World Academy of Science, Engineering and Technology
International Journal of Mathematical and Computational Sciences
Vol:18, No:8, 2024

# On Decomposition of Maximal Prefix Codes

Nikolai Krainiukov, Boris Melnikov

Open Science Index, Mathematical and Computational Sciences Vol:18, No:8, 2024 publications.waset.org/10013756.pdf

***Abstract*****—**We study the properties of maximal prefix codes. The codes have many applications in computer science, theory of formal languages, data processing and data classification. Our approaches to study use finite state automata (so-called flower automata) for the representation of prefix codes. An important task is the decomposition of prefix codes into prime prefix codes (factors). We discuss properties of such prefix code decompositions. A linear time algorithm is designed to find the prime decomposition. We used the GAP computer algebra system, which allows us to perform algebraic operations for free semigroups, monoids and automata.

***Keywords*****—**Maximal prefix code, regular languages, flower automata, prefix code decomposing.

## I. Introduction

THIS paper discusses some properties of codes and maximal prefix codes. Practical application is based on the representation of the maximal prefix codes as a sequence of words in specific order. Many books [1]-[5] and papers [6]-[8] have shown how the most important theoretical tools are used in this technology.

We investigate decomposition problems for the class of finite prefix codes. The finite prefix code is a set of words, such that none of the words in the code is a prefix of another word. In other words, none of the words in the code are the left divisor of another word in that code. Prefix codes are languages recognized by special kind automata, so called flower automata. This representation of the finite prefix code makes it possible to decompose this code into a product of factor codes.

From a practical point of view, this type of representation and decomposition algorithm can be used in some classification algorithms in hardware technologies.

Section II of the paper contains preliminary information; terminology related to formal languages, codes and finite automata. Section III discusses the properties of maximal prefix codes and presentation codes with flower automata. Section IV describes algorithm of factorization of finite code.

## II. Basic Definition: Words and Automata

In this section, we give the necessary notions related to words, finite automata, codes and properties of codes. The following definitions taken from [1]-[3] will be used.

Let us consider a finite set letters $\Sigma = \{a, b, c \dots\}$ which we call an alphabet $\Sigma$. A word or string $w$ is finite length sequence of letters over alphabet $\Sigma$. The word $w \in \sum^*$ is an element of $\Sigma^*$ and $\Sigma^*$ is the set of all finite words over $\Sigma$ . The set of $\Sigma^*$ with respect to the concatenation operation forms a free monoid. Language $L$ is subset of monoid $\Sigma^*$. Free monoid $\Sigma^*$ contains the empty word $\varepsilon$ . Semigroup $\Sigma^+ = \Sigma^* \setminus \varepsilon$ is monoid $\Sigma^*$ without empty word $\varepsilon$.

A basic operation of formal languages is concatenation of two words $w = uv$. The concatenation can be expanded to the formal languages $L = L1 \cdot L2$ . But the complexity of the inverse operation of decomposing a language $L$ into a nontrivial concatenation $L1, L2$ is not well understood and more complicated. A concatenation is obvious if one of languages $L1, L2$ consists exactly of the empty string. A language $L$ is prime if it cannot be expressed as a non-trivial concatenation of two languages $L1 \cdot L2$ . The prime factorization is the decomposition into prime factors.

The problem of prime factorization is undecidable for context-free languages [8], [9].

The word $u$ is a prefix of a word $v$, denoted as $u \leq v$, if $v = uw$ , for some $w \in \sum^*$ . We say that $u$ and $v$ are prefix comparable if either $v \leq u$, or $u \leq v$.

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 $X$ . This is the reason, why a code $X$ never contains the empty word $\varepsilon$, because word $w = \varepsilon w = w\varepsilon$ has different presentation. It is easy to see that any subset words from a code $X$ is a code.

A finite automaton $A$ over alphabet $\Sigma$ consists of a finite 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:

$$A = (Q, \Sigma, I, T).$$

The automaton is finite when the set Q is finite.

Fig. 1 shows the automaton with three states, the set of initial states $I = \{1\}$, the set of terminal states $T = \{3\}$, the set of edges $T = \{(1, a, 1), (1, a, 2), (1, a, 3), (1, b, 3), (3, b, 3)\}$.

## III. Maximal Prefix Codes and Presentation Codes with Flower Automata

In order to determine whether the set $X$ is a code, there are criteria characterizing this property. One of these properties [1], [3] is consisted in the following:

If a subset X of $\Sigma^*$ is a code, then any morphism $\beta : \Delta^* \to \Sigma^*$ which induces a bijection of some alphabet $\Delta$ onto $X$ is injective.

N. Krainiukov is with Shenzhen MSU–BIT University, Shenzhen, People's Republic of China (corresponding author, phone: 135-287-049-16; e-mail: 6620210015@smbu.edu.cn).

B. Melnikov is with Shenzhen MSU–BIT University, Shenzhen, People's Republic of China (e-mail: bormel@mail.ru).

The work is supported by a grant from the scientific program of Chinese universities “Higher Education Stability Support Program” (section “Shenzhen 2022 – Science, Technology and Innovation Commission of Shenzhen Municipality”) – 深圳 市2022年高等院校定支持划助目.

Open Science Index, Mathematical and Computational Sciences Vol:18, No:8, 2024 publications.waset.org/10013756.pdf

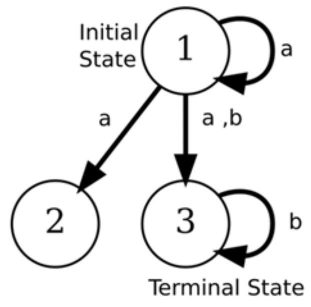

Fig. 1 Automaton with three states

The set $X = \{a, ab, aba\}$ is not a code since the word $w = aaba$ has two distinct factorizations $w = a(aba) = (a)(ab)(a)$.

Let us define the relation $u \leq v$ a word $u$ is the left divisor of another word $v$. The infinite tree may present the free monoid $\sum^*$. In this case, the root of tree of relation $\leq$ over $\sum^*$ is shown as in Fig. 2 as node 1. The root of the tree presents empty word $\varepsilon$. Each node of the tree represents a word $w$ in $\sum^*$. 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 $u \leq v$ of the relation $\leq$ in the set. For all words $u, v \in X$, if $u \leq v$ then $u = v$.

The set $X = \{bb, aba\}$ is prefix code. A convenient representation for the prefix code is a tree view. Fig. 2 shows the presentation of prefix code $X = \{bb, aba\}$. The bold lines present the words of code, the dotted lines present the words $\in \sum^*$. To a given code $X$ of $\sum^*$ can be associated a subtree of the literal representation of the prefix code $X$ is called the maximal prefix code if it is prefix and if it is properly contained in no other prefix code $Y$ of $\sum^*$, that is, if $X \subset Y \subset \sum^*$ and $Y$ prefix code implies $X = Y$.

The prefix code $X = \{bb, aba\}$ is not the maximal prefix code because we can added words $\{aa, ab, abb,\}$ to code. These words are not comparable in relation $\leq$ with words $\{bb, aba\}$.

Let us consider a convenient representation of the finite code $X = \{v1, v2, \ldots vn\}$ in the form of a flower automaton $A$. The words $v1, v2, \ldots vn$ are the edges of the flower automaton.

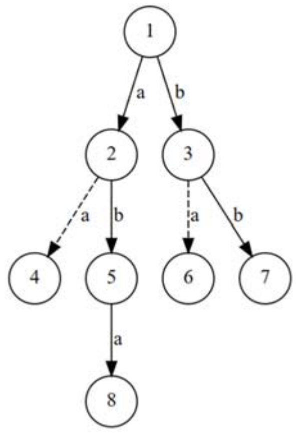

Fig. 2 Prefix code X={aba, bb}

Fig. 3 shows flower automaton for code $= \{bb, aba\}$. The initial and terminal states are the same states $\{1\}$. All words of code $X$ are the path from state $\{1\}$ to state $\{1\}$.

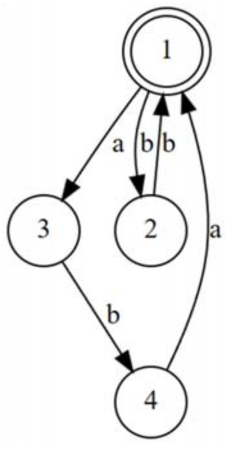

Fig. 3 Flower automaton for code X = {bb, aba}

## IV. Algorithm of Factorization of Finite Code

### A. Problem of Factorization

The problem of prime factorizations of natural numbers is fundamental in theory of numbers. In the theory of formal languages, the problem factorization of formal languages was introduced already at a very early stage [8], [9]. Let us consider this problem about finite languages. When a nonempty finite language $L$ can be written as a product: $L = L1 \cdot L2 \cdot \ldots \cdot Lk$, where of the factors $Li, 1 \leq i \leq k$, finite languages.

The problem of factorization languages is related with the subsets of potential roots in the problems of the formal languages theory [10]-[13].

### B. Factorization of Codes

The algorithm for solving this problem for finite codes is described below.

Let the code $X$ is the product of two codes $X1, X2$ $X = X1 \cdot X2 = \{w_1, w_2, \ldots, w_l\}$, where the words are lexicographically ordered.

For certainty, we denote the words of the code $X$ in the form of a product of words $w_{ij} = v_i u_j$.

TABLE I
PRODUCT OF TWO CODES

| | $u_1$ | $\cdots$ | $u_m$ |
|---|---|---|---|
| $v_1$ | $v_1u_1$ | $\cdots$ | $v_1u_m$ |
| $\cdots$ | | | |
| $\cdots$ | | | |
| $v_n$ | $v_nu_1$ | .... | $v_nu_m$ |

Algorithm of codes factorization:

Step1. Choose a word $w_{min}$ of minimum length from $X = \{w_1, w_2, \ldots, w_l\}$.

Step2. Choose words $A_1 = \{w_{11}, w_{12}, \ldots, w_{1k}\}$ from code $X$ starting with the same (first) letter as the word $w_{min}$ of minimum length.

Step3. From the set $A_1$ we choose the words

$A_2 = \{w_{21}, w_{22}, \dots, w_{2k}\}$ the second letter, which coincides with the second letter of the word $w_{min}$ .

…

StepM. From the set $A_{M-1}$ we choose the words $A_M = \{w_{M1}, w_{M2}, \dots, w_{Mk}\}$ the Mth-order letter, which coincides with the Mth-order letter of the word $w_{min}$ of minimum length. If set $A_M = \emptyset$ go to step M+1, if it is not go to next step.

StepM+1. The set $A_{M-1}$ will contain words whose left divisor will coincide with maximum part the minimum word $w_{min}$ in the code $X1$. This part will be equal the word $v_{min}$ of minimal length in code X1. The left division of words in set $A_{M-1}$ to this word $v_{min}$ gives the words of the code X2.

StepM+2. Now we got the words from the code $X2$. To receive words for the code $X1$ we do the right division of the words from the set $X$. That we will get the words of the
code $X1$.

The algorithm will finish its work after $K$ steps, where $K$ is the number of letters in a word of minimum length in the code X.

The uniqueness of the factorization follows from the construction of the solution and the uniqueness of the word of minimum length.

The algorithm has complexity $O(n)$, where $n$ the number of words code $X$ because it is necessary to search among all the words of the source code $X$. Thus, it is possible to obtain a factor decomposition of the code in linear time from the sum of the lengths of all the words of the source code.

Let's consider the problem of decomposition for language in product of two languages. This is a very difficult problem for context-free languages, since this problem is unsolvable algorithmically, that is, there is no algorithm that would allow representing a context-free language as a product of two languages.

If the final language is given in the form of a list of words, then the factorization problem is obviously solved by a complete search of the options for decomposing words into factors, that factorization problem is obviously solved by brute force. Since the description of the final language as a list can be exponentially larger than the corresponding DFA, the algorithm (brute force) is not useful for solving our problem.

To check the prime factorization of the finite language $L$, we need to consider whether there are languages $L_1$ and $L_2$ they decompose $L = L1 \cdot L2$ , because we need to work with language $L$ in forming a determinate finite automaton (DFA) we need to study the states in which words are actually separated.

Let's use the finite regular language $L$, giving both DFA $A = (Q, \Sigma, \delta, \{s\}, F)$ and $P \subseteq Q$ a set of states. We call the $P$ set of sections and define the regular finite languages:

$$L_1^P := \{w \in \Sigma^* | \delta(s, w) \in P\}$$

$$L_2^P := \bigcap_{p \in P} \{w \in \Sigma^* | \delta(p, w) \in F\}$$

The set of section $P$ allows us search decompositions to those that arise from this construction, because this construction gives factorization in form $\supseteq L_1^P \cdot L_2^P$ .

The problem is, after guessing the set of partitions of $P$, to check whether $L \subseteq L_1^P \cdot L_2^P$. Unfortunately, the intersection of $O(n)$ sets and the union of two languages are ineffective, since both can lead to an exponential increase in the number of states of the corresponding automaton. Therefore, this algorithm for factoring finite languages will probably have exponential complexity, but in prefix codes the algorithm is more simple.

## C. Practical Application

To verify the correctness of the proposed algorithms, we implemented a system computer algebra *GAP* that accurately performs the logical flow of algorithm cycle by cycle.

The *GAP* system is a free, open software package for computation in discrete abstract algebra. We used also package “automata” in *GAP* system. Using this package, we can build deterministic and non-deterministic finite automata, perform operations with regular languages and regular expressions. In this package, we have implemented the algebraic part. This part of the “automata” package aims to enhance its utility for finite automata and formal languages theorists, since monoids are already implemented in *GAP*, we can take advantage of this fact.

Package has the function to compute a flower automaton in order to obtain a finite automaton corresponding to the given prefix code. Below is the example of program code for construction free mononid `m1` with two generators:

```
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
```

It is easy to form a list of elements of free monoids (associative words) and the product of two lists, `s1` and `s2`:

```
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 will convert the list of associative words into a list of words:

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

With functions from packages we construct a minimal deterministic automaton `new_3` from a list of words `aw1`:

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

Open Science Index, Mathematical and Computational Sciences Vol:18, No:8, 2024 publications.waset.org/10013756.pdf

```
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 ]
```

The external program *Graphviz* is used for graph visualization, allowing users to visualize automata. Its first versions were developed by AT&T, and now it is available as a set of utilities and libraries, as well as in source code under the Eclipse Public License (EPL). Its diagram engine uses the DOT graph description language, which is a textual description of the graph structure: vertices, their connections, groups, and attributes for their visual design. This convenient tool presently operates smoothly under *LINUX*.

## V. Conclusion

The article discusses the basic algorithms for the decomposition of prefix codes and their practical implementation in the system computer algebra $GAP$. The complexity of the decomposition algorithm is $O(n)$, where $n$ is the number of words in the prefix code. The application of an approximate algorithm for finite languages will probably be discussed in the following articles.

## References

[1] G. Lallement *Semigroups and Combinatorial Applications.* – NJ, Wiley & Sons, Inc. – 1979. – 376 p
[2] J. Berstel, D. Perrin, *Theory of Codes*, Academic Press, New York, 1985.
[3] W. Brauer. Introduction in the Finite Automata Theory. Moscow: Radio I Svyaz Publ., 1987, 390 p. (in Russian).
[4] M. Lothaire, Algebraic Combinatorics on Words, Cambridge University Press, Cambridge, 2002
[5] B. Melnikov. *Regular languages and nondeterministic finite automata.* Moscow: RGSU Publ., 2018. 179 p. (in Russian).
[6] B.F. Melnikov, A. A. Melnikova Polynomial algorithm for checking the fulfillment of the condition of the morphic image of the extended maximal prefix code, *International Journal of Open Information Technologies*. – 2022. – Vol. 10. No. 12. – P. 1–9 (in Russian).
[7] V. Dolgov, B. Melnikov, A. Melnikova. "The loops of the basis finite automaton and the connected questions." Bulletin of the Voronezh State University. Series: Physics. Mathematics, no. 4, pp. 95–111, 2016 (in Russian).
[8] A. Mateescu, A. Salomaa, S. Yu, On the decomposition of finite languages. *Technical Report 222*, Turku Centre for Computer Science (1998).
[9] Mateescu, A., Salomaa, A., S. Yu, (2002). Factorizations of languages and commutativity conditions. *Acta Cybernetica*, 15(3), 339-351.
[10] B.F. Melnikov. Semi-lattices of the subsets of potential roots in the problems of the formal languages theory. Part I. Extracting the root from the language. *International Journal of Open Information Technologies*. 2022. Vol. 10. No. 4. P. 1–9 (in Russian).
[11] B.F. Melnikov Semi-lattices of the subsets of potential roots in theproblems of the formal languages theory. Part II. Constructing an inverse morphism *International Journal of Open Information Technologies.* 2022. Vol. 10. No 5. P. 1–8 (in Russian).
[12] B.F. Melnikov Semi-lattices of the subsets of potential roots in the problems of the formal languages theory. Part III. The condition for the existence of a lattice*, International Journal of Open Information Technologies*. 2022. Vol. 10. No. 7. P. 1–9 (in Russian), pp. 8–16.
[13] V.V. Dang, N.L. Dodonova, S. Yu. Korabelshchikova, B.F. Melnikov SH-weak duality of semigroups. and minimum semigroup of SH-approximation *University proceedings. Volga region. Physical and mathematical sciences.* 2019. No. 1 (49). P. 29–39 (in Russian).

**Nikolai Krainiukov** received M.Sc.degree in Applied Mathematics and Control from Moscow Institute of Physics and Technology, Moscow, Russia in 1982, and, the Ph.D. degree in Application of mathematical methods, mathematical modeling in scientific research from Samara Aero-Space university, Samara, Russia in 1992. He is currently Associate Professor Faculty of Computational Mathematics and Cybernetics, Shenzhen MSU – BIT University, Shenzhen, Chine. The research activities and interests of Prof. Krainiukov are currently focused on the regular languages, prefix codes, finite automata and networks.

**Boris Melnikov** received Specialist in Applied Mathematics from Lomonosov Moscow State University, Moscow, Russia in 1984, Candidate of sciences (PhD) in Mathematical support and software of computers, computer systems, complexes, and nets from Lomonosov Moscow State University, Moscow, Russia in 1990, and, Doctor of sciences in Mathematical support and software of computers, computer systems, complexes, and nets from Lomonosov Moscow State University, Moscow, Russia in 1997. His research interests include several aspects of heuristic algorithms, discrete optimization and networks.

Open Science Index, Mathematical and Computational Sciences Vol:18, No:8, 2024 publications.waset.org/10013756.pdf