Scientific Research Publishing

Journal of Applied Mathematics and Physics, 2024, 12, 1571-1581
https://www.scirp.org/journal/jamp
ISSN Online: 2327-4379
ISSN Print: 2327-4352

On Some Properties of Graph of Prefix Code

Nikolai I. Krainiukov[1], Mikhail E. Abramyan[1,2], Boris F. Melnikov[1]

[1]Faculty of Computational Mathematics and Cybernetics, Shenzhen MSU-BIT University, Shenzhen, China
[2]Department of Algebra and Discrete Mathematics, Southern Federal University, Rostov-on-Don, Russian Federation
Email: n.krainiukov@smbu.edu.cn, m-abramyan@yandex.ru, bormel@mail.ru

How to cite this paper: Krainiukov, N.I., Abramyan, M.E. and Melnikov, B.F. (2024) On Some Properties of Graph of Prefix Code. *Journal of Applied Mathematics and Physics*, **12**, 1571-1581.
https://doi.org/10.4236/jamp.2024.124096

Received: February 22, 2024
Accepted: April 27, 2024
Published: April 30, 2024

Abstract

We investigate decomposition of codes and finite languages. A prime decomposition is a decomposition of a code or languages into a concatenation of nontrivial prime codes or languages. A code is prime if it cannot be decomposed into at least two nontrivial codes as the same for the languages. In the paper, a linear time algorithm is designed, which finds the prime decomposition. If codes or finite languages are presented as given by its minimal deterministic automaton, then from the point of view of abstract algebra and graph theory, this automaton has special properties. The study was conducted using system for computational Discrete Algebra GAP.

Keywords

Finite Languages, Minimal Deterministic Automata, Concatenation, Codes, Graph of Automaton, Free Algebra

1. Introduction

A formal language is set of the words over some alphabet [1]. The standard set operations, like union and interception, can be performed under formal languages [2]. Another simple basic operation of formal languages is their concatenation. However, the complexity of the inverse operation of decomposing a formal language into a nontrivial concatenation of factor languages is more sophisticated and has a long history of studying [3]. A concatenation is trivial if one of the languages consists exactly of the empty string $\{\varepsilon\}$. A non-empty language is said to be prime if it cannot be written as a catenation of two languages neither one of which is the singleton language consisting of the empty word.

We investigate decomposition problems for the class of finite languages. The finite language is finite set of words. This class contends the codes, the codes are languages recognized by special kind of automata, so-called flower automata. The representation of the finite language in special graph of automata makes it poss-

ible to decompose into a product of prime languages.

Section 2 contains the basic definition and notation of formal languages, codes, graphs, finite automata and formal serials over the semiring.

Section 3 discusses the algorithm of decomposition prefix codes and finite language and a practical example of application of this algorithm.

Section 4 applies computer discrete algebra system GAP to decompose prefix codes and finite language.

2. Definitions and Notation

In this section, we remember the definition and notation about formal languages, codes, free monoid, free algebra, automata and graphs. The following definitions taken from [4] [5] [6] will be used.

An alphabet Σ is finite set letters $\Sigma = \{a, b, c, \ldots\}$. A word or string w is finite length sequence of letters over alphabet Σ. We denote as Σ^* the set of all finite words. The set of Σ^* with respect to the concatenation operation forms a free monoid. Language L is subset of monoid Σ^*. Free monoid Σ^* contains the empty word ε. Semigroup $\Sigma^+ = \Sigma^* \setminus \varepsilon$ is monoid Σ^* without empty word ε.

A basic operation of free monoid Σ^* is concatenation of two words $w = uv$. The concatenation can be expanded to the formal languages L_1, L_2. The result of concatenation is the language $= L_1 \cdot L_2 = \{w \mid w = v \cdot u, v \in L_1, u \in L_2\}$.

This operation is like integer multiplication. The inverse operation that is factorization is more complicated. There is problem to find factors/divisors for big integer numbers and prime integer numbers.

The complexity of the inverse operation of decomposing a language L into a nontrivial concatenation L_1, L_2 is also complicated.

A concatenation is obvious if one of languages L_1, L_2 consists exactly of the empty string. A language L is prime language if it cannot be decomposed to a non-trivial concatenation of two languages L_1·and L_2. The prime factorization of language is the decomposition into prime factors, prime languages.

The problem of prime factorization is undecidable for context-free languages [6]. A set X is a code if any word in X^+ can be written uniquely as a product of words in X. To say other words, word $w \in X^+$ has a unique factorization in words from X. A code X never contains the empty word ε, because word $w = \varepsilon w = w\varepsilon$ has different presentation. Any subset words from a code X is a code too.

The word u is a prefix of a word v, denoted as $u \le v$, if $v = uw$, for some $w \in \Sigma^*$. We say that u and v are prefix comparable if either $v \le u$, or $u \le v$. The set X is a prefix code if no element of X is a proper prefix of another element in X. This is equivalent to say that there are no comparable words $u \le v$ of the relation $\le$ in the set X. For all words $u, v \in X$, if $u \le v$, then $u = v$.

We use standard graph theory notions, as contained in [7], so we only fix the notation and give a few definitions and example below.

A digraph (directed graph) $G = (V, E)$ consists of a finite set of vertices V and

a set of edges $E \subseteq V^2$. For a subset of vertices $\subseteq V$, let $G[U]$ denote the induced sub(di)graph $(U, E \bigcap U \times U)$, which is obtained by restricting the vertex set V of G to U and redefining the edge set E appropriately.

The graph $G = (V, E)$ has vertices $V = \{1,2,3,4,5\}$ and edges $E = \{(1,1),(1,2),(1,3),(2,5),(5,5),(5,2),(3,3),(3,4),(4,1)\}$ (**Figure 1**).

An automaton A [2] [4] [8] over alphabet Σ consists of a set of states Q, the initial states $I \subset Q$, the final/terminal states $T \subset Q$, and a set $F \subset Q \times A \times Q$ called the set of edges. The automaton is denoted by:

$$A = (Q, F, I, T)$$

The automaton is finite when the set Q is finite. The language L is recognized by A, denoted $L(A)$, is the set of words in Σ^* which are labels of paths from I to T.

Figure 2 shows the automaton A with four states, the set of initial states $I = \{1\}$, the set of terminal states $T = \{3,4\}$, the set of edges $F = \{(1,a,2),(1,a,4),(2,a,3),(2,b,4)\}$. The finite language $L(A) = \{aa, ab, b\}$ is recognized by automaton A.

Recall that an algebra [9] over a field K is a K-vector space A with a binary operation (multiplication) $A \times A \to A$, $(a,b) \to ab$ specified on it, satisfying the following requirements:

1) $a(b+c) = ab + ac$, $(b+c)a = ba + ca$ for any $a, b, c \in A$;

2) $(\lambda a)b = a(\lambda b) = \lambda(ab)$ for any $\lambda \in K$, $a, b \in A$.

We will additionally assume that:

3) There is a unit in A, *i.e.* an element 1 such that $1a = a1 = a$ for any $a \in A$;

4) Algebra A is associative, *i.e.* $(ab)c = a(bc)$ for any $a, b, c \in A$.

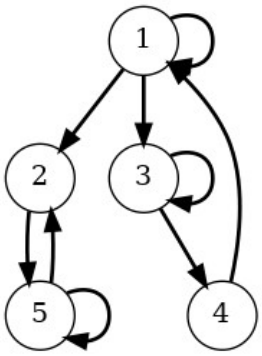

Figure 1. Graph G with $V = 5$ vertices and $E = 9$ edges.

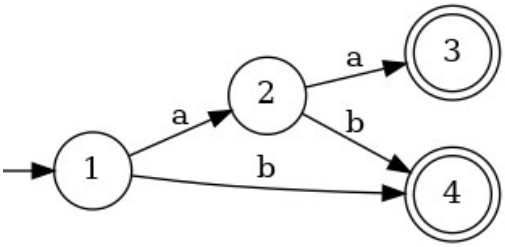

Figure 2. Automaton A with four states.

Throughout the following, we will additionally assume that the field K is the field of rational numbers or the field $Z/2Z$. We can embed monoid Σ^* over alphabet $\Sigma = \{x_1, x_2, \ldots x_n\}$ into free algebra of polynomials $K[x_1, x_2, \ldots x_n]$ with homomorphism $\varphi : \Sigma^* \to K[x_1, x_2, \ldots x_n]$ by definition on letters of alphabet Σ:

$$\varphi(x_i) = x_i, i = 1, \ldots, n.$$

We need to use the formal series over the alphabet Σ with coefficients in semiring K. We recall the definition of semiring [10].

Let K be a semiring.

A semiring K is a set equipped with two operations denoted + and $\cdot$ satisfying the following axioms:

1) The set K is a commutative monoid for addition + with a neutral element denoted by 0;

2) The set K is a monoid for multiplication $\cdot$ with a neutral element denoted by 1;

3) Multiplication is distributive on addition;

4) For all $x \in K$, $0 \cdot x = x \cdot 0 = 0$.

A formal series over alphabet Σ with coefficients in K is a mapping:

$$\sigma : \Sigma^* \to K$$

We denote the set of formal series σ over Σ^* by K^{Σ^*} or $K \ll \Sigma^* \gg$. The value of σ on $w \in \Sigma^*$ is denoted (σ, w) and we can formally denote:

$$\sigma = \sum\nolimits_{w \in \Sigma^*} (\sigma, w)$$

The support of a series $\sigma \in K \ll \Sigma^* \gg$ is the set:

$$supp(\sigma) = \{w \in \Sigma^* \mid (\sigma, w) \neq 0\}.$$

If the set $supp(\sigma)$ is finite, then we denote that set of formal series σ by $K < \Sigma^* >$.

Another words the set of formal series $\sigma \in K^{\Sigma^*}$ such that $(\sigma, w) = 0$ for all but a finite number of $w \in \Sigma^*$ is denoted $K < \Sigma^* >$ and then σ is called a polynomial.

We define the formal series $+\tau$, $\sigma\tau$, and $k\sigma$ by:

$$(\sigma + \tau, w) = (\sigma, w) + (\tau, w)$$

$$(\sigma \cdot \tau, w) = \sum\nolimits_{w = uv \in \Sigma^*} (\sigma, u)(\tau, v)$$

$$(k\sigma, w) = k(\sigma, w).$$

For example, let K is Boolean semiring with addition and multiplication table:

$$0 + 0 = 0,\ 0 + 1 = 1 + 0 = 1,\ 1 + 1 = 1$$

$$0 \cdot 0 = 0,\ 1 \cdot 0 = 0 \cdot 1 = 0,\ 1 \cdot 1 = 1.$$

alphabet $\Sigma = \{a, b\}$,

and series σ, τ is defined $(\sigma, a) = 1$, $(\sigma, ab) = 1$, $(\tau, \varepsilon) = 1$, $(\tau, b) = 1$:

$$\sigma = 1 \cdot a + 1 \cdot ab$$

$$\tau = 1 \cdot \varepsilon + 1 \cdot ab$$

Then,

$$\sigma + \tau = 1 \cdot \varepsilon + 1 \cdot a + 1 \cdot ab$$

$$\sigma \cdot \tau = 1 \cdot a + 1 \cdot ab + 1 \cdot aab + 1 \cdot abab$$

It is easy to see that in this case, $K < \Sigma^* >$ is a free semiring of noncommutative polynomials and for finite languages $X, Y \subseteq \Sigma^*$ $\sigma_{X \cup Y} = \sigma_X + \sigma_Y$, $\sigma_{X \cdot Y} = \sigma_X \cdot \sigma_Y$.

3. Decomposition of Codes and Finite Languages

At the first, we consider decomposition problems for the class of finite prefix codes. Then, subset X is a prefix code if no element of X is a proper prefix of another element in X. This is equivalent to the fact that there are no comparable words $u \le v$ of the relation $\le$ in the set X. That is for all words, $v \in X$, if $u \le v$, then $u = v$. For example, the set $X = \{bb, aba\}$ is prefix code. A convenient representation for the prefix code is a tree view.

Figure 3 shows the presentation of prefix code $X = \{bb, aba\}$.

The bold lines present the words of code, the dotted lines present the words $\in \Sigma^*$. A given code X can be associated a subtree of the literal representation of this code X. The infinite tree may present the free monoid Σ^*. In this case the root of tree of relation $\le$ over Σ^* is drawing in **Figure 3** as node 1.

Regular prefix codes are languages recognized by finite automata and such that no word is a prefix of another. The representation of the code is a deterministic finite automaton. Let two automaton A_1, A_2 present two prefix codes X_1, X_2 appropriately, then we can draw finite set of words $X = X_1 \cdot X_2$ same kind like **Figure 3**. In this case, tree presents code X consists of the words of prefix code X_1 and then words of prefix code X_2. It is easy to see that X is prefix code

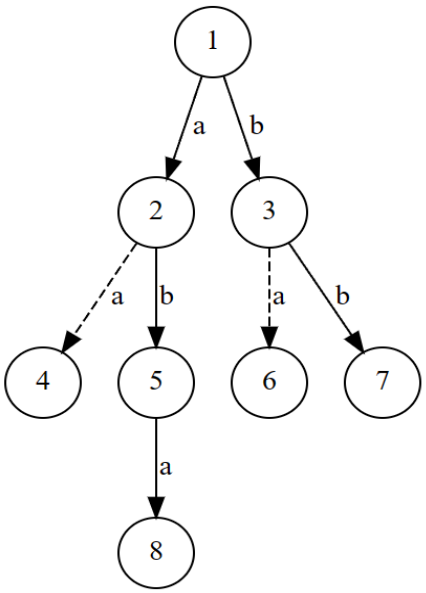

Figure 3. Presentation of prefix code X.

too. The word of code X is the leaf of this tree. There is the only (unambiguous) path from root of this tree to the leafs for each word of code X. Then, deterministic automaton A, which presents code X, has a graph like in **Figure 4**. The words $u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n$ are words of codes $X_1 = \{u_1, u_2, \ldots, u_n\}$ and $X_2 = \{v_1, v_2, \ldots, v_n\}$, where n, m are the number of words in codes X_1 and X_2.

We denote the vertex $V_{divisor}$ of graph $G = (V, E)$ of minimal deterministic automaton A and call divisor-border vertex $V_{divisor}$ of automaton A. The DB-vertex of complete minimal deterministic automaton A [11] of prefix code X is the vertex that we have to divided graph G by this vertex on several automaton A_i. The automaton A_i is automaton appropriately factors X_i of code $X = X_1 X_2 \ldots X_k$.

Theorem

A prefix code $X = X_1 X_2 \ldots X_k$ is divided on k prefix code X_i if and only if a prefix code X has graph G of its minimal deterministic automaton A with $(k-1)$ DB-vertex.

Proof

If prefix code $X = X_1 X_2 \ldots X_k$ is divided on k prefix code X_i, then the words of this code X present the unambiguous paths from root of the tree for prefix code X to the leaves of the tree. So, we can factorize this tree after the end of words each codes X_i as the result of this operation is like **Figure 4**. Then, graph G of its minimal deterministic automaton A has $(k-1)$ DB-vertex.

If the graph G has $(k-1)$ DB-vertex, we can prove by mathematical induction. The theorem is obvious for =2. Suppose that it is true for $k = n$, then prove it for $k = n+1$. We can divide the graph G on last DB-vertex V_n on two graph G_n and G_1. For graph G_n, we have presentation $X_1 X_2 \ldots X_n$ and just concatenate the words of automaton for graph G_1. The results $X = X_1 X_2 \ldots X_n X_{n+1}$.

Example

For prime prefix code $X_1 = \{"a","ba","bb"\}$, $X_2 = \{"aa","ba","bb"\}$.

$X = X_1 X_2 = \{"aaa","aba","abb","baaa","baba","babb","bbaa","bbba","bbbb"\}$

We construct the automaton from the prefix code X.

```
Display(A);
   |  1   2   3   4   5   6   7
---------------------------------------
 a |  5   4   1   6   5   1   4
 b |  5   7   1   3   5   5   4
Initial state:     [ 2 ]
Accepting state: [ 1 ]
```

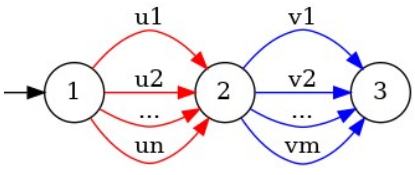

Figure 4. Graph of automaton A.

Figure 5 shows the automaton A with DB-vertex (state) $V=4$.

The state $V=5$ is a "dead" state of automaton A.

Figure 6 shows Graph G_1 and G_2 for automata A.

We decompose the prefix code $X=X_1X_2$ to the product of two prefix codes X_1, X_2.

Now, we can formulate Theorem of decomposing for finite language L.

Theorem

A finite language $L=L_1L_2\ldots L$ is divided on k finite languages L_i if and only if a finite language L has graph G of its minimal deterministic automaton A with $(k-1)$ DB-vertex and the number of final state automaton A is equal one.

Proof

It is the same as the Theorem for prefix codex.

Algorithm for factorization prefixes code and finite language.

Input

An finite prefix set $X=\{u_1,u_2,\ldots,u_n\}$.

Step 1

Built the minimal deterministic automaton A of prefix set X.

Step 2

Define the number n of DB-vertex in the graph of automaton A.

It can be do with complexity of $O(n\cdot\max(\text{length of words in prefix set } X))$.

Step 3

Cut the graph of automaton A and find $n+1$ factors of prefix set $X=X_1X_2\ldots X_nX_{n+1}$.

There is algebraic approach to decomposing the finite language. The main idea is the embedding language $L=\{v_1,v_2,\ldots,v_n\}$ into free algebra of polynomials $K[x_1,x_2,\ldots,x_n]$ or set of formal series σ over semiring K. We can embed language $L\subseteq\Sigma^*$ over alphabet $\Sigma=\{x_1,x_2,\ldots,x_n\}$ with homomorphism $\varphi:\Sigma^*\to K[x_1,x_2,\ldots,x_n]$ by definition on letters of alphabet Σ:

$$\varphi(x_i)=x_i,\ i=1,\ldots,n$$

The homomorphism $\varphi(v_i)=p_i$ maps the word $v_i\in L$ into monomials $^\circ$ p_i.

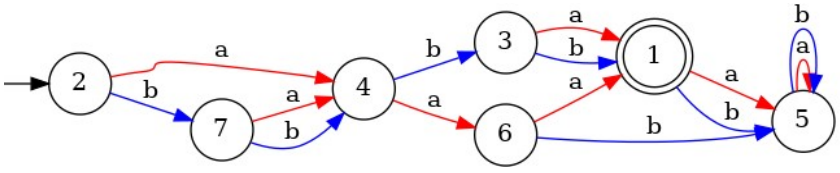

Figure 5. Automaton A with DB-vertex $V=4$.

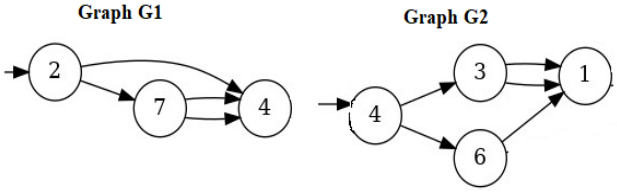

Figure 6. Graph G_1 and G_2 for automata A.

The set of monomials generate the ideal I_L. Then, we can try to find the divisors of this ideal I_L. We use an exhaustive search for the divisors just added monomials q_d with less power to the ideal I_L and apply Buchberger algorithm for this set to find the noncommutative Groebner basis. If the Groebner basis will be equals the added monomials q_d, then we find this divisors for finite language L. This algorithm has exponential time, but it will be ended because finite alphabet Σ and finite language L.

4. Apply System GAP for Decomposition Prefix Codes and Finite Language

The computer discrete algebra system GAP is the system for Groups, Algorithms and Programming. The name GAP reflects the original aim of the system, but now GAP has many packages for wide field investigation of modern algebra. The discrete algebra system GAP has become somewhat broader, and contained the information about algorithms and programming for other algebraic structures, such as semigroups, algebras, discrete automata and many other things.

Below describes the data and structures used in packages "automata" for finite deterministic and nondeterministic automata [12] [13] and some functions to determine property about them.

We can easy create free monoid over alphabet $\Sigma = \{a, b\}$.

```
gap> m1:=FreeMonoid(["a","b"]);
<free monoid on the generators [ a, b ]>
gap> gm1:=GeneratorsOfMonoid(m1);
[ a, b ]
gap> a:=gm1[1];
a
gap> b:=gm1[2];
b
```

alphabet $\Sigma = \{a, b\}$.

It is easy to form the list of elements free monoid (associative words) and product of two lists:

```
gap> s1:=[a,b*a,b*b];
[ a, b*a, b^2 ]
gap> s2:=[a*a,b*a,b*b];
[ a^2, b*a, b^2 ]
gap> s3:=Mult_Sp(sp1,s2);
[ a^3, a*b*a, a*b^2, b*a^3, (b*a)^2, b*a*b^2, b^2*a^2, b^3*a, b^4 ]
```

Now, we transform the list of associative words to list of words:

```
gap> aw1:=AssocWord_Sp(s3, gm1);
[ "aaa", "aba", "abb", "baaa", "baba", "babb", "bbaa", "bbba", "bbbb" ]
```

Build the minimal deterministic automaton *new3* from list of words:

```
new3:=ListOfWordsToAutomaton("ab",aw1);
gap> Display(new3);
```

```
    |  1  2  3  4  5  6  7
----------------------------------------
 a |  5  4  1  6  5  1  4
 b |  5  7  1  3  5  5  4
Initial state:     [ 2 ]
Accepting state: [ 1 ]
```

Then, we can draw the graph of automaton *new3:*

newS3:=DotStringForDrawingAutomaton(new3);

At the last, we discuss algebraic algorithm decomposing the finite set of words free monoid Σ*.

We can embedding the finite set of words set $X=\{u_1,u_2,\ldots,u_n\}$ in free semiring of polynomials as described earlier $\varphi:\Sigma^*\to K[x_1,x_2,\ldots,x_n]$ and find the noncommutative Groebner bases in this ideal generate by polynomials from words $\{u_1,u_2,\ldots,u_n\}$ [14]. In our example, K is field of rational numbers.

```
gap> A1:=FreeAssociativeAlgebraWithOne(Rationals, "a", "b");;
gap> gA1:=GeneratorsOfAlgebraWithOne(A1);
[ (1)*a, (1)*b ]
gap> e:=One(A1);
(1)*<identity ...>
gap> a:=gA1[1];
(1)*a
gap> b:=gA1[2];
(1)*b
```

We defined the generators of free associative algebra with one.

Now, we use the list of associative words as the prefix code. The prefix code is the same as above example with automaton *new3*.

```
gap> s2:=[a*a,b*a,b*b];
[ a^2, b*a, b^2 ]
gap> s3:=[ a^3, a*b*a, a*b^2, b*a^3, (b*a)^2, b*a*b^2, b^2*a^2, b^3*a, b^4 ];;
gap> pL2:=[ (1)*a^3, (1)*a*b*a, (1)*a*b^2, (1)*b*a^3, (1)*(b*a)^2, (1)*b*a*b^2, (1)*b^2*a^2, (1)*b^3*a, (1)*b^4,(1)*a^2, (1)*b*a, (1)*b^2 ];;
gL2:=GP2NPList(pL2);
[ [ [ [ 1, 1, 1 ] ], [ 1 ] ], [ [ [ 1, 2, 1 ] ], [ 1 ] ], [ [ [ 1, 2, 2 ] ], [ 1 ] ], [ [ [ 2, 1, 1, 1 ] ], [ 1 ] ],
  [ [ [ 2, 1, 2, 1 ] ], [ 1 ] ], [ [ [ 2, 1, 2, 2 ] ], [ 1 ] ], [ [ [ 2, 2, 1, 1 ] ], [ 1 ] ],
  [ [ [ 2, 2, 2, 1 ] ], [ 1 ] ], [ [ [ 2, 2, 2, 2 ] ], [ 1 ] ], [ [ [ 1, 1 ] ], [ 1 ] ], [ [ [ 2, 1 ] ], [ 1 ] ],
  [ [ [ 2, 2 ] ], [ 1 ] ] ][  [ [ 1, 2, 2 ], [ 1, 2, 1 ], [ 1, 1, 1 ], [ 1 ] ], [ 1, 1, 1, 1 ] ]
```

Function *GP2NPList()* transforms list of polynomials into internal presentation polynomials in the package ***gbnp*** . Now, we use the list *gL2* for

```
gap> GB := Grobner(gL2);
#I  number of entered polynomials is 12
#I  number of polynomials after reduction is 3
```

```
#I  End of phase I
#I  End of phase II
#I  List of todo lengths is [ 0 ]
#I  End of phase III
#I  The computation took 2 msecs.
[ [ [ [ 1, 1 ] ], [ 1 ] ], [ [ [ 2, 1 ] ], [ 1 ] ], [ [ [ 2, 2 ] ], [ 1 ] ] ]
gap> PrintNPList(GB);
 a^2
 ba
 b^2
```

The result polynomials a^2, ba, b^2 consists a basis of divisor for polynomials list $[a^3, a*b*a, a*b^2, b*a^3, (b*a)^2, b*a*b^2, b^2*a^2, b^3*a, b^4, a^2, ba, b^2]$. Thus, we obtain the decomposition of the prefix code into products of codes using the Buchberger algorithm. Similarly, factorization can be obtained for a finite language.

5. Conclusions

The linear time algorithm is designed, which finds the prime decomposition for prefix codes. It can be applied to some modifications for finite languages.

The algorithm for factorization of finite languages has exponential complexity. From the point of view of abstract algebra, we can apply Buchberger algorithm for an exhaustive search of the factors of the finite language.

The study was conducted using system for computational Discrete Algebra GAP. The results of decomposition of languages are obtained when considering modified graphs of this automaton.

There is a big gap between complexity factorization of prefix codes (lineal complexity) and finite languages (exponential complexity). Further research will related to the search for a polynomial algorithm to determine the class of languages for which such an algorithm is applicable.

Founding

This work is supported by a grant from the research program of Chinese universities "Higher Education Stability Support Program" (Section "Shenzhen 2022—Science, Technology and Innovation Commission of Shenzhen Municipality").

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Aho, A. and Ullman, J. (1973) The Theory of Parsing, Translation, and Compiling. Vol. 1, Prentice Hall, Upper Saddle River.

[2] Brauer, W. (1984) Automation Theory: An Introduction to the Theory of Finite Au-

tomata. Vieweg + Teubner Verlag, Wiesbaden.

[3] Mateescu, A., Salomaa, A. and Yu, S. (1998) On the Decomposition of Finite Languages. Technical Report 222, Turku Centre for Computer Science, Turku.

[4] Lallement, G. (1979) Semigroups and Combinatorial Applications. Wiley & Sons, Inc., Hoboken, NJ, 376 p.

[5] Berstel, J. and Perrin, D. (2008) Theory of Codes. Academic Press, New York, 345 p.

[6] Mateescu, A., Salomaa, A. and Yu, S. (2002) Factorizations of Languages and Commutativity Conditions. *Acta Cybernetica*, **15**, 339-351.

[7] Diestel, R. (2005) Graph Theory. 3rd Edition, Springer, Berlin, 421 p.

[8] Melnikov, B. (2018) Regular Languages and Nondeterministic Finite Automata. RGSU Publ., Moscow. (In Russian)

[9] Winberg, E.B. (2005) Course of Algebra. Factorial Press, Providence. (In Russian)

[10] Berstel, J. and Perrin, D. (2005) Codes and Automata. Springer, Berlin, 545 p.

[11] Melnikov, B. (2017) The Complete Finite Automaton. *International Journal of Open Information Technologies*, **5**, 9-17.

[12] Melnikov, B. and Dolgov, V. (2022) Simplified Regular Languages and a Special Equivalence Relation on the Class of Regular Languages. Part I. *International Journal of Open Information Technologies*, **10**, 12-20. (In Russian)

[13] Abramyan, M.E. (2021) Computing the Weight of Subtasks in State Minimization of Nondeterministic Finite Automata by the Branch and Bound Method. *University Proceedings. Volga Region. Physical and Mathematical Sciences*, **2**, 46-52. (In Russian) https://doi.org/10.21685/2072-3040-2021-2-4

[14] Mora, T. (1994) An Introduction to Commutative and Noncommutative Griibner Bases. *Theoretical Computer Science*, **134**, 131-173. https://doi.org/10.1016/0304-3975(94)90283-6