Artificial Intelligence and Human-Computer Interaction
Y. Ye and P. Siarry (Eds.)

doi:10.3233/FAIA240146

On Some Properties of Maximal Prefix Codes and Machine Learning for Automata

Nikolai KRAINIUKOV [a,1], Mikhail ABRAMYAN [a,b], and Boris MELNIKOV [a]
[a] *Shenzhen MSU–BIT University, Shenzhen, China*
[b] *Southern Federal University, Rostov-on-Don, Russian Federation*
ORCiD ID: Nikolai Krainiukov https://orcid.org/0000-0002-3461-3070
ORCiD ID: Mikhail Abramyan https://orcid.org/0000-0002-2802-6144
ORCiD ID: Boris Melnikov https://orcid.org/0000-0002-6765-6800

Abstract. In this paper we study the prefix codes and application of prefix codes for problem of machine learning for deterministic finite state automaton. We give an example for the problem of constructing an inverse morphism, also parameterized by the number of transitions of automata. We investigate the factorization of prefix codes can give more simple structure of DFA for understanding his behaviors. To verify the correctness of the proposed approach, we implemented a system computer algebra *GAP* that accurately performs the logical flow of algorithm cycle by cycle.

Keywords. Finite state automata, Prefix Codes, Inverse morphism, Iterations of finite languages.

1. Introduction

The codes have many applications in theory of formal languages, data processing and data classification. In this article we continue the topics of our previous works [1-5] but we can see the problem of other point of view. We study the problem of identifying (learning) a deterministic finite state automaton (DFA). The application codes are based on the representation of the codes as a sequence of words in specific order. Problems in grammatical inference [6] is to find a (non-unique) smallest DFA that is consistent with a set of given positive S^+ and negative S^- examples of the finite set of words. The smallest size DFA is defined by the amount of states it contains.

2. Basic definition: words, codes and automata

In this section, we give the necessary notions related to words, codes and finite automata and provide an overview of existing machine learning methods for building finite automata.

Let consider a finite set of letters $\Sigma = \{a, b, c \ldots\}$ which we call an alphabet Σ. A word or string w is finite length sequence of letters over alphabet Σ. The set Σ^* is the set

[1] Corresponding Author: Nikolai Krainiukov, Faculty of Computational Mathematics and Cybernetics, Shenzhen MSU–BIT University, Main Building office 331, No. 1, International University Park Road, Dayun New Town, Longgang District, Shenzhen, Guangdong Province, China. E-mail: n.krainiukov@smbu.edu.cn.

of all finite words over Σ with the concatenation operation. Free monoid Σ^* contains the empty word ε. Semigroup $\Sigma^+ = \Sigma^* \setminus \varepsilon$ is monoid Σ^* without empty word .

The formal language L is the subset $L \subset \Sigma^*$ of monoid Σ^*. The words $w, v \in L$ are an element of Σ^*, $|w|$ is the length of word w and if the language L is finite language, then $|L|$ is the number of words in language L. A basic operation of formal languages is concatenation of two words $w = uv$. The concatenation can be expanded to the formal languages $= L1 \cdot L2$.

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 . A code X never contains the empty word ε, because in this case 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.

In order to determine whether the set X is a code, there are criteria characterizing this property. One of these properties [6-7] is consisted in following:

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

An injective inverse morphism preserves the property of being a code [7-8]. Let $\beta : \Delta^* \to \Sigma^*$ be an injective morphism. If X is a code over Δ^*, then $\beta(X)$ is a code over Σ^*. If Y is a code over Σ^*, then $\beta^{-1}(Y)$ is a code over Δ^*.

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's 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 draw at Figure 1 as node 1. The root of tree presents empty word ε. Each node of the tree represents a word w in $\sum^*$.

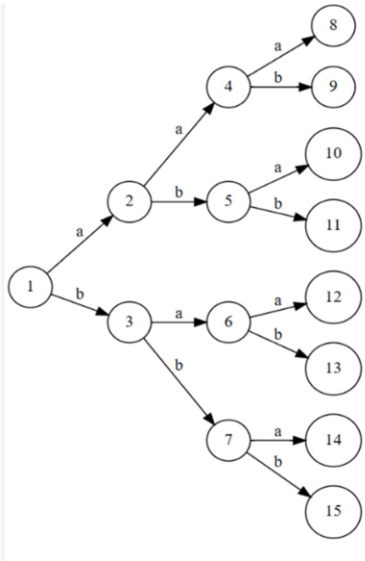

Figure 1. Infinite tree present of monoid $\sum^*$.

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. That is for all words , $v \in X$, 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. Each word of the code $u \in X$ represents a path from the root of the tree marked with letters on the branches of this tree.

The prefix code X is called 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 maximal prefix code because we added words $\{aa, ab, abb,\}$ to code. These words are no comparable by relation $\leq$ with words $\{bb,\ aba\}$.

We will call the code both a previously defined morphism $\beta : \Delta^* \to \Sigma^*$ and the corresponding finite language X. At the same time, in the case of injective of this morphism β, according to [9], we should call the code (as a set, a language) as well as coding (as a process) – unambiguous.

A finite state automaton is a well-known model that can be used to recognize a regular language [10].

An automaton A over alphabet Σ consists of a set of states Q , the initial states $I \subset Q$, the terminal states $T \subset Q$, and a set $F \subset Q \times A \times Q$ called the set of edges.

Here Q and Σ are finite sets called the state set and, respectively, the input alphabet of automaton A.

The automaton is denoted by $A = (Q, I, T)$. The DFA is said to be deterministic automaton if F is a function (may be partial function on $Q \times A$) from $F: Q \times A \to Q$ and to be complete if F is a total function from $F: Q \times A \to Q$.

The automaton is finite when the set Q is finite.

An automaton $A = (Q, I, T)$ over Σ is unambiguous if for all $p, q \in Q$ and $w \in \sum^*$, there is at most one path from p to q with label w in automaton A.

Figure 2 shows the automaton A with five states, the set of initial states $I = \{1\}$, the set of terminal states $T = \{1\}$, the set of edges $T = \{(1, a, 3), (1, b, 2), (3, b, 4), (2, b, 1), (2, a, 5)\}$. This automaton A defines the regular language $= (bb)^*$.

An algorithm is a machine learning algorithm if it improves its behavior as experience accumulates. This means that the algorithm adjusts the parameters of the model either on pre-prepared test cases or on its own mistakes, and over time solves the task better and better. Some machine learning algorithms are capable of noticing previously unknown patterns in data, highlighting knowledge that did not exist before.

Usually, in machine learning, finite automata are considered as recognizers of regular languages or as converters of words of one regular language into words of another language. The class of languages recognized by DFA is called regular languages. It is known that it coincides with the class of languages described by regular expressions and automatic grammars

The one of the old problems in grammatical inference is the problem of learning a deterministic finite state automaton [11]. The problem of DFA identification is to find a may be non-unique smallest DFA that is consistent with a set of given positive $S^+ = \{ v_1,\ v_2 \ldots\ v_n\}$ and negative $S^- = \{ w_1,\ w_2, \ldots\ w_m \}$ examples of the finite set of words. The size of a DFA is measured by the amount of states it contains. An identified DFA has to be as small as possible because the simplest is to be preferred in practical usage.

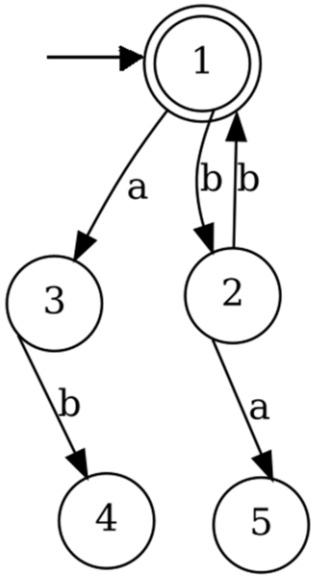

Figure 2. The automaton A with five states.

The problem of constructing an automaton that recognizes a certain language from a variety of examples is relevant. The task can be strengthened to build an automaton with a minimum number of states.

However, in [6] it is shown that this problem is NP-hard.

3. Factorization of codes and constructing an automaton

This section we discuss application codes to construct an automaton that recognizes a language from a variety of examples.

The automaton recognizes those words formed by the labels of transitions on paths from a specific start state to a final state. We use $L(A)$ to denote the language of a DFA A

Given a pair of finite sets of positive sample strings $S^+ = \{ v_1, v_2 \dots v_n\}$ and negative sample strings $S^- = \{ w_1, w_2, \dots w_m \}$ called the input sample. The problem of identification DFA is to find a smallest DFA A that is consistent with $S = \{S^+, S^-\}$. That means $S^+ \subseteq L(A)$ and $S^- \subseteq \Sigma^* \backslash L(A)$.

We use input sample S for construct the so-called prefix tree for sample.

How can you store a lot of strings S at all? Firstly, in the form of a list. If the list contains n +m strings of length less than $\max\{ |v_1|, |v_2| \dots |v_n|, \dots, |w_1|, |w_2|, \dots |w_m| \}$.

Using prefix relation $\leq$ order on a set of strings, prefix trees can be specified. Figure 3 shows the prefix tree for input sample $S^+ = \{abaa, abbb\}$ and $S^- = \{aa, aba\}$.

Each node corresponds to some string that is stored implicitly, in the form of a sequence of edge on the way to it. Also, each node may store one bit of information that determines whether this word belongs to the input sample S for positive sample S^+ or for negative sample S^-. Any additional values associated with this word can be stored in some labels. The root corresponds to an empty word ε. If a node corresponds to the string

w, and the edge coming from it is marked with the symbol $a \in \Sigma$, then this edge goes to the node corresponding to the string wa.

The node 1 is a start state; the nodes 5, 7 are the terminal nodes for positive example S^+ ; the nodes 8, 9 are the terminal nodes for negative example S^- . The prefix tree has reserved less memory than the list, repeated prefixes of words allow to save memory for the presented input sample S. The prefix tree is a schema of automaton A which is called tree-shaped DFA. The other name of automaton A is an augmented prefix tree acceptor. This tree-shaped DFA contended states that can be divided to same different kind of state. A DFA is called augmented because it contains (is augmented with) states for which it is yet unknown whether they are accepting or rejecting. These states are 4 and 9 on Figure 3.

Next it applied a state-merging algorithm to first construct a tree-shaped DFA A from this input, and then to minimized size of DFA A to merge the states of DFA. The authors wrote in [12] "A merge of two states q and q' combines the states into one: it creates a new state q'' that has the same incoming and outgoing transitions of both q and q'. Such a merge is only allowed if the states are consistent, i.e., it is not the case that q is accepting while q' is rejecting, or vice versa."

Now we suppose that input sample S^+ and S^- are the prefix codes. Tree-shaped DFA A from this input S will be unambiguity automaton because S^+ and S^- are the prefix codes. In these cases there are the maximal prefix code C^+ which include the positive example $S^+ \subset C^+$ and corresponded maximal prefix code C^- which include the negative example $S^- \subset C^-$. This imply that there are inverse injective morphisms $\beta_1^{-1}: \Sigma^* \to \Delta_1^*$ and $\beta_2^{-1}: \Sigma^* \to \Delta_2^*$ (Δ_1 , Δ_2 - corresponding alphabets). The letters of alphabets Δ_1 , Δ_2 - corresponding are coding the words of maximal prefix codes C^+ and C^- . We can choice the injective morphisms β_1 , β_2 that they are coding each letter of alphabets Δ_1 , Δ_2 the corresponded words of codes C^+ and C^-.

Suppose that codes $S^+ = X1 \cdot X2$ and $S^- = Y1 \cdot Y2$. According to theorem of prefix codes that the set of all prefix code over Σ^* is formed the free monoid [7-8]. We can factorization prefix code to the prime codes.

For prefix codes on Figure 3 the factories codes are $S^+ = \{ab\} \cdot \{aa, bb\}$ and $S^- = \{a\} \cdot \{a, ba\}$ i.e. $X1 = \{ab\}$, $X2 = \{aa, bb\}$, $Y1 \cdot = \{a\}$ and $Y2 = \{a, ba\}$

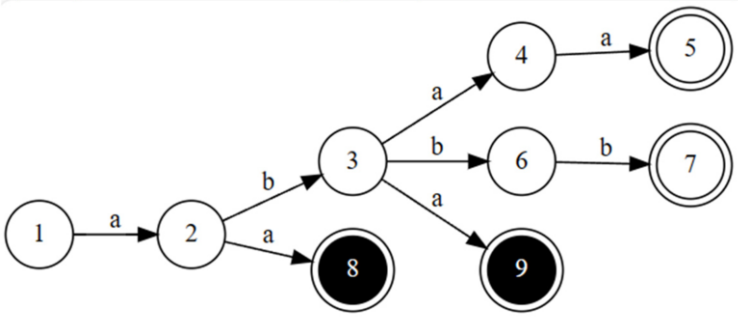

Figure 3. Prefix tree for $S^+ = \{abaa, abbb\}$ and $S^- = \{aa, aba\}$.

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

Choose a word w_{min} of minimum length from $X = \{w_1, w_2, \dots, w_l\}$ and words $A_1 = \{w_{11}, w_{12}, \dots, w_{1k}\}$ from code X starting with the same (first, second and so on) letters as the word w_{min} of minimum length. After this we can find the second factors

of code . 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.

Factorization of prefix codes and inverse morphism using encoding of letters of another alphabet can make it possible to simplify the structure of the automaton and reduce the number of its states. Combining the use of forward and reverse morphisms for prefix codes and encoding using a new/different alphabet makes it possible to flexibly select the number of states of the automaton and letters in the alphabet. There is an additional possibility to choose between the complexity of the automaton (the number of states) and coding (forward and reverse morphism)

An example of this approach is shown in Figures 4 and 5 for $S^+ = \{abaa, abbb\}$ and $S^- = \{aa, aba\}$. The constructed automata have code words at the transitions between states. The automaton obtained after factorization of codes has the structure of a simple product of code words on transitions between states.

In this paper, a method for identifying (learning) a DFA is proposed for finite automata with factorization of prefix codes. This method is based on reducing the specified problem of the coding (forward and reverse morphism).

The efficiency of the method has been tested on both manually generated and randomly generated examples.

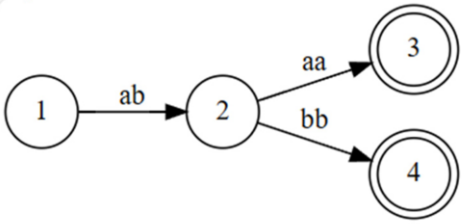

Figure 4. Automaton for $S^+ = \{abaa, abbb\}$ with code's words $\{ab\} \cdot \{aa, bb\}$.

4. Conclusion

The results of the experiments allow us to speak about the possibility of the proposed method in comparison with existing methods for solving the problem under consideration.

We implemented a system computer algebra *GAP* that accurately performs the logical flow of algorithm cycle by cycle [13].

This part of package "automata" to make it useful to finite automata (deterministic and non-deterministic) and formal languages theorists, since monoids are already implemented in GAP and we can take advantage of this fact.

The results of the present paper suggest several directions for further research. Here we briefly outline two such directions: the problem about optimal choice inverse morphisms [14] and algorithm of factorization of codes, not only prefix codes.

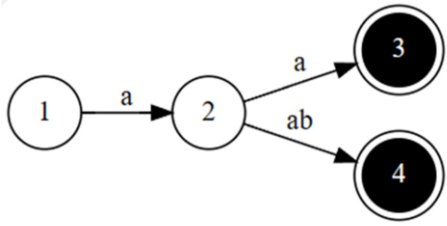

Figure 5. Automaton for $S^{-} = \{aa, aba\}$ with code's words $\{a\} \cdot \{a, ba\}$.

5. Acknowledgments

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

References

[1] Melnikov B, Korabelshchikova S, Dolgov V. "On the task of extracting the root from the language" International Journal of Open Information Technologies. 2019:7(3):1–6 (in Russian).
[2] Melnikov B, "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. 2022:10(4): 1–9 (in Russian).
[3] Alekseeva A, Melnikov B. "Iterations of finite and infinite languages and nondeterministic finite automata" Vector of Science of Togliatti State University, 2011:3 (17):30–33 (in Russian).
[4] Melnikov B, Melnikova A. "Infinite trees in the algorithm for checking the equivalence condition of iterations of finite languages. Part I "International Journal of Open Information Technologies. 2021:9(4):1–11 (in Russian).
[5] Melnikov B, Melnikova A. "Infinite trees in the algorithm for checking the equivalence condition of iterations of finite languages. Part II "International Journal of Open Information Technologies. 2021:9(5):1–11. (in Russian).
[6] Gold E. "Complexity of automaton identification from given data." Information and Control 1978:37(3): 302–320.
[7] Lallement G. Semigroups and Combinatorial Applications. Berkeley, NJ, Wiley & Sons, Inc., 1979. p.453.
[8] Berstel J, Perrin D. Theory of Codes, Academic Press, New York, 1985. p.378.
[9] Yablonskiy S. Introduction to Discrete Mathematics. Study Guide for Universities. Moscow: Vysshaya Shkola. 2001. 384 p. (in Rusian).
[10] Brauer W. Introduction in the Finite Automata Theory. Moscow: Radio I Svyaz Publ.; 1987. 390 p. (in Russian).
[11] de la Higuera C. "A bibliographical study of grammatical inference." Pattern Recognition, 2005:38(9): 1332–1348.
[12] Heule M, Verwer S. "Using a Satisfiability Solver to Identify Deterministic Finite State Automata." BNAIC 2009, pp. 91-98.
[13] von zur Gathen J and Gerhard J. Modern computer algebra, Cambridge University Press (2003).
[14] Rotman J. Advanced Modern Algebra, Englewood Cliffs, NJ: Prentice Hall, 2002, p.298.