computed in polynomial time. Now it is enough to solve the equivalence problem for two *nfa*s in $PSPACE$. The latter problem is in $PSPACE$, in fact $PSPACE$ complete, see [6]. □

As a straightforward consequence of Theorem 2.1 and Corollary 2.2 we have:

Theorem 2.3 *Let X be a regular language given by an* nfa *or a regular expression, and let Y be a finite language given by an acyclic* nfa*. Then the commutation problem for X and Y is $PSPACE$ complete.*

In what follows we replace general *nfa*'s by acyclic ones, and show that the complexity of our problem changes as well.

Theorem 2.4 *Let X be a finite language given by an acyclic* nfa*, and let $Y = \Sigma$. Then the commutation problem for X and Y is co-NP complete.*

Proof: The proof is analogous to that of Theorem 2.1. We want to prove that checking $XY \neq YX$ is NP-hard. We take a nondeterministic Turing machine M which works in polynomial time. We assume that M stops on every input and on every nondeterministic choice. For a given word w define a set $HISTORY(w)$ being the set of all possible histories of computations of M on the word w. All words in $HISTORY(w)$ are of polynomial size with respect to $|w|$. We choose c and k such that for every $w \in \Sigma^*$ and every $u \in HISTORY(w)$

$$|u| \leq c|w|^k.$$

We can also assume that in all histories of w the configurations are of a constant size, as was the case in the proof of Theorem 2.1. Further from technical reasons we define

$$HISTORY'(w) = \{uB^j : u \in HISTORY(w), j = c|w|^k - |u|\}.$$

All words in $HISTORY'(w)$ are of the same length $c|w|^k$ which depends only on $|w|$. Denote by *Accept*(w) the subset of $HISTORY'(w)$ containing all accepting histories for w. Given a word w denote $m = c|w|^k$. Now we construct a regular language R such that $R = \Sigma^m$ if $HISTORY(w)$ does not contain an accepting history for w, and $R = \Sigma^m - Accept(w)$ if $HISTORY(w)$ contains an accepting history for w. Now the result is a consequence of the considerations in Theorem 1.2.

The definition of R is basically that of the proof of Theorem 2.1: It consists of four expressions the three last ones being as there, and the R_1 describing those words which are not of the form $\#u_0\#u_1\#\ldots\#u_l\#B^j$ where u_i's are of suitable constant length containing just one occurrence of the state set of M and j is as specified above. $\square$

Corollary 2.5 *Let X be a finite language given by an acyclic* nfa *and Y be a finite language given explicitly or by an acyclic* nfa. *Then the commutation problem for X and Y is co-NP complete.*

Proof: We may assume that Y is given by an acyclic *nfa*. First we guess w in X and v in Y such that wv is not in YX or vw is not in XY. Words w and v are of polynomial size. Checking that a guess is correct can be done in polynomial time since the languages XY and YX are regular ones. $\square$

Observe here that if we replace acyclic *nfa*'s by a star-free regular expressions then the complexities of considered problems are the same. We return to the case of *dfa*'s in Section 4.

3 Context-free grammars

In this section we consider the commutation problem in the setting that languages are specified by context-free grammars, *cfg*'s for short, or pushdown automata, *pda*'s for short. Context-free grammars are much more powerful language generators than finite automata, as is examplified by the following result of [7].

Proposition 3.1 *Let X be a context-free language and Y a two element set. It is undecidable whether or not X and Y commute.*

Consequently, from the point of view of our goals we have to restrict to *cfg*'s (or *pda*'s) which generate only finite languages. Then, of course, the question of Proposition 3.1 becomes decidable.

In our later considerations we need some basic properties of *cfg*'s and *pda*'s. First of all the *size* of a *cfg* or a *pda* is defined as the number of symbols in its description. Note that this number is linearly related to the number of productions of a *cfg* or of the transitions of a *pda*. Also the following lemma follows from basic results of context-free languages.

Lemma 3.2 *For each* pda *$\mathcal{A}$ there exists a* cfg *which is equivalent to $\mathcal{A}$ and of polynomial size with respect to that of $\mathcal{A}$, and conversely.*

This result allows us to use either the grammar based or the automata based approach.

Now, assume that a *cfg* $\mathcal{G}$ generates a finite language; this is an easily decidable property. It can generate single exponentially long words with respect to its size. An example is the following grammar

$$\mathcal{G}: \begin{array}{ll} X_i \to X_{i+1}X_{i+1}, & \text{for } i = 0, \ldots, n-1, \\ X_n \to a \mid b. & \end{array}$$

The above shows also that a cfg of linear size can generate a finite language of double exponential cardinality.

Lemma 3.3 *Each finite language generated by a* cfg *is exponentially bounded with respect to the size of the grammar.*

The proof of Lemma 3.3 is straightforward. Another basic property of finite context-free languages is specified in the next lemma. Intuitively it shows that *exponential counting* is possible by a small *pda*.

Lemma 3.4 **(i)** *There is a* pda *with* $\mathcal{O}(n)$ *states accepting the language*

$$L_n = \{a^i \# b^i | 1 \leq i \leq 2^n\}.$$

(ii) *There is a polynomial size* pda *(in terms of n) accepting the language*

$$L_{c,n} = \{w \in (\Sigma \cup \#)^* || w| \leq 2^{cn}, w \textit{ contains a factor } y \in \#\Sigma^*\# \textit{ with } |y| \neq 2^n\},$$

where $\# \notin \Sigma$*, and c is a constant.*

Proof: What has to be done when accepting these languages is counting up to 2^n (or 2^{cn}). This can be done by using only $\mathcal{O}(n)$ states by remembering the current value in the stack as a binary number having the least significant digit on the top. Then to add 1 means that replace the first 0 by 1 and all 1's above it by 0. This is not difficult to do: the machine pops all the topmost 1's and counts by states how many have been popped, then replace the 0 by 1 and add so many 0's to the top as was counted in the states. Similarly, subtraction by 1 is easy.

The above clearly implies (i).

Part (ii) looks at first glance more difficult since one has to do the exponential adding two times and independently. However, this can be done as follows. The states of the *pda* have two components. The first components are used to count up to 2^n basically as above. It will repeatedly check that factors in between two consecutive marks are of the length 2^n. Hence, $\mathcal{O}(n)$ values of the first components are enough. Whenever, a factor of length 2^n is found the second counter, realized by the second component of the states, is incremented by one. This counter counts up to $2^{(c-1)n} - 1$, i.e. the size of the counter is $\mathcal{O}(n)$. Now the automaton rejects the input if it succeeds to check that the input consists of $2^{(c-1)n} - 1$ words of Σ^{2^n} each of those separated by the marker #. Otherwise the word is accepted. The details how this is achieved are left to the reader.

Now, the result follows from the identity

$$2^n(2^{(c-1)n} - 1 + 1) = 2^{cn}.$$

Note here that the term +1 comes from the number of markers #. □

Theorem 3.5 *Assume a* cfg *G generates a finite set L. The problem of testing* $L\Sigma = \Sigma L$ *is co-NEXPTIME-complete.*

Proof: We show that there is a polynomial size grammar generating all sequences of exponential size which are invalid histories of an accepting computation of a Turing machine working in nondeterministic exponential time. We use the ideas from Lemma 3.4 to construct a *pda* $\mathcal{A}$ accepting all words of size at most 2^{cn} except valid accepting histories. Assume the accepting history is a word of the form

$$\#w_1\#w_2\#w_3\ldots\#w_t, \tag{1}$$

where $t = 2^n$ and $|w_i| = 2^n$, $w_i \in \Sigma^*\mathcal{Q}\Sigma^*$ where $\mathcal{Q}$ is the set of states of the Turing machine, and $\# \notin \Sigma$.

By the definition of (1) we choose $c = 3$ in Lemma 3.4 (ii). Then we construct a *pda* $\mathcal{A}$ which accepts all words of length at most 2^{3n} which are not of the form (1). In addition we can require that $\mathcal{A}$ accepts all of the above words which does not contain exactly one state symbol of the Turing machine in each of the words w_i. All this leads to a polynomial size *pda.*

So far $\mathcal{A}$ has not excluded valid computations of the Turing machine. Consequently, we modify $\mathcal{A}$ such that it nondeterministically checks that, for some i, the words w_i and w_{i+1} *are not* consecutive configurations of the machine. And in this case the word is accepted. The checking is easy to do by a pushdown store. The problem is that it is already needed in counting the length of w_i and w_{i+1} in the proof of Lemma 3.4 (ii), and of course two stores are not allowed. This problem can be overcome as follows.

When for a nondeterministically chosen i the word w_i is read then the counting of the proof of Lemma 3.4 is not done for w_i (and w_{i+1}). Instead a position of q in w_i is stored in the stack and after the machines checks that the marker # is found within 2^n input symbols. After that the stack is modified to remember the position of q and the corresponding position in w_{i+1} is searched for. Then the *pda* can check whether the state symbols in w_i and w_{i+1} does not correspond to a transition of the Turing machine; if not the input word is accepted. In addition the *pda* continues by searching the next marker within 2^n input symbols.

In the above computation the lengths of w_i and w_{i+1} are not checked, but those of all other w_j's are checked. If they are not correct the input is accepted in another computation (if only the total length is at most 2^{cn}). So the problem is to accept those words which are of the form (1) and not valid computations. But this is exactly what is done by the computation described above.

It follows that the *pda* constructed accepts all words of length at most 2^{cn}, if there is no valid history of a computation in 2^n time of the Turing machine. If there is a valid computation then $\mathcal{A}$ accepts some word of length at most 2^n, and at the same time it does not accept some other word of the same length. We can conclude by Lemma 1.2. Indeed, due to Lemma 3.2 $\mathcal{A}$ can be transformed into a equivalent *cfg* of a polynomial size. □

Corollary 3.6 *Let a finite language L be given by a context-free grammar G and let Y be a finite language given explicitly or by an acyclic* nfa *or by a context-free grammar G'. Then the commutation problem for L and Y is in co-$NEXPTIME$-complete.*

Proof: It is enough to prove that the problem is in co-$NEXPTIME$, so it is enough to prove that the problem $LY \neq YL$ is in $NEXPTIME$. We may assume that Y is given by a *cfg*. First we guess words w in L and v in Y such that wv is not in YL or vw is not in

LY. By Lemma 3.3 the word wv is of length at most exponential with respect to the sizes of G and G'. Now YL is context-free so the checking whether wv is in YL can be done in exponential time. Similarly, we can check whether vw is in LY. $\square$

Similarly we obtain:

Corollary 3.7 *Let a finite language L be given by a context-free grammar G and let Y be a regular language given by an* nfa *or by a regular expression. Then the commutation problem for L and Y is in $EXPSPACE$.*

Proof of Corollary 3.7. It is enough to prove that there is a regular expression of size single exponential which describes the language L. First we remove from the grammar G useless nonterminals and then we transform it into a context-free grammar G' in Chomsky normal form. The size of G' is polynomial with respect to the size of G. Since G' describes a finite language, nonterminals of G' can be partially ordered by a relation $A > B$ iff $A \to BC$ or $A \to CB$ is a prodution in G'. We extend the relation $>$ into a linear order $>'$ in any way. Let $A_0 >' A_1 >' \ldots >' A_n$ be all nonterminals of G' ordered by $>'$. The regular expression for L can be constructed as follows. Let $A_0 \to B_1C_1$, $A_0 \to B_2C_2, \ldots, A_0 \to B_{k_0}C_{k_0}$ be all productions in G' in which left hand side is A_0. We call such productions productions *for* A_0. We start by regular expression $\mathcal{R}_0 = B_1C_1 \cup B_2C_2 \cup \ldots B_{k_0}C_{k_0}$. The expression contains $2k_0$ nonterminals. Let $A_1 \to D_1E_1$, $A_1 \to D_2E_2, \ldots, A_1 \to D_{k_1}E_{k_1}$ be productions of G' for A_1. Now in $\mathcal{R}_0$ we replace all occurrences of A_1 by the expression $(D_1E_1 \cup D_2E_2 \cup \ldots \cup D_{k_1}E_{k_1})$. In this way we obtain a regular expression $\mathcal{R}_1$ of size $O(2k_0)O(2k_1)$ with at most $2^2k_0k_1$ occurrences of nonteminals $\{A_2, A_3, \ldots, A_n\}$. Similarly having k_2 productions for A_2 we construct a regular expression $\mathcal{R}_2$ of size $O(2k_0)O(2k_1)O(2k_2)$ containing at most $2^3k_0k_1k_2$ occurrences of nonterminals $\{A_3, A_4, \ldots, A_n\}$. Suppose the number of productions for A_i is k_i. Then proceeding as previously the elimination of consecutive nonterminals we eventually get a regular expression $\mathcal{R}_n$ of size $O(2k_0)\ldots O(2k_n)$ without nonterminals. Hence, the size of $\mathcal{R}_n$ does not exceed $c^n(2k_0)\ldots(2k_n)$ for a suitable constant c. Suppose the size of G' is m. Then, clearly, $m = 2(k_0 + \ldots + k_n)$. Since arithmetic mean is at least as big as geometric mean we have

$$2k_0 \cdot 2k_1 \cdot \ldots \cdot 2k_n \leq \left(\frac{m}{n}\right)^n$$

for $1 \leq n \leq \frac{m}{2}$ which takes maximum for $n = \frac{m}{2}$. Hence, the size of $\mathcal{R}_n$ is $2^{O(m)}$ i.e. is single exponential function of m. This completes the proof. $\square$

4 Deterministic finite automata

For deterministic finite automata (*dfa*'s, in short) the complexity status of the commutation problem is open. This is due to the asymmetry of the complexity of concatenating a *dfa* language by a finite set from the left and from the right.

We have partial results, which suggest that the problem would be $PSPACE$-complete. Let us consider the problem of the complexity of a language of the form $L' = L(\mathcal{A}) \cdot Y$, where Y is a finite language, measured in the number of states of the minimal *dfa* accepting L'. Observe that if we take $L(\mathcal{A}) = \Sigma^*$ and $Y = 1 \cdot \Sigma^k$, then $\mathcal{A}$ can have only one state, the automaton for Y has $O(k)$ states, but the smallest *dfa* for L' needs an exponential number of states. This happens because the total size of Y is exponential with respect to k. The situation is much different when the total size of Y is part of the input size. In this case L' is accepted by a *dfa* of a polynomial size. However, for the language $Y \cdot L(\mathcal{A})$ the situations can be dramatically different. This can be seen in our next theorem, which can be found also in [19]. We present a proof which is more suitable for our considerations.

Theorem 4.1 **(i)** *Assume $\mathcal{A}$ is a deterministic automaton and Y is a finite language. Then we can construct in polynomial time a deterministic automaton accepting $L(\mathcal{A}) \cdot Y$, which has a polynomial number of states, in terms of total size of $\mathcal{A}$ and Y.*

(ii) *There are* dfa*'s $\mathcal{A}$ and finite languages Y, where the total size of $\mathcal{A}$ and Y is n, such that the number of states of the minimal* dfa *accepting $Y \cdot L(\mathcal{A})$ is exponential in n.*

Proof: Part (i) is obvious since the family $\{w^{-1}Y | w \in \Sigma^*\}$ is of polynomial size with respect to the size of Y.

Part (ii) is proved as follows: consider the language

$$L_{\neq}(n) = \{1^n\$u\#v : u, v \in \{a,b\}^n, \exists i : 1 \leq i \leq n \quad \text{and the } i\text{th letter of } u \text{ is different to that of } v\}.$$

It is easy to see that a *dfa* for $L_{\neq}(n)$ needs an exponential number of states. On the other hand, we can write

$$L_{\neq}(n) \ = \ 1^n \cdot (L_1 \cup L_2 \cup L_3 \ldots \cup L_n),$$

where each L_i is accepted by a linear size *dfa*. Indeed, L_i's are chosen to test the inequality on the i-th position.

Next we define the languages

$$\begin{aligned} &L = 1 \cdot L_1 \cup 1^2 \cdot L_2 \cup 1^3 \cdot L_3 \ldots 1^n \cdot L_n, \text{ and} \\ &Y = 1 \cup 1^2 \cup 1^3 \ldots 1^n \end{aligned}$$

The language L is accepted by a linear size *dfa*. The automaton simply reads the number of leading ones, before the symbol "\$", and then depending on this number switches to simulate the corresponding *dfa* for L_i. Then

$$(Y \cdot L) \cap (1^n\$ \cdot \Sigma^*) = L_{\neq}(n).$$

Hence any *dfa* for $Y \cdot L$ should have exponential number of states. Otherwise $L_{\neq}(n)$ would have a small *dfa*, as a product of the *dfa* for $Y \cdot L$ and linear size *dfa* for $1^n\$ \cdot \Sigma^*$. □

We continue with the following result. Here a *prefix set* is a set where no word is a prefix of another word.

Theorem 4.2 *Let $\mathcal{A}$ be a* dfa.

(i) *If Y is an explicitly given finite language then we can test the inclusion $Y \cdot L(\mathcal{A}) \subseteq L(\mathcal{A}) \cdot Y$ in deterministic polynomial time.*

(ii) *If P is an explicitly given finite prefix set then we can test the equality $P \cdot L(\mathcal{A}) = L(\mathcal{A}) \cdot P$ in deterministic polynomial time.*

Proof: Due to Theorem 4.1 we can construct a deterministic automaton $\mathcal{A}'$ for $L(\mathcal{A}) \cdot Y$ with polynomial number of states. Let $\mathcal{A}''$ be the complement of $\mathcal{A}'$. We also construct a *nfa* $\mathcal{C}$ for $Y \cdot L(\mathcal{A})$ with polynomial number of states. Then it is enough to check $L(\mathcal{C}) \cap L(\mathcal{A}'') = \emptyset$. This can be done by combining $\mathcal{C}$ and $\mathcal{A}''$ into a single *nfa* $\mathcal{D}$ with polynomial number of states. Hence, we can test $L(\mathcal{C}) \cap L(\mathcal{A}'') = \emptyset$ by checking if $L(\mathcal{D}) = \emptyset$. This problem is reducible to a path problem in a graph.

The second point follows from the fact that $P \cdot L(\mathcal{A})$ is a language accepted by a polynomial size *dfa*. □

The following fact suggests that it is unlikely that the equality $Y \cdot L(\mathcal{A}) = L(\mathcal{A}) \cdot Y$, for finite Y, and a *dfa* $\mathcal{A}$, can be tested in polynomial time.

Theorem 4.3 *Let $\mathcal{A}$ and $\mathcal{B}$ be* dfa*'s and Y an explicitly given finite language. Testing the inclusion $L(\mathcal{B}) \subseteq Y \cdot L(\mathcal{A})$ is PSPACE-complete.*

Proof: Let M be a deterministic Turing machine M working in space m for the input word w of size n. We construct a *dfa* accepting non-valid computations of M for w similarly as we constructed a *dfa* for the language $L_{\neq}(m)$.

We define a sequence of languages $L_1, L_2, \ldots L_n$ which together "test" that two consecutive configurations of M are invalid. This can be done in a similar way as for the language $L_{\neq}(n)$. We omit the details. Eventually we construct the language L such that

$$1^n\$\Sigma^* \subseteq Y \cdot L \text{ iff there is no valid accepting computation of } M.$$

The language $1^n\$\Sigma^*$ can be accepted by a *dfa* with $n + 2$ states. □

5 Finite languages

In this section we consider the commutation of finite languages, and give some complexity results which beat the naive algorithms.

Let X be an explicitly given finite language. Recall that $|X|$ denotes the number of words in X and $||X||$ the total size of all words in X, i.e. the sum of the lengths of all words in X.

We start with a few auxiliary results.

Lemma 5.1 **(i)** *If X is a finite set of words over a non-unary alphabet Σ, then*

$$|X| \leq c \frac{||X||}{\log_{|\Sigma|} ||X||} \quad \textit{for a suitable constant } c.$$

(ii) *Let X and Y are two finite languages over a finite alphabet Σ and let $|X| = n$ and $|Y| = m$. Then $||XY|| \leq m||X|| + n||Y||$.*

Proof: Part (i). For a fixed number N we consider a set of words X satisfying the following three conditions:

- $||X|| = N$,
- $||X||$ is minimal possible.

Consider a trie T for X, see [3]. The trie has $|X|$ distinguished nodes and each path from the root of T to a distinguished vertex of T is labeled by a word in X. Clearly, each leaf of T is distinguished. We prove the following three simple facts.

Claim I. T is a balanced trie.

Claim II. Any son of a vertex with less than $|\Sigma|$ sons in T is a leaf.

Claim III. Each vertex in T is distinguished.

Proofs of the claims:

I) Suppose that the difference between depths of two leaves of T is at least 2. Then we remove the deeper one and put it as a son of the other one. In this way the cardinality of the obtained set is the same as the cardinality of X, and its size is strictly smaller than $||X||$; a contradiction.

II) Suppose there is a vertex v in T having less than Σ sons which is not a leaf. Then we remove a leaf whose ancestor is v and put it as a son of v. Again the set represented by a new trie has the same number of elements as X and is of smaller size; a contradiction.

III) Since each leaf is distinguished, we may assume that an internal vertex v of T is not distinguished. Then we take any path from v to a leaf, remove v and move all other vertices of the path one level up. New trie corresponds to a set of the same cardinality as X and smaller size; a contradiction. □

By the claims we may assume that T is a full $|\Sigma|$-ary trie of $|X|$ nodes. Such a trie has at least $|X|/2$ leaves and is of height at least $\log_{|\Sigma|}|X|$. Hence,

$$\frac{|X|}{2}\log_{|\Sigma|}|X| \leq ||X||.$$

Since the function $\frac{n}{\log_{|\Sigma|} n}$, for $n > e$, is increasing the part (i) follows.

Part (ii) follows from the estimates:

$$\begin{aligned} ||XY|| &\leq \Sigma_{x\in X, y\in Y}|xy| \leq \Sigma_{x\in X}\Sigma_{y\in Y}(|x|+|y|) \leq \Sigma_{x\in X}(m|x| + \Sigma_{y\in Y}|y|) \\ &\leq \Sigma_{x\in X}(m|x| + ||Y||) \leq m||X|| + n||Y||. \end{aligned}$$

□

Now we can state our first result for finite sets.

Theorem 5.2 *Let X and Y be explicitly given finite sets over a constant size alphabet. Then the commutation problem for X and Y can be solved in $O(|X| \cdot ||Y|| + |Y| \cdot ||X||)$ time. In particular, if the total size of the sets is n, then the problem can be solved in $O(\frac{n^2}{\log n})$ time.*

Proof: First we construct a trie for the language XY. Then for each word in YX we check whether it is in XY. Next we construct a trie for the language YX and for each word in XY we check whether it is in YX.

The estimation $O(\frac{n^2}{\log n})$ is a consequence of the previous lemma. □

When most of the words in X, Y are long words then a better measure of the input size is a pair (k, n), where n is the total length of all words in $X \cup Y$ and k is the number of words. Indeed, usually $k << n^2$.

Theorem 5.3 *Let X and Y be finite explicitly given languages of total size n and cardinality at most k. Then the commutation problem for X and Y can be solved in $O((k^2 + n) \log^2 n)$ time.*

Proof: Denote $W = X \cup Y$. We use the concept of the *dictionary of basic factors*, denoted $DBF(W)$, for the set W of words, see [3]. The sub-words of words in W whose lengths are powers of two, are called basic factors. The dictionary $DBF(W)$ assigns to each sub-word w an integer $name(w) \in [1, \dots, n]$ such that each basic factor w is uniquely identified by the pair $(length(w), name(w))$.

The following fact has been shown in [3].

Claim I: $DBF(W)$ can be constructed in time $O(n \log n)$.

For words whose length is not necessarily a power of two define

$$code(w) = (length(w), name(w'), name''(w)),$$

where w', w' are respectively the largest prefix and the largest suffix of w whose length is a power of two.

For a set X of words denote: $code(X) = \{code(x) \ : \ x \in X\}$.

We replace the sets X and Y by $X' = code(X)$, $Y' = code(Y)$. for two codes x', y' of words x, y denote by $x' \otimes y' = z'$, where z' is the code of xy.

Claim II: Given a dictionary $DBF(X \cup Y)$, we can compute in $O((k^2 + n) \log n)$ time the values $x' \otimes y'$ for all pairs (x', y'), where $x' \in code(X)$, $y' \in code(Y)$.

We show the computation of $x' \otimes y'$ for example strings x, y, let

$$x' = code(x), \quad y' = code(y), \quad \text{where } \ x = abbac, \quad y = bbcabccabab$$

The code for xy will consist of the code for the maximal prefix which length is a power of two, and maximal suffix. In this particular case they are the same. The code for $z = xy$

is composed as follows:

$$\underbrace{\overbrace{abba}^{\alpha 1}\ \overbrace{c}^{\alpha 2}\ \overbrace{b}^{\alpha 3}\ \overbrace{bc}^{\alpha 4}}_{\gamma 1}\ \underbrace{abccabab}_{\gamma 2}$$

The crucial point is that the computation is reduced to the computation of product $\otimes$ for basic factors with already known names (stored in DBF). We can write

$$z' \;=\; \gamma 1 \otimes \gamma 2, \text{ where } \;\gamma 1 = \alpha 1 \otimes ((\alpha 2 \otimes \alpha 3) \otimes \alpha 4).$$

Observe that α's and γ's are codes of basic factors of words in $X \cup Y$.

In the dictionary of basic factors we can keep for each basic factor the information what are the names of its halves. Also for two basic factors of the same length we can find the name of their composition in $O(\log n)$ time. Some names of compositions of adjacent basic factors which appear in words xy should be added to the original dictionary. This can be done in $O((k^2 + n)\log^2 n)$ time.

In this way the name for each xy, where $x \in X$, $y \in Y$, can be computed using logarithmic number of queries for names of compositions of two basic factors. Altogether this cost $O(k^2 \log^2 n)$ time, since we have k^2 possible pairs xy.

It is easy to see that:

$$X' \otimes Y' = Y' \otimes X' \;\Leftrightarrow\; XY = YX$$

Now the algorithm is very simple. We compute $X' \otimes Y'$ and $Y' \otimes X'$ and test equality of two sets consisting of triples of short integers. We can do it by sorting these sets lexicographically and testing equality of sorted lists. □

6 Open problems

We conclude by stating several open problems. What is the complexity of the commutation problem when X and Y are of the following types:

1. X and Y are regular languages given by *dfa*'s?
2. X is given by a deterministic *dfa* and Y is finite?
3. X is finite and given by a deterministic *dfa* and Y is finite?
4. X is given by a *dpda* and Y is finite?
5. X is finite and given by a *dpda* and Y is finite?
6. X is regular and $Y = \Sigma$ where size of Σ is a constant?

We note that most of our problems are connected to deterministic models of automata.

References

[1] C. Choffrut, J. Karhumaki, and N. Ollinger, The commutation of finite sets: a challenging problem, *Theoret. Comput. Sci.* 273, 69–79, 2002.

[2] J. H. Conway, *Regular algebra and finite machines*, Chapman Hall, 1971.

[3] M. Crochemore and W. Rytter, *Text algorithms*, Oxford University Press, 1994.

[4] V. Diekert, Makanin's algorith, Chapter 12 in: M. Lothaire (ed.), *Algebraic combinatorics on words*, Cambridge University Press, 387–442, 2002.

[5] A. Ehrenfeucht and P. Zeiger, Complexity measures for expressions, *J. Comput. System Sci.* 12, 134–146, 1976.

[6] M.R. Garey and D.S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, San Francisco, H. Freeman, 1978.

[7] T. Harju, O. Ibarra, J. Karhumäki, and A. Salomaa, Some decision problems concerning semilinearity and commutation, *J. Comput. System Sci.* 65, 278–294, 2002.

[8] C. Hagenach and A. Muscholl, Computing ε-free NFA from regular expressions in $\mathcal{O}(n\log^2 n)$ time, *LNCS* 1450, 277–285, 1998.

[9] J. Hromkovic, S. Siebert, and T. Wilke, Translating regular expressions into small ε-free nondeterministic finite automata, *J. Comput. System Sci.* 62, 565–588, 2001.

[10] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[11] J. Karhumäki, Combinatorial and computational problems on finite sets of words, Proc. MCU 2001, *LNCS* 2055, 69–81, 2001.

[12] J. Karhumäki, A. Latteux, and I. Petre, The commutation with codes and ternary sets of words, Proceedings of STACS'03, *LNCS* 2607, 74–84, 2003.

[13] J. Karhumäki and I. Petre, Conway's Problem for three word sets, *Theoret. Comput. Sci.* 289, 705–725, 2002.

[14] E. Leiss, *Language equations*, Springer-Verlag 1998

[15] M. Lothaire, *Combinatorics on words*, Addison-Wesley, 1983

[16] G.S. Makanin, The problem of solvability of equation in a free semigroup, *Mat. Sb.* 103(2), 147–236, 1977, in Russian; English translation in: *Math USSR Sbornik* 32, 129–198, 1977.

[17] W. Plandowski, Satisfiability of word equations is in PSPACE, Proc. of FOCS99, *IEEE*, 495–500, 1999.

[18] B. Ratoandramanana, Codes et motifs, *RAIRO Theor. Informat.* 23, 425-444, 1989.

[19] S. Yu, Regular languages, in: Gh. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages Vol. I*, Springer-Verlag, 41–110, 1997.