RAIRO-Theor. Inf. Appl. 43 (2009) 269–279
DOI: 10.1051/ita:2008028

Available online at:
www.rairo-ita.org

POLYNOMIAL LANGUAGES WITH FINITE ANTIDICTIONARIES*

Arseny M. Shur[1]

Abstract. We tackle the problem of studying which kind of functions can occur as complexity functions of formal languages of a certain type. We prove that an important narrow subclass of rational languages contains languages of polynomial complexity of any integer degree over any non-trivial alphabet.

Mathematics Subject Classification. 68Q45, 68R15.

The combinatorial complexity of a formal language (simply *complexity* throughout this paper) is a function that measures the diversity of the language. The most well-known and intensively studied particular case of complexity is the subword complexity of an infinite word (see Sect. 9 of [2], for example). Also, some attention is drawn to complexity of languages of power-free words, starting with [1], where some bounds for the complexity of the language of binary cube-free words were given. However, the complexity is an important characteristics of *any* language. So, there are good reasons to study the complexity of languages in a more general framework, moving from the question "what is the complexity of a given language?" to the question "which complexities can languages from a given class have?". In this paper we continue the study of the last question, initiated in [6], and prove that polynomial complexities of any degree are possible for an interesting and important subclass of rational languages.

1. Preliminaries

We recall some notation and definitions on words, finite automata and complexity functions.

Keywords and phrases. Regular language, finite antidictionary, combinatorial complexity, wed-like automaton.

* *The paper was written when the author stayed at the University of Turku.*

[1] Ural State University, Ekaterinburg, Russia; Arseny.Shur@usu.ru

Article published by EDP Sciences

An *alphabet* Σ is a non-empty set of *letters*. A *word* is a finite sequence of letters, say $W = a_1 \dots a_n$. The symbol λ stands for the empty word. A word U is a *factor* (respectively *prefix*, *suffix*) of the word W if W can be written as PUQ (respectively UQ, PU) for some (possibly empty) words P and Q. A factor (prefix, suffix) of W is called *proper* if it does not coincide with the whole word W. As usual, we write Σ^n for the set of all n-letter words and Σ^* for the set of all words over Σ. The subsets of Σ^* are called *languages*. A language is *factorial* if it is closed under taking factors of its words, and *antifactorial* if no one of its words is a factor of another one.

A *deterministic finite automaton* (DFA) is a 5-tuple $(\Sigma, Q, \delta, s, T)$ consisting of a finite input alphabet Σ, a finite set of states (vertices) Q, a partial transition function $\delta : Q \times \Sigma \to Q$, one initial state s, and a set of terminal states T. The underlying digraph of the automaton contains states as vertices and transitions as directed labeled edges. Then every path in this digraph is labeled by a word, and every cycle is labeled by a *cyclic word*. We make no difference between a DFA and its underlying digraph. A *accepting path* is any path from the initial to a terminal vertex. A DFA *recognizes* the language which is the set of all labels of the accepting paths. The class of such languages coincides with the class of all rational languages (Kleene's theorem). An automaton is called *consistent* if each its vertex is contained in some accepting path. A *trie* is a DFA whose underlying digraph is a tree such that the initial vertex is its root and the set of terminal vertices is the set of all its leaves.

For an arbitrary language L over a finite alphabet Σ the *complexity function* is defined by $C_L(n) = |L \cap \Sigma^n|$. Normally we are interested in the growth rate rather than the precise form of the complexity function. As usual, we call a complexity function *polynomial* if it is $O(n^p)$ for some $p \geq 0$ (bounded from above by a polynomial of degree p), and *exponential* if it is $\Omega(\alpha^n)$ for some $\alpha > 1$ (bounded from below by an exponential function at base α). We also write $\Theta(n^p)$ for the function which is bounded from above and from below by polynomials of degree p.

The definition of Ω (and, hence, of Θ) suits well only for increasing functions. So, if the complexity function is not increasing, we estimate its fastest increasing subsequence. Actually, in this paper we deal only with factorial languages. For a factorial language the complexity is known to be either bounded by a constant or strictly increasing [4]. We also note that the complexity of the language of all finite factors of an infinite word W is well-known in combinatorics of words under the name of *subword complexity* of W.

In general, a language over an alphabet Σ can have an arbitrary complexity function, which does not exceed $|\Sigma|^n$. But if we restrict ourselves to reasonable classes of languages, we can describe possible complexity functions more precisely. The following theorem describes all possible types of complexity functions for rational languages.

Theorem 1.1 [6]**.** *Let a consistent deterministic finite automaton $\mathcal{A}$ recognize the language L. Then*

(1) *If $\mathcal{A}$ is acyclic, then L is finite.*

(2) *If $\mathcal{A}$ contains two cycles sharing one vertex, then L is exponential.*

(3) *If $\mathcal{A}$ contains a cycle, and all cycles in $\mathcal{A}$ are disjoint, then L is polynomial, and its complexity function is $\Theta(n^{m-1})$, where m is the maximum number of cycles connected by an accepting path.*

2. Main results

In this paper we are interested in constructing languages from an important subclass of rational languages, having a given polynomial complexity. In order to introduce this class we need the concept of antidictionary.

A word W is *forbidden* for the language L if it is not a factor of any element of L. A forbidden word is *minimal* if all its proper factors are not forbidden. The set of all minimal forbidden words for L is called the *antidictionary* of L. The antidictionary is always antifactorial. If a factorial language L over the alphabet Σ has the antidictionary AD, then the following equalities holds:

$$L = \Sigma^* \setminus AD \cdot \Sigma^* \cdot AD, \qquad AD = \Sigma \cdot L \cap L \cdot \Sigma \cap \Sigma^* \setminus L.$$

These equalities imply that a factorial language is rational if and only if so is its antidictionary. In particular, the factorial languages with finite antidictionaries form a proper subclass of the class of rational languages. This subclass plays a special role in the investigations on complexity functions of languages. First, for languages with finite antidictionaries there is an effective algorithm, evaluating the complexity function (see, *e.g.*, [5]). Second, this algorithm can be used for estimating the complexity function of an arbitrary factorial language in the following way. If L is a factorial language and AD is its antidictionary, then one can take a finite subset $AD_n = AD \cap (\Sigma \cup \ldots \cup \Sigma^n)$ and evaluate the complexity function of the language L_n with the antidictionary AD_n, getting a reasonable upper bound for the complexity of L. Thus, the very natural question is: *what kind of complexity functions can languages with finite antidictionaries have*?

Many properties of words are stable under all permutations of the alphabet. Any language with this property and closed under permutations is called *symmetric*. For example, the property "to avoid some regularity" (such as to contain no squares, or cubes, or some other patterns) is obviously stable under all such permutations, and defines the symmetric language of "regularity-free" words. A symmetric language surely has the symmetric antidictionary. So, it is natural to consider a restricted version of the above question as well: *what kind of complexity functions can languages with finite symmetric antidictionaries have*?

Here we give a partial answer to both of the above questions. Namely, we show that polynomials of any degree can be obtained as complexities of languages with finite (and even symmetric finite) antidictionaries.

Theorem 2.1. *Let Σ be an alphabet such that $|\Sigma| > 1$, and m be a nonnegative integer. Then there exists a factorial language over Σ with a non-symmetric finite antidictionary and the complexity $\Theta(n^m)$.*

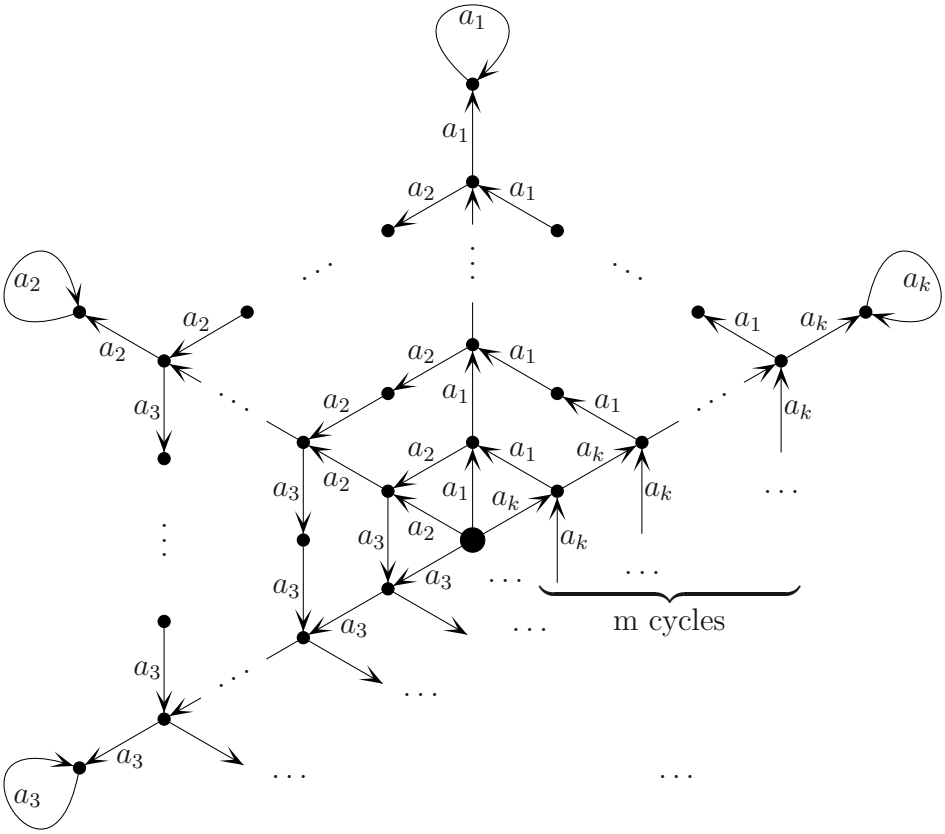

FIGURE 1. The "web-like" automaton $\mathcal{W}_{k,m}$ for the asymmetric language of complexity $\Theta(n^m)$ over a k-letter alphabet. The cyclic order is $a_1 \prec a_2 \prec \ldots \prec a_k \prec a_1$. The bigger circle represents the initial vertex.

Theorem 2.2. *Let Σ be an alphabet such that $|\Sigma| > 1$, and m be a nonnegative integer. Then there exists a factorial language over Σ with a symmetric finite antidictionary and the complexity $\Theta(n^m)$.*

Both of these theorems are proved by constructing an appropriate sequence of examples. An example is an antidictionary; it suffices to build an automaton recognizing the language with this antidictionary and apply Theorem 1.1 to it. The automata for Theorem 2.1 have clear structure and can be drawn in the general case, see Figure 1. On the other hand, there is a strong evidence that any automata for the examples proving Theorem 2.2 should be complicated and hardly can be drawn besides the case in Figure 3. So, for Theorem 2.1 we present only the main constructions: the antidictionaries and the corresponding automata. The reader should be able to recover details after studying the more involved proof

of Theorem 2.2, which is given below in a full extent. As an interesting feature we note that in the simplest case of the two-letter alphabet the examples for both theorems coincide. The existence of a non-symmetric example for the binary case was already shown in [6], Theorem 4.1.

3. Asymmetric case

Proof of Theorem 2.1. Fix a k-letter alphabet Σ and a cyclic order on it. We shall write $\bar{a}$ for the successor of a in this order. The family of antidictionaries $\{AD_m\}$ over Σ is defined as follows:

$$AD_m = \{ab | a, b \in \Sigma, b \neq a, b \neq \bar{a}\} \cup \{a^2\bar{a}\bar{\bar{a}}, a^3\bar{a}^2\bar{\bar{a}}, \ldots, a^m\bar{a}^{m-1}\bar{\bar{a}} | a \in \Sigma\} \cup \{a^{m+1}\bar{a} | a \in \Sigma\}.$$

The language over k-element alphabet Σ with the antidictionary AD_m is recognized by the "web-like" automaton $\mathcal{W}_{k,m}$, which can be built in the following steps (see Fig. 1).

1. Use the same k-letter alphabet Σ and the same cyclic order on it as for constructing AD_m.
2. Draw the initial vertex and k vertices with loops. Label loops with distinct letters.
3. Add vertices and edges to connect the initial vertex to each of the loops by a separate path of $m+1$ edges, labeled by the same letter as in the loop. These paths will be called *rays*, and the added vertices will be called *intermediate*.
4. Add vertices and edges to provide mk separate paths, connecting the intermediate vertices from different rays, as follows. If such a vertex is on the ray with labels a on the distance i from the initial vertex, then the path starting in this vertex consists of i edges labeled by $\bar{a}$ and leads to the vertex which is on the distance i from the initial vertex on the ray labeled by $\bar{a}$.

The language recognized by the automaton $\mathcal{W}_{k,m}$ has the complexity $\Theta(n^m)$ by statement (3) of Theorem 1.1.

4. Symmetric case

To prove Theorem 2.2 we shall construct an automaton recognizing a language with a known finite antidictionary. For this purpose we use the algorithm of [3] which is shortly described here in our terminology.

Algorithm 1.
Input: an antidictionary AD.
Output: a DFA $\mathcal{A}$ recognizing the factorial language L with the antidictionary AD.
Step 1. Construct a trie $\mathcal{T}$, recognizing AD. ($\mathcal{T}$ is actually the digraph of the prefix order on the set of all prefixes of AD.)

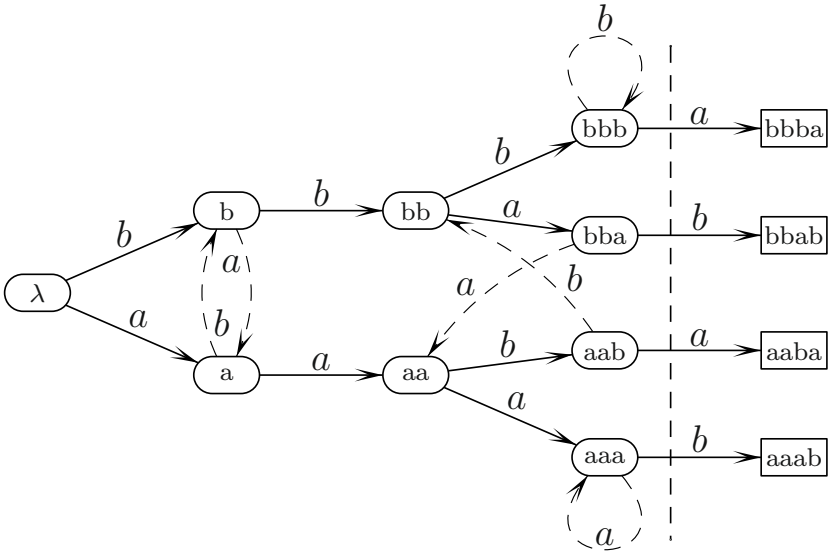

FIGURE 2. Building the automaton $\mathcal{W}_{2,2}$. The forward edges are drawn by ordinary lines, while the backward ones are represented by dash lines. The initial vertex is λ. The vertices to the right of the vertical line are deleted on step 4 of Algorithm 1.

Step 2. Associate each vertex in $\mathcal{T}$ with the word labeling the accepting path ending in this vertex. (Now the set of vertices is the set of all prefixes of AD.)
Step 3. Add all possible edges to $\mathcal{T}$, following the rule:
the edge (U, V) labeled by the letter a should be added if
U is not terminal, and
U has no outgoing edge labeled by a, and
V is the longest suffix of Ua which is a vertex of $\mathcal{T}$.
(These edges are called *backward* while the edges of the trie are called *forward*.)
Step 4. Remove all terminal vertices and mark all remaining vertices as terminal to get $\mathcal{A}$.

To illustrate the work of Algorithm 1, we give an example. Let us build an automaton recognizing the binary factorial language with the antidictionary $AD_2 = \{aaba, bbab, aaab, bbba\}$ (Fig. 2). On first two steps, the trie $\mathcal{T}$ is built (its edges are drawn by ordinary lines, its terminal vertices are rectangles), and all vertices are labeled with words. Then on step 3 the backward edges (drawn by dashed lines) are added. For example, a backward edge from the vertex aab labeled by b leads to the vertex bb, since bb is the longest suffix of $aabb$ which is a vertex of $\mathcal{T}$. Finally, the "rectangle" vertices are removed on step 4. As a result, we obtain the automaton $\mathcal{W}_{2,2}$.

Some useful properties of the automaton $\mathcal{A}$ are collected in the following lemma.

Lemma 4.1.

(1) *The automaton $\mathcal{A}$ is deterministic and consistent.*
(2) *The set of vertices of $\mathcal{A}$ coincides with the set of all proper prefixes of the words from AD.*
(3) *The accepting paths in $\mathcal{A}$ are exactly the paths starting in the initial vertex.*
(4) *If the word Wa is forbidden for some vertex W of $\mathcal{A}$ and some letter a, then no outgoing edge from W labeled by a exists.*

Proof.

(1) The automaton $\mathcal{A}$ is deterministic by construction and consistent since all its vertices are terminal and can be achieved from the initial vertex by the edges of the trie.
(2) The set of vertices of the trie $\mathcal{T}$ is the set of all prefixes of AD. The terminal vertices of the trie are its leaves, and then are labeled by the words of AD. When we remove these vertices in step 4, only the vertices labeled by the proper prefixes remain.
(3) It is obvious, because all vertices of $\mathcal{A}$ are terminal.
(4) After step 3 the transition function of the current automaton is well-defined, so we surely have an outgoing edge from W labeled by a. We prove that this edge leads to a terminal vertex, and hence, is removed at step 4. If this edge is forward, then it leads to the vertex Wa of the trie. This vertex is a prefix of the word from antidictionary and a forbidden word simultaneously. Thus, it belongs to AD and therefore is a terminal vertex of the trie. Suppose that this edge is backward. W itself is not forbidden by (2), while Wa is. Then the word Wa has a forbidden suffix. Its minimal forbidden suffix U belongs to AD and, at the same time, U is its longest suffix which is a vertex of the trie. Hence the considered edge leads to U, and U is terminal. The statement is proved. □

Proof of Theorem 2.2. We fix a k-letter alphabet Σ and consider a family of symmetric finite antidictionaries $\{AD_m\}$, where AD_m is the minimal symmetric language containing the following set of words (all a_i's are supposed to be different elements of Σ):

$$
\begin{array}{lllll}
\{ & a_1^{m+1}a_2, & & & \\
 & a_1^m a_2^m a_1, & a_1^m a_2^m a_3^m a_1, & \ldots, & a_1^m a_2^m \ldots a_{k-1}^m a_1, \\
 & a_1^m a_2^{m-1} a_3, & & & \\
 & a_1^{m-1} a_2^{m-1} a_1, & a_1^{m-1} a_2^{m-1} a_3^{m-1} a_1, & \ldots, & a_1^{m-1} a_2^{m-1} \ldots a_{k-1}^{m-1} a_1, \\
 & a_1^{m-1} a_2^{m-2} a_3, & & & \\
 & \ldots & & & \\
 & a_1^2 a_2 a_3, & & & \\
 & a_1 a_2 a_1, & a_1 a_2 a_3 a_1, & \ldots, & a_1 a_2 \ldots a_{k-1} a_1 \qquad \}.
\end{array}
\tag{4.1}
$$

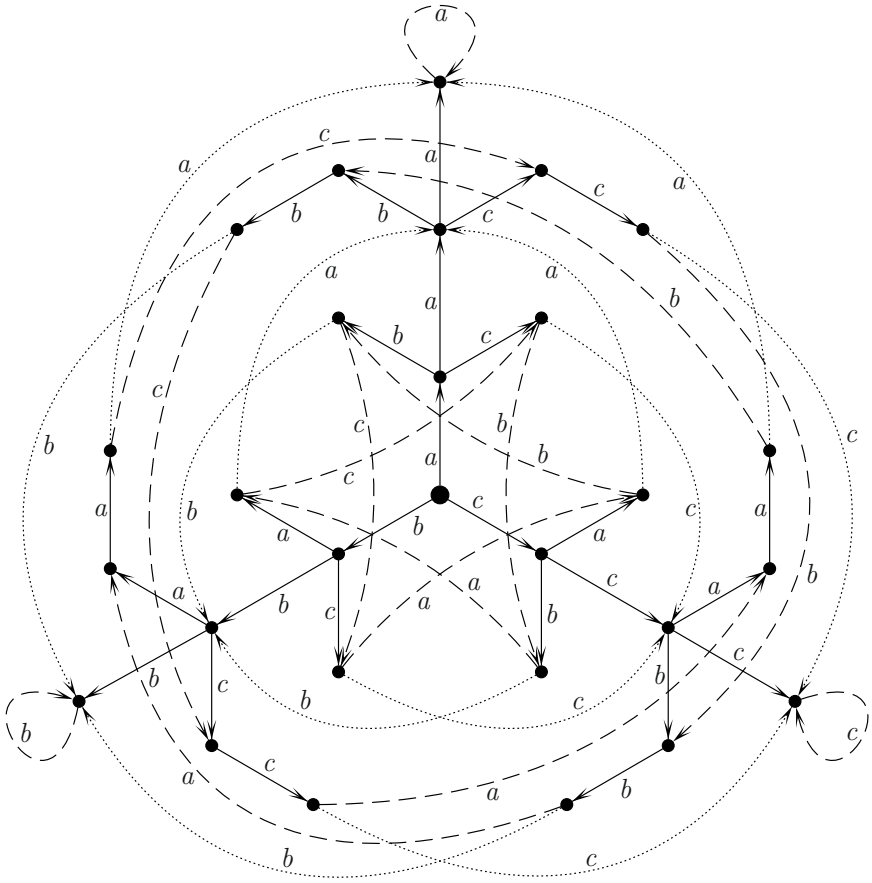

FIGURE 3. The automaton for the quadratic symmetric language over the 3-letter alphabet. The forward edges are drawn by ordinary lines, while the backward ones are represented by dash or dotted lines depending on whether they belong to a cycle or not. The bigger circle represents the initial vertex.

So, to get AD_m from this set one should apply to all its elements all possible permutations of Σ.

Our goal is to prove that the factorial language L_m with the antidictionary AD_m has the complexity $\Theta(n^m)$. We apply Algorithm 1 to AD_m to obtain the automaton $\mathcal{A}_m$ which satisfies the basic condition of Theorem 1.1 by statement (1) of Lemma 4.1. Such an automaton for the case $k = 3$, $m = 2$ is shown in Figure 3 (this is the most representative example which can be drawn in one page). We would appeal to Figure 3 to comment our considerations. To prove the required statement we show that $\mathcal{A}_m$ satisfies the condition (3) of Theorem 1.1, and the maximum number of cycles connected by a accepting path is $m+1$.

We use statement (2) of Lemma 4.1 and examine (1) to derive immediately the following important observation:

(i) A word W is a vertex in $\mathcal{A}_m$ if and only if $W = a^{m+1}$ for some $a \in \Sigma$ or $W = a_1^s \dots a_{r-1}^s a_r^t$, where $1 \le r \le k$, $a_i \in \Sigma$, $a_i \ne a_j$ for $i \ne j$, and $0 < t \le s \le m$.

Now we proceed in two steps. On the first step we define and analyze a family of cycles in $\mathcal{A}_m$, while the main result of the second step is that no other cycles occur in this automaton. After these steps a few remarks conclude the proof.

Step 1. First we show that for any cyclic ordering $a_1 \prec a_2 \prec \dots \prec a_k \prec a_1$ of Σ there exists a cycle labeled by the cyclic word $a_1 a_2 \dots a_k$ on the vertices $a_2 a_3 \dots a_k$, $a_3 \dots a_k a_1$, …, $a_1 a_2 \dots a_{k-1}$ (by (i) all these words are actually vertices). Indeed, since the word $a_1 a_2 \dots a_k$ is not a vertex, the outgoing edge from $a_1 a_2 \dots a_{k-1}$ labeled by a_k is backward one and leads to $a_2 a_3 \dots a_k$. Similarly, the outgoing edge from $a_2 a_3 \dots a_k$ labeled by a_1 leads to $a_3 \dots a_k a_1$, and so on. Finally, k such backward edges constitute the required cycle. We also note that each of the mentioned vertices determines a unique cyclic ordering. So, all $k!$ such vertices form $(k-1)!$ disjoint cycles of length k, one for each ordering. In the following we refer to them as to *level one* vertices and *level one* cycles. In Figure 3 one can see two level one cycles, labeled by the cyclic words *abc* (counterclockwise) and *acb* (clockwise).

Now we introduce *level s* cycles for any $s \le m$. For arbitrary cyclic ordering $a_1 \prec a_2 \prec \dots \prec a_k \prec a_1$ of Σ such a cycle is labeled by the cyclic word $a_1^s a_2^s \dots a_k^s$ of length ks and consists of the vertices

$$\begin{array}{llll} a_2^s \dots a_{k-1}^s a_k, & \dots, & a_2^s \dots a_{k-1}^s a_k^s, \\ a_3^s \dots a_k^s a_1, & \dots, & a_3^s \dots a_k^s a_1^s, \\ \dots, & & \\ a_1^s \dots a_{k-2}^s a_{k-1}, & \dots, & a_1^s \dots a_{k-2}^s a_{k-1}^s, \end{array} \tag{4.2}$$

called *level s* vertices. To check the existence of this cycle, first note that for any j, $1 \le j < s$, the vertices $a_1^s \dots a_{k-2}^s a_{k-1}^j$ and $a_1^s \dots a_{k-2}^s a_{k-1}^{j+1}$ are connected by a forward edge. Second, the outgoing edge from $a_1^s \dots a_{k-2}^s a_{k-1}^s$ labeled by a_k is backward and leads to the vertex $a_2^s \dots a_{k-1}^s a_k$. All the above holds true for all vertices from other rows of (2). Thus, we get the required cycle. Once again, any level s vertex determines a unique cyclic ordering, and therefore belongs to a unique level s cycle. On the other hand, from the list (2) it is clear that level s vertex can not belong to level t cycle for $s \ne t$. Thus, we have that all introduced cycles are disjoint. See Figure 3 for two level 2 cycles labeled by *aabbcc* (counterclockwise) and *aaccbb* (clockwise).

At last, we define level $m{+}1$ cycles. For arbitrary $a \in \Sigma$ the vertex a^{m+1} has only one outgoing edge, and this edge is labeled by a also, since all words $a^{m+1}b$ for $b \ne a$ are forbidden, so that no more outgoing edges exist by statement 4) of Lemma 4.1. This edge obviously constitutes a loop; so, level $m{+}1$ cycles are just these loops. They do not intersect other involved cycles. Thus, all defined cycles are disjoint and the step is completed.

Step 2. We verify that any vertex of $\mathcal{A}_m$ is a prefix of a level s vertex for some s, $1 \le s \le m{+}1$. Using statement (2) of Lemma 4.1, we just check the list (1). It contains $2m$ rows, and we see that

- proper prefixes of the word in the first row are prefixes of the level $m{+}1$ vertex a_1^{m+1};
- proper prefixes of the words in the second and the third rows are prefixes of the level m vertex $a_1^m \dots a_{k-1}^m$;
- proper prefixes of the words in the next two rows are prefixes of the level $m{-}1$ vertex $a_1^{m-1} \dots a_{k-1}^{m-1}$;
- $\dots$
- proper prefixes of the words in the last row are prefixes of the level 1 vertex $a_1 \dots a_{k-1}$.

We say that a vertex is of *prefix level* s, if it is a prefix of some level s vertex but not that of any level $s{-}1$ vertices. In Figure 3 the vertices λ, a, b, c together with six level 1 vertices are of prefix level 1, the vertices aa, bb, cc together with twelve level 2 vertices are of prefix level 2, and three level 3 vertices are of prefix level 3.

Now we prove that any backward edge, which starts at a vertex of prefix level s and does not belong to a level s cycle, ends in some vertex of prefix level $s{+}1$. Using (i) we check all possibilities for the starting vertex of an edge. The case $W = a^{m+1}$ is trivial, because this vertex has a unique outgoing edge which constitutes a level $m{+}1$ cycle. Let $W = a_1^s \dots a_{r-1}^s a_r^t$. A few subcases are possible.

1. $r = 1$. All k outgoing edges are forward ones.
2. $r > 1$ and $t < s$. The outgoing edge labeled by a_r is forward one, while no other outgoing edges exist by statement (4) of Lemma 4.1, because the word $a_1^s \dots a_{r-1}^s a_r^t a$ is forbidden for any $a \ne a_r$.
3. $1 < r < k$ and $t = s$. The word $a_1^s \dots a_{r-1}^s a_r^s a$ is forbidden if $a \in \{a_1 \dots, a_{r-1}\}$, and we have no edge in this case by statement (4) of Lemma 4.1. Furthermore, this word is a vertex if $a \notin \{a_1, \dots, a_r\}$, and we have a forward edge for each of such letters. Finally, the edge labeled by a_r is backward one and leads to a_r^{s+1}.
4. $r = k$ and $t = s$. Here the outgoing edges are labeled with a_1 and a_k (otherwise we get a forbidden word and apply statement (4) of Lemma 4.1. Both edges are backward ones. The edge labeled with a_1 belongs to a level s cycle, while the other edge leads to a_r^{s+1}, as in the previous case.

We see that in all subcases the backward edges which do not belong to level s cycles ends in the vertex a_r^{s+1}, which is obviously of prefix level $s{+}1$.

Now we are able to show that all cycles in $\mathcal{A}_m$ are exhausted by the level s cycles for $1 \le s \le m{+}1$. Note that a forward edge can not decrease the prefix level of a vertex. The backward edges from level s cycles retain the prefix level, while the other backward edges strictly increase it, as we already proved. It means that cycles can not contain these "other" backward edges, because no edge can decrease the prefix level back. But any cycle contains a backward edge, because all forward edges are that of a trie. Therefore, any cycle contains a backward edge of some level s cycle. So we try to get a cycle starting with such an edge. We omit the

trivial case of level $m{+}1$ cycle, so let this edge connect some vertex $a_1^s a_2^s \dots a_k^s$ to the vertex $a_2^s \dots a_k^s a_1$. As was studied above in subcase 2, the vertices $a_2^s \dots a_k^s a_1^r$, $1 \le r < s$, have only one outgoing edge. Hence we inevitably come to the vertex $a_2^s \dots a_k^s a_1^k$. As was studied in subcase 4, here we have two outgoing edges, one of them cannot belong to any cycle, while the other one continues our level s cycle. Repeating this argument k times, we get exactly the level s cycle, as required. We also note that leaving level s cycle one has to move to a prefix level $s{+}1$ vertex. This means that only one level s cycle can be contained in a path. The step 2 is finished.

Now we can conclude the proof. Since there are no cycles in $\mathcal{A}_m$ other than level s cycles for $1 \le s \le m{+}1$, and all such cycles are disjoint, we obtain that the language L_m is polynomial by Theorem 1.1. Since level 1 cycles can be achieved from the initial state, and after some level s cycle a path can meet a cycle of level at least $s{+}1$, we have that a maximum number of cycles along one path is $m{+}1$. The statement of the theorem now follows from Theorem 1.1. □

Acknowledgements. The author is grateful to O. Karyakina for the idea of the web-like automaton. Special thanks to J. Karhumäki and to the referee for valuable remarks on the paper.

References

[1] F.-J. Brandenburg, Uniformly growing k-th power free homomorphisms. *Theoret. Comput. Sci.* **23** (1983) 69–82.
[2] C. Choffrut and J. Karhumäki, Combinatorics of words, in *Handbook of formal languages*, Vol. 1, Chap. 6, edited by G. Rosenberg, A. Salomaa. Springer, Berlin (1997), 329–438.
[3] M. Crochemore, F. Mignosi and A. Restivo, Automata and forbidden words. *Inform. Process. Lett.* **67** (1998) 111–117.
[4] A. Ehrenfeucht and G. Rozenberg, On subword complexities of homomorphic images of languages. *RAIRO-Theor. Inf. Appl.* **16** (1982) 303–316.
[5] Y. Kobayashi, Repetition-free words. *Theoret. Comput. Sci.* **44** (1986) 175–197.
[6] A.M. Shur, Combinatorial complexity of rational languages. *Discr. Anal. Oper. Res., Ser. 1* **12** (2005) 78–99 (in Russian).

Communicated by J. Karhumäki.
Received January 20, 2006. Accepted October 14, 2008.