Languages with Finite Antidictionaries: Growth Rates and Graph Properties
Languages with Finite Antidictionaries: Growth Rates and Graph Properties

Arseny M. Shur
Ural State University
Ekaterinburg, Russia
Arseny.Shur@usu.ru

Supported by Federal Science and Innovation Agency of Russia under the grants RI-111.0/002/075 and 2227.2003.01,
by Russian Foundation for Basic Research under the grant 05-01-00540,
and by Federal Education Agency of Russia under the grant 49123.
The paper was written when the author stayed at the University of Turku.
Abstract

The factorial languages with finite antidictionaries and exponential complexity functions are studied, together with their recognizing automata. Improvements are added to the algorithm evaluating the growth rate. Some graph properties of recognizing automata are proved, and binary languages with small recognizing automata are classified.

Keywords: Combinatorial complexity, growth rate, factorial languages, forbidden words, antidictionaries, finite automata

TUCS Laboratory
Discrete Mathematics For Information Technology Laboratory
The study of possible complexity functions of languages with finite antidictionaries was initiated in [10], where polynomials of any integer degree were shown to be such functions. In this paper we continue this study, considering only exponential functions. In the general form, the question for exponential functions can be stated as follows: what growth rates can languages with finite antidictionaries have?

After necessary notation and definitions, we recall some basic results on complexity of languages. Then we establish some graph properties of finite automata, recognizing the languages with finite antidictionaries. We extract a subgraph of such an automaton, called C-graph, which is responsible for the growth rate of the given language. We also introduce a reduction procedure for finite antidictionaries and show that all possible growth rates are achieved by the languages with reduced antidictionaries.

In the case of the binary alphabet C-graphs of reduced antidictionaries are studied in more details. This study allows us to list all possible C-graphs with at most four vertices up to an isomorphism, and get corresponding growth rates. We also briefly discuss some properties of C-graphs, concerning graph spectra and divisors, because these objects are closely connected to the growth rate.

The paper is organized as follows. The first section contains definitions and basic results on complexity functions, including an algorithm for the growth rate evaluating. The second section is devoted to a known algorithm for constructing an automaton recognizing the given language with finite antidictionary, and to graph properties of the obtained automata. In section 3 C-graphs are introduced and two examples of calculating growth rates for series of languages are given. The section 4 is devoted to reduced antidictionaries and their C-graphs. The last section contains the theorem about small binary C-graphs and some concluding remarks.

1. Preliminaries

An alphabet $\Sigma$ is a non-empty set, elements of which are called letters. A word is a finite sequence of letters, say $W = a_1 \ldots a_n$. The length of $W$ is denoted by $|W|$. The symbol $\lambda$ 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 $\Sigma^n$ for the set of all $n$-letter words and $\Sigma^*$ for the set of all words over $\Sigma$. The subsets of $\Sigma^*$ 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.

Two words $U$ and $W$ are said to be conjugates, if $U = PQ$, $W = QP$ for some words $P$ and $Q$. Conjugacy is an equivalence relation on $\Sigma^*$. We often refer to a conjugacy class as to cyclic word, that is, the word in which the
first and the last letter are considered to be adjacent, and no initial point is specified. All words from this class are obtained by specifying the initial points.

The word $W = a_1 \ldots a_n$ has period $p > 0$, if $a_i = a_{i+p}$ for all $i = 1, \ldots, |W| - p$. We write $U^n$ for the concatenation of $n$ copies of the word $U$. A word $W$ is called primitive, if $W = U^n$ implies $n = 1$. Conjugates have the same periods, whence we can speak about primitive cyclic words also.

A deterministic finite automaton (DFA) is a 5-tuple $(\Sigma, Q, \delta, s, T)$ consisting of a finite input alphabet $\Sigma$, 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. When we consider the label of a cycle as an ordinary word, we specify the vertex to start with. We make no difference between a DFA and its underlying digraph. A reading 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 reading paths, and is consistent if each its vertex is contained in some reading 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.

Since all graphs we deal with in this paper, are directed, we write “graph” instead of “digraph”.

For an arbitrary language $L$ over a finite alphabet $\Sigma$ the complexity function is defined by $C_L(n) = |L \cap \Sigma^n|$. For a factorial language the complexity is known to be either bounded by a constant or strictly increasing (cf. [1], first proved in [5]). 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 $\Omega(a^n)$ for some $\alpha > 1$ (bounded from below by an exponential function of base $\alpha$). We write $\Theta(n^p)$ for the function which is bounded from above and from below by polynomials of degree $p$.

The growth rate of the language $L$ is the upper limit $\alpha = \lim_{n \to \infty} \sqrt[n]{C_L(n)}$. The following theorem exhibits basic properties of growth rates for the factorial languages. In particular, we point out that for the factorial languages the upper limit can be replaced by the limit.

**Theorem 1.1.** ([8]). The growth rate of a factorial language $L \in \Sigma^*$ is given by $\alpha(L) = \inf_{n \in \mathbb{N}} \sqrt[n]{C_L(n)}$. Furthermore, $0 \leq \alpha(L) \leq |\Sigma|$, and

1) $\alpha(L) = 0 \iff L$ is finite;
2) $\alpha(L) = |\Sigma| \iff L = \Sigma^*$;
3) $\alpha(L) > 1 \iff C_L$ is exponential;
4) $\alpha(L) = 1$ otherwise.

Intuitively, the limit $\lim_{n \to \infty} C_L(n+1)/C_L(n)$ suits better for the name of the “growth rate”. Indeed, if such a limit exists, it is equal to $\alpha(L)$. (This is an easy corollary of the well-known theorem from analysis: the sequence of
geometric means of a converging sequence converges to the same limit.) On
the other hand, this limit exists for a much more narrow class of functions,
and no analogues of Theorem 1.1 are known.

The following theorem describes all possible types of complexity functions
for rational languages and serves as the starting point of our investigations.

**Theorem 1.2.** ([11]). Let a language \( L \) be recognized by a consistent DFA \( \mathcal{A} \). 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 encountered by a reading path.

In order to introduce the class of factorial languages which is the object
of our study, we need the concept of antidictionary.

A word \( W \) is forbidden for the language \( L \) if it is a factor of no 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 \( \Sigma \) has the antidictionary \( AD \), then the following equalities holds:

\[
L = \Sigma^* \setminus AD \cdot \Sigma^* \setminus AD, \quad AD = \Sigma \cdot L \cap L \cdot \Sigma \cap \Sigma^* \setminus L.
\]

Thus, any factorial language is determined by its antidictionary. We also
point out that a factorial language is rational if and only if so is its anti-
dictionary. In particular, the factorial languages with finite antidictionaries,
which we are interesting in, form a proper subclass of the class of rational
languages. This subclass plays a special role in the investigations on complex-
ity functions of languages. Namely, for languages with finite antidictionaries
there is an effective algorithm, evaluating the growth rate. The first full
description of it was made in [8], and is based on the method of [7]. In a
suitable notation, this algorithm looks as follows.

**Algorithm 1.**

Input: an antidictionary \( AD \).
Output: a number \( \alpha \) which is the growth rate of the factorial language \( L \)
with the antidictionary \( AD \).
Step 1. Let \( l = \max_{W \in AD} |W| \), and calculate the set \( L \cap \Sigma^{l-1} = \{U_1, \ldots, U_s\} \).
Step 2. Calculate the matrix \( M = (m_{ij}) \) by the rule

\[
m_{ij} = \begin{cases} 1, & \text{if } U_i c = d U_j \text{ for some } c, d \in \Sigma \\ 0, & \text{otherwise.} \end{cases}
\]

Step 3. Take the eigenvalue of \( M \) of the maximum absolute value for \( \alpha \).

Two notes should be added to the description of this algorithm. First, the
matrix \( M \) is actually the adjacency matrix of a deBruijn graph. These graphs
have wide applications, from pseudorandom numbers to DNA sequences, see [4]. Second, the eigenvalue of maximum absolute value of a nonnegative matrix is a nonnegative real number, which is called the Frobenius root of this matrix (cf. [6]). The Frobenius root of the adjacency matrix of a graph is called the index of this graph (see [3]).

We improve Algorithm 1, replacing the matrix $M$ with another adjacency matrix, the size of which is usually significantly smaller. In this way the calculations of the matrix, as well as its Frobenius root, are simplified substantially. Our modification is based on the following theorem, which generalizes Theorem 3.1 of [8].

**Theorem 1.3.** Let a language $L$ be recognized by a consistent DFA $\mathcal{A}$. Then the growth rate of $L$ coincides with the index of $\mathcal{A}$.

**Proof.** Let $A = (a_{ij})$ be the adjacency matrix of $\mathcal{A}$, $m$ be its size, $r$ be its Frobenius root, and $|A|$ be the sum of all elements of $A$. One of the well-known properties of the Frobenius root (see [6]) is the inequality

$$\min_i \sum_{j=1}^m a_{ij} \leq r \leq \max_i \sum_{j=1}^m a_{ij}.  \tag{1}$$

Thus, $r \leq |A| \leq mr$. Since $r^n$ is the Frobenius root of the matrix $A^n = (a^n_{ij})$ for any $n$, we have $r^n \leq |A^n| \leq mr^n$, whence we get the growth rate of $|A^n|$: $$\lim_{n \to \infty} |A^n|^{1/n} = r.$$

Note that $a^n_{ij}$ is the number of paths of length $n$ in $\mathcal{A}$ from the state $q_i$ to $q_j$. Hence, $|A^n|$ is the total number of paths of length $n$ in $\mathcal{A}$, and $P_i = \sum_{j=1}^m a^n_{ij}$ is the number of paths of length $n$ in $\mathcal{A}$, starting at $q_i$. If we denote $R_j(n) = a^n_{ij}$, the combinatorial complexity $C_L$ equals the sum of these *reading functions* $R_j$ over the set of terminal states. Thus, the maximum growth rate of the functions $R_j$ over the set of terminal states is $\alpha(L)$.

Suppose that $\mathcal{A}$ contains an edge $(q_i, q_j)$. Then $R_j(n+1) \geq R_i(n)$, yielding that the growth rate of $R_j$ is greater than or equal to the one of $R_i$. Therefore, the growth rates of the reading functions can only increase along a path in the automaton. Since $\mathcal{A}$ is consistent, for every state there exists a path from it to some terminal state. Thus, the overall maximum of the growth rates of the reading functions is achieved on a terminal state. We obtain that the sum $\sum_{j=1}^m R_j = P_1$ has the growth rate $\alpha(L)$.

There exists a path from the initial vertex to any vertex $q_i$, because $\mathcal{A}$ is consistent. Dually to the above argument on reading functions, we conclude that $P_1$ has at least the same growth rate as $P_i$. Since $|A^n| = \sum_{i=1}^m P_i(n)$, its growth rate is equal to the maximum of growth rates of $P_i$, that is, to the growth rate of $P_i$. This gives us the required equality $r = \alpha(L)$. \qed
2. Properties of FAD-automata

The following algorithm of [2] builds an automaton that recognizes a factorial language with the given finite antidictionary. We use the adjacency matrix of this automaton instead of the matrix $M$ in Algorithm 1. This automaton for short will be called FAD-automaton (from Finite AntiDictionary).

Algorithm 2.

Input: an antidictionary $AD$.
Output: a DFA $A$ recognizing the factorial language $L$ with the antidictionary $AD$.

Step 1. Construct a trie $T$, recognizing $AD$. ($T$ is actually the graph of the prefix order on the set of all prefixes of $AD$.)

Step 2. Associate each vertex in $T$ with the word labeling the reading 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 $T$, following the rule:
the edge $(U, W)$ labeled by $a$ should be added if

- $U$ is not terminal, and
- $U$ has no outgoing edge labeled by $a$, and
- $W$ is the longest suffix of $Ua$ which is a vertex of $T$.

(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 $A$.

Example 2.1. The antidictionary $AD = \{a^2, bbabb\}$ is recognized by the trie

and the FAD-automaton $A$ looks like

Throughout the paper, the symbols $T$ and $A$ are reserved for the trie and the automaton of Algorithm 2 respectively. Some useful properties of $A$ are collected in the following lemma.

Lemma 2.1.

1) The automaton $A$ is deterministic and consistent.

□
2) The set of vertices of \( A \) coincides with the set of all proper prefixes of the words from \( AD \).

3) The reading paths in \( A \) are exactly the paths starting in the initial vertex.

4) If the word \( Ua \) is forbidden for some vertex \( U \) of \( A \) and some letter \( a \), then no outgoing edge from \( U \) labeled by \( a \) exists.

5) The label of any reading path, ended in a vertex \( U \), has \( U \) as a suffix.

**Proof.**

1) The automaton \( A \) is deterministic by construction and consistent because 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 \( T \) 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 \( A \) are terminal.

4) After step 3 the transition function of the current automaton is well-defined, so we surely have an outgoing edge from \( U \) 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 \( Ua \) 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. \( U \) itself is not forbidden by 2), while \( Ua \) is. Then the word \( Ua \) has a forbidden suffix. Its minimal forbidden suffix \( W \) belongs to \( AD \) and, at the same time, \( W \) is its longest suffix which is a vertex of the trie. Hence the considered edge leads to \( W \), and \( W \) is terminal. The statement is proved.

5) We prove this fact by induction on length of a reading path. To prove the base of induction, we take any reading path labeled by a one-letter word, say \( c \). The edge labeled by \( c \) leads from the vertex \( \lambda \) either to the vertex \( c \) or back to \( \lambda \); hence, the base is fulfilled. For the inductive step assume that the desired property holds for all reading paths of length at most \( n \), and take a reading path of length \( n+1 \), labeled by some word \( Wc \), where \( c \) is the label of the last edge, say \((U_1, U_2)\). Then \( U_2 = U_1c \), if the last edge is forward, and \( U_2 \) is a suffix of \( U_1c \), if this edge is backward. In both cases \( U_2 \) is a suffix of \( Wc \), because \( U_1 \) is a suffix of \( W \) by the inductive assumption. The proof is finished.

\[ \square \]

**Remark 2.1.** The statement 1 of the above lemma justifies the application of Theorem 1.2 and Theorem 1.3 to FAD-automata.

**Remark 2.2.** The statement 4 of the above lemma is actually the main claim in the proof of the correctness of Algorithm 2 (see the full proof in [2]).

**Remark 2.3.** Algorithm 2 can be applied not only to antifactorial sets of words. If a set \( H \) of words contains no proper prefixes of its elements, the algorithm also returns an automaton accepting all words without factors in
Indeed, $H$ is recognizable by a trie; thus, all other steps of the algorithm and further conclusions remain valid.

We continue the previous example to shows the use of Algorithm 2 for the calculation of growth rates.

**Example 2.1 (continued).** If we order the vertices of $A$ like $\lambda, a, b, bb, bba, bbab$, then the adjacency matrix $A$ looks like

$$
egin{pmatrix}
0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0
\end{pmatrix}.
$$

Its characteristic polynomial is $x^6 - x^5 - x^4 + x^3 - x$, and its Frobenius root is $r \sim 1.5551$. □

Now we prove some graph properties of the FAD-automata.

**Lemma 2.2.** Let $U$ be a vertex of the trie $T$. If $T$ also contains the vertex $cU$ for any letter $c$, then $U$ belongs to no cycle of the automaton $A$.

**Proof.** Any cycle of $A$ contains at least one backward edge, because all forward edges are the ones of a trie. Hence, the necessary condition for the vertex $U$ to belong to a cycle is that there is a path to $U$ containing a backward edge. If such a path exists, then at least one backward edge leads to some prefix of $U$ (including $U$ itself). Now examine step 3 of the Algorithm. Suppose that a prefix $U'$ of $U$ is a proper suffix of the word $Wd$ for some vertex $W$ and letter $d$, say, $Wd = W'cU'$, where $c$ is a letter also. But the word $cU'$ is also a vertex as a prefix of the vertex $cU$. Hence, $U'$ can not be the destination vertex of the backward edge from $W$ labeled by $d$. Thus, no backward edges lead to the prefixes of $U$, and the lemma is proved. □

**Lemma 2.3.** If the vertex $S$ of $A$ belongs to a cycle of length $p$, then $S$ has period $p$ or $|S| < p$.

**Proof.** Let $U$ be the label of this cycle, read from the vertex $S$. Then $S$ is a suffix of $SU$ by condition 5) of Lemma 2.1, so we can write $SU = U'S$. By a well-known property of words (see [9], Proposition 1.3.4, for example), this equality implies $U = PQ, U' = QP$, and $S = Q(PQ)^n$, $n \geq 0$, for some words $P$ and $Q$. The statement of the lemma follows from this. □

**Proposition 2.1.** If the vertex $S$ of $A$ belongs to two cycles of length $p$ and length $q$ respectively, then $|S| < p + q - \gcd(p, q)$.

7
Proof. Arguing by contradiction, assume that $|S| \geq p + q - \gcd(p, q)$. Then $S$ has periods $p$ and $q$ by Lemma 2.3. By Fine and Wilf's theorem (see [9], Proposition 1.3.5, for example), $S$ has the period $\gcd(p, q)$. Let $U_1, U_2$ be the labels of two given cycles, read from the vertex $S$. Then $S$ is a suffix of both $SU_1$ and $SU_2$ by condition 5) of Lemma 2.1. On the other hand, $S \geq \max\{p, q\}$ by our assumption, whence $U_1$ and $U_2$ both are suffixes of $S$:

Thus, $U_1$ and $U_2$ both have the period $\gcd(p, q)$, and in view of their lengths, they are integer powers of the same word of length $\gcd(p, q)$. Hence, one of them, say $U_1$, is a prefix of another one. Since $A$ is deterministic, the path started from $S$ and labeled by $U_1$ is unique. As we supposed above, $U_1$ labels the first cycle. Finally, we see that either $U_2 = U_1$ or $U_2$ does not label a cycle, with both alternatives implying a contradiction.

The following useful corollary is straightforward.

**Corollary 2.1.** If the vertex $S$ of $A$ belongs to two different cycles of length $p$, then $|S| < p$.

The above corollary is true for an arbitrary alphabet, while the following important one suits the binary alphabet only.

**Corollary 2.2.** Over the binary alphabet, all length 2 cycles in $A$ are disjoint.

Proof. Arguing by contradiction, assume that the vertex $S$ belongs to two cycles of length 2. By Proposition 2.1, $|S| < 2$. If $S = \lambda$, then the outgoing edges from $S$ leads to the vertices $a$ and $b$. By Lemma 2.2, $S = \lambda$ belongs to no cycle. This contradicts to our assumption.

Now let $|S| = 1$, say, $S = a$. Then all ingoing edges of $S$ are labeled by $a$. Since $A$ is deterministic, one of the outgoing edges is labeled by $a$ also. This edge does not form a loop on $S$, so, it leads to the vertex $aa$. But the edge labeled by $a$ can not lead from $aa$ back to $S = a$, whence one of the cycles does not exist.

**Proposition 2.2.** The label of any cycle of $A$ is a primitive (cyclic) word. If a cycle is labeled by the word $U$ starting at some vertex $S$, then $S$ is a suffix of an appropriate power of $U$.

Proof. We first show the latter statement. As in Lemma 2.3, we have $SU = U'S$, and then $U = PQ$, $U' = QP$, and $S = Q(PQ)^n$, $n \geq 0$, for some words $P$ and $Q$. The statement immediately follows from this.

Now assume the contrary to the first statement. Namely, consider a cycle of $A$ labeled by a word $X^n$, $n \geq 2$, starting from some vertex $S$. As we
already proved, $S$ is a suffix of some power of $X^n$, whence $S = X'X^m$ for some $m \geq 0$ and some proper (possibly empty) suffix $X'$ of $X$. Hence, $S$ is a suffix of $SX$ also. Since the path labeled by $X$ does not lead from $S$ back to $S$, the word $SX$ has some longer suffix $S'$ in $A$ by step 3 of Algorithm 2. The form of $S$ yields that $S'$ is a suffix of $SX^n$ also, whence we get a contradiction with the assumption that the path labeled by $X^n$ leads from $S$ back to $S$. □

**Proposition 2.3.** The vertex $S$ of $A$ has a loop if and only if $S = c^m$ for some letter $c$ and $m \geq 0$, while $c^{m+1}$ is not a vertex of $T$.

**Proof.** Necessity. By Lemma 2.3 the vertex $S$ with a loop has period 1 or length 0, whence we get the first condition. According to step 3 of Algorithm 2, $S$ has no outgoing edge labeled by $c$ in the trie, whence $c^{m+1}$ is not in $T$.

Sufficiency. The word $S = c^m$ is the longest suffix of the word $Sc = c^{m+1}$ which is a vertex, by the conditions of the proposition. Hence, the edge from $S$ labeled by $c$ leads back to $S$, according to step 3 of Algorithm 2. □

**Corollary 2.3.** $A$ has at most $|\Sigma|$ loops.

**Corollary 2.4.** $A$ has no loop labeled by the letter $c$ if and only if there exists an integer $n$ such that $c^n \in AD$.

**Proposition 2.4.** Any multiple edge of $A$ has degree at most $|\Sigma| - 1$ and leads to the initial vertex.

**Proof.** After step 3 of Algorithm 2 any vertex $W$ of $A$ has the outgoing degree $|\Sigma|$, with at least one outgoing edge being forward. Hence, $W$ has at most $|\Sigma| - 1$ outgoing backward edges. Any two outgoing edges of $W$ have different labels, say $c$ and $d$. According to step 3 of Algorithm 2, these edges lead to some suffixes of the words $Wc$ and $Wd$ respectively. Their only common prefix is $\lambda$, whence the result. □

Let us look at Example 2.1 once more. We point out that, using Algorithm 1 directly, we would get a $8 \times 8$ matrix instead of $6 \times 6$ one we actually got. But in fact, only $5 \times 5$ matrix is needed, because the vertex $\lambda$ is not essential for calculating the growth rate. Some other cases of such redundant vertices are not so obvious, but can be discovered using Theorem 1.2. Thus, the removal of redundant vertices of the FAD-automaton obviously simplifies the calculation of the growth rate. The maximum subgraphs without redundant vertices are studied in the next two sections. We note that all graph properties of the FAD-automata, stated above, hold true for all their subgraphs as well.

9
3. C-graphs

Recall that a subgraph of a graph $G = (V, E)$ is called induced, if it has the form $(V', E_{|V'})$, where $V'$ is a nonempty subset of $V$ and $E_{|V'}$ is the set of all edges of $E$ with both ends in $V'$. A (connected) component of $G$ is a maximal by inclusion induced subgraph which is connected (i.e., there exists a path from any vertex to any other vertex). It is clear that a component of a graph is either a singleton graph, or a simple cycle, or a join of two or more cycles. In the first two cases the component is said to be trivial. Now we give the key definition.

The graph of cycles, or simply $C$-graph of the antidictionary $AD$ is the maximum induced subgraph with only non-trivial components, of the corresponding FAD-automaton $A$. Since we consider only languages of exponential complexity, the condition 2) of Theorem 1.2 implies that $A$ has at least one non-trivial component, so the definition is consistent. In Example 2.1 above, the C-graph is induced by all vertices of $A$ except $\lambda$. The following lemma shows that $A$ can be replaced by its C-graph when calculating the growth rate.

Lemma 3.1. The index of $A$ coincides with the index of its C-graph.

Proof. This lemma follows from the fact that the index of a graph equals the maximum index of its components (see [3]). By the definition, to get the C-graph we remove a set of vertices from $A$, and all these vertices belong to trivial components. The index of any cycle is equal to 1, the index of a singleton graph is zero, while the index of $A$ is greater, than 1, by Theorem 1.1. We see that all components of $A$ with the indices greater, than 1, remain in the C-graph.

Now, to obtain all possible growth rates of the languages with finite antidictionaries, one needs to list all possible C-graphs. This task seems to be hard. The Examples 3.1 and 3.2 below approve the intuitive suggestions that such growth rates can be arbitrarily close to 1 and to $|\Sigma|$, and that algebraic numbers of any degree can serve as growth rates. The problem of enumeration of C-graphs can be slightly simplified using the reduction procedure described in the next section. This simplification allows us to obtain a complete list of C-graphs with at most four vertices for the binary languages.

In some special cases it is possible to calculate the exact growth rate for an infinite series of languages. (By “exact” we mean that the characteristic polynomial can be obtained explicitly, whether or not the exact value of the Frobenius root is available.) Here we exhibit two infinite families of languages, for which the growth rates can be calculated easily. A more complicated, but still “calculable” family of languages, connected to the Thue-Morse words, can be found in [11].
Example 3.1. Let $\tilde{A}D_n = \{a^2, aba, \ldots, ab^{n-1}a\}$, and let $\tilde{L}_n$ denote the factorial language with the antidictionary $\tilde{A}D_n$. Then $\tilde{A}D_n$ is recognized by the trie

![Trie diagram]

and $\tilde{L}_n$ is recognized by the automaton

![Automaton diagram]

Thus, the C-graph coincides with the whole automaton, and is defined by the following adjacency matrix $\tilde{A}_n$:

\[
\begin{pmatrix}
1 & 1 & 0 & \ldots & 0 & 0 \\
0 & 0 & 1 & \ldots & 0 & 0 \\
0 & 0 & 0 & \ldots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \ldots & 0 & 1 \\
1 & 0 & 0 & \ldots & 0 & 0
\end{pmatrix}
\]

It is easy to see that the determinant $|\tilde{A}_n - xE|$ is equal to $(-1)^n x^n + (-1)^{n-1} x^{n-1} + (-1)^{n-1}$. Taking the leading coefficient positive, we obtain the characteristic polynomial $x^n - x^{n-1} - 1$. Standard considerations from analysis show that this polynomial has a unique root $\tilde{z}_n$ in $[1, 2]$, and $\tilde{z}_n \to 1$ as $n \to \infty$. For example, $\tilde{z}_2$ is the golden ratio, $\tilde{z}_4 \sim 1.3803$, and $\tilde{z}_{1000} \sim 1.0053$. □

The above example has one more point of interest. For any $n$ one has $\tilde{A}D_n \subset \tilde{A}D_{n+1}$, whence $L_n \supset \tilde{L}_{n+1}$. Thus, the sequence $\{L_n\}$ has a “limit” $\bar{L} = \bigcap_{n=1}^{\infty} \tilde{L}_n$. It is easy to see that $\bar{L} = b^*ab^*$, which is a rational language of linear complexity with the infinite antidictionary $\tilde{A}D = ab^*a$.

Example 3.2. Let the antidictionary $\hat{A}D_n$ consist of the single word $a^n$, and let $\hat{L}_n$ denote the factorial language with this antidictionary. Then $\hat{A}D_n$ is recognized by the trie

![Trie diagram]

and $\hat{L}_n$ is recognized by the automaton

![Automaton diagram]
The C-graph coincides with the whole automaton, and is defined by the following adjacency matrix $\hat{A}_n$:

$$
\begin{pmatrix}
1 & 1 & 0 & \ldots & 0 & 0 \\
1 & 0 & 1 & \ldots & 0 & 0 \\
1 & 0 & 0 & \ldots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
1 & 0 & 0 & \ldots & 0 & 1 \\
1 & 0 & 0 & \ldots & 0 & 0
\end{pmatrix}
$$

Calculating the determinant $|\hat{A}_n - xE|$ and taking the leading coefficient positive, we obtain the characteristic polynomial $x^n - x^{n-1} - x^{n-2} - \ldots - 1$. It has a unique root $\hat{z}_n$ in $[1, 2]$, and $\hat{z}_n \to 2$ as $n \to \infty$. As in the previous example, this sequence begins with the golden ratio $\hat{z}_2$; $\hat{z}_4 \sim 1.9276$, and $\hat{z}_{500} \sim 1,9998$. □

4. Reduced antidictionaries

Since we need to calculate the index of some graph, we want to reduce the size of this graph as much as possible. Two possibilities of reduction are already used. Namely, we use a good algorithm, that constructs the automaton by the antidictionary, and we extract the essential subgraph of this automaton. The third possibility is to reduce the antidictionary preserving the growth rate. A simple example shows what we want to do. Suppose that the antidictionary contains the words $Wc$ for a fixed word $W$ and all letters $c$. Then the vertex $W$ of $A$ has no outgoing edges, whence it forms a singleton component. So, this vertex may be removed, and it can be done, for example, by replacing all words $Wc$ in the antidictionary with the single word $W$.

Recall that the reversal of the word $W = a_1 \ldots a_n$ is $\overline{W} = a_n \ldots a_1$, and $\overline{L} = \{\overline{W} \mid W \in L\}$. The reduction step for the antidictionary $AD$ is one of the following transformations.

- If all vertices of the FAD-automaton $A$ of $AD$ with the given prefix $W$ belong to trivial components of $A$, then replace all words of $AD$, containing $W$, with a single word $W$.

- If all vertices of the FAD-automaton $\overline{A}$ of $\overline{AD}$ with the given prefix $\overline{W}$ belong to trivial components of $\overline{A}$, then replace all words of $AD$, containing $W$, with a single word $W$. 

12
Proposition 4.1. The reduction step preserves the growth rate.

Proof. We prove this statement for the first transformation rule and then use the symmetry argument. For convenience, divide the reduction step into two substeps. First, replace all words of $AD$, having a prefix $W$, with the word $W$; second, remove from the obtained set all words with the proper factor $W$. The set $AD'$ obtained after the first substep need not be antifactorial, but we use Remark 2.3 and apply Algorithm 2 to it, getting a trie $T'$ and an automaton $\mathcal{A}'$. We show that $\mathcal{A}'$ coincides with the automaton $\bar{\mathcal{A}}$, obtained from $\mathcal{A}$ by deletion of all vertices with the prefix $W$.

The underlying tree of $T'$ is obtained from the one of $T$ by deletion of the subtree with the root $W$, hence the sets of vertices of $\mathcal{A}'$ and $\bar{\mathcal{A}}$ coincide, as well as the sets of forward edges. Take a backward edge $(U_1, U_2)$ of $\bar{\mathcal{A}}$. By step 3 of Algorithm 2, $U_2$ is the maximum length suffix of some word $U_1c$ among the vertices of $T$. The maximality over $T$ implies the maximality over $T'$, whence the backward edge of $\mathcal{A}'$, labeled by $c$, leads to $U_2$ as well.

Now assume that $(U_1, U_2)$ is a backward edge of $\mathcal{A}'$, that is, $U_2$ is the maximum length suffix of $U_1c$ among the vertices of $T'$. If it is the maximum length suffix over $T$ also, then $\bar{\mathcal{A}}$ contains the same edge. But if $T$ contains a longer suffix of $U_1c$, then this suffix contains $W$. By condition 4) of Lemma 2.1, the backward edge $(U_1, U_2)$ in $\mathcal{A}'$ does not exist. Thus, we finally proved that $\mathcal{A}' = \bar{\mathcal{A}}$.

All the vertices deleted from $\mathcal{A}$ to obtain $\mathcal{A}'$ belong to trivial components of $\mathcal{A}$. Thus, the C-graphs of $\mathcal{A}$ and $\mathcal{A}'$ coincide, as well as the growth rates, in view of Lemma 3.1, and we are done with the first substep.

As to the second substep, it just preserves the language with the set $AD'$ of forbidden words, together with its complexity and growth rate. Indeed, a factorial language is defined by the set of minimal forbidden words, and this set is not changed during this substep. The statement of the lemma is proved for the first transformation rule.

The validity of the second rule follows from symmetry. Indeed, if a language $L$ has the antidictionary $AD$, then $\overline{L}$ has the antidictionary $\overline{AD}$, whence $C_L(n) = C_{\overline{L}}(n)$.

The antidictionary is called reduced, if no reduction step is possible.

Example 4.1. Let $\Sigma = \{a, b\}, AD = \{ba^3, babb, baabb\}$. The reduction procedure takes four steps, all made by the second rule.

1) Replace $ba^3$ with $a^3$: the vertex $a^3$ of the “reverse” FAD-automaton has only one outgoing edge, which is a loop.

2) Replace $baabb$ with $aabb$: since both words $baabb$ and $aabb$ are forbidden, the vertex $bbaa$ of the “reverse” FAD-automaton has no outgoing edges.

3) Replace $aabb$ and $bab$ with $abb$, like in step 2).

4) Replace $abb$ with $bb$, like in step 1).

The reduced antidictionary $\{a^3, b^2\}$ is obtained. It is worth noting that the
FAD-automaton and the C-graph for this antidictionary contain four and three vertices respectively, instead of six and four for AD. 

**Remark 4.1.** For any $n \geq 1$ the words $a^n b$, $ab^n$, and their reversals do not belong to reduced antidictionaries over the binary alphabet.

Below we obtain several properties of C-graphs for reduced antidictionaries. These properties are very useful in the proof of Theorem 5.1.

**Lemma 4.1.** Let $AD$ be a reduced antidictionary. After step 3 of Algorithm 2, each leaf of the trie $T$ can be reached from some vertex of the C-graph.

**Proof.** We assume that a list $X$ of $T$ can not be reached from any vertex of the C-graph, and obtain that $AD$ is not reduced. We have $X \in AD$. Let $X = X'c$, where $c$ is a letter. Obviously, $X'$ is not reachable from any vertex of the C-graph by our assumption. Moreover, all vertices of $A$ with the suffix $X'$ are not reachable from the C-graph also. Indeed, $T$ has no vertices with the proper suffix $X$, since $AD$ is antifactorial; hence, any vertex of the form $ZX'$ has a backward edge to $X$ labeled by $c$.

We take the automaton $A$ and delete all vertices from the C-graph. The remained automaton $A'$ still reads the same words with the suffix $X'$, as $A$ itself. Since $A'$ has no non-trivial components, the language, recognized by $A'$, has polynomial complexity by Theorem 1.2. Thus, the language $L$ with the antidictionary $AD$ has only polynomially many words with the suffix $X'$. By symmetry, the language $\overline{L}$ with the antidictionary $\overline{AD}$ has only polynomially many words with the prefix $\overline{X}$.

Let us read the word $\overline{X'}$ from the initial vertex of the automaton $\overline{A}$. We come to some suffix of $\overline{X'}$, say $Y$. We have only polynomially many paths in $\overline{A}$ starting at $Y$. By Theorem 1.2, neither of the vertices reached from $Y$ belongs to a non-trivial component. Therefore, the second reduction rule is applicable to $AD$. 

**Lemma 4.2.** If all words of the binary reduced antidictionary $AD$ begin with the same letter, then the vertex $\lambda$ of $A$ belongs to the C-graph.

**Proof.** Let $a$ be the first letter of all words of $AD$. Then the vertex $\lambda$ of $A$ has a loop labeled by $b$ by Proposition 2.3. Since any vertex of $A$ is reachable from $\lambda$, it suffices to show that there is a backward edge coming to $\lambda$ from some other vertex. Take the vertex $ab^n$ of $A$ with the maximum possible $n$ (note that the case $n = 0$ is also possible). By Remark 4.1 and the maximality, this vertex has the only outgoing edge in the trie $T$, and this edge is labeled by $a$. Thus, this vertex has an outgoing backward edge, which is labeled by $b$. No one of nonempty suffixes of the word $ab^{n+1}$ is in $T$. Then, this backward edge comes to $\lambda$ by step 3 of Algorithm 2. The lemma is proved.
A vertex of $\mathcal{A}$ is said to be marginal, if it has no outgoing forward edges in $\mathcal{A}$. If $U$ is marginal, then the only vertex of $\mathcal{A}$ with the prefix $U$ is $U$ itself. So, the next remark immediately follows from the definitions.

**Remark 4.2.** For a reduced antidictionary $AD$, all marginal vertices of $\mathcal{A}$ belong to the C-graph.

**Lemma 4.3.** For a reduced binary antidictionary $AD$, all marginal vertices of $\mathcal{A}$ have the outgoing degree 1.

*Proof.* A marginal vertex $U$ has no outgoing forward edges by definition, whence $\deg^+(U) < |\Sigma| = 2$. On the other hand, it belongs to a non-trivial component of $\mathcal{A}$ by Remark 4.2, whence $\deg^+(U) \geq 1$. $\square$

**Proposition 4.2.** The C-graph of any reduced binary antidictionary contains a vertex with the ingoing and outgoing degrees both equal to 1.

*Proof.* Suppose that the considered C-graph is defined by the reduced antidictionary $AD$. Let $U$ be the maximum length proper prefix of some maximum length word of $AD$. Then $U$ is marginal, belongs to the C-graph by Remark 4.2, and $\deg^+(U) = 1$ by Lemma 4.3.

Now check the ingoing edges of $U$. The length of any word in $AD$ is at least two, whence $|U| > 0$, and so $U$ has an ingoing forward edge. Let $U$ also have an ingoing backward edge from some vertex $W$. By step 3 of Algorithm 2, $U$ is a proper suffix of the word $Wc$ for appropriate letter $c$. This condition implies $|U| \leq |W|$, whence $|U| = |W|$ by the definition of $U$, and $W$ is also the maximum length proper prefix of some maximum length word of $AD$. Thus, $W$ belongs to the C-graph also. We replace $U$ by $W$ and repeat the argument. By finiteness of $AD$ we shall find either a vertex without ingoing backward edges, or a vertex with an ingoing backward edge from $U$. In the first case the required condition is fulfilled, while in the second one we get a contradiction with the definition of C-graph. Indeed, we have found a cycle constituted by the vertices of outgoing degree 1; hence, it is a trivial component. $\square$

**Lemma 4.4.** If the C-graph of a reduced binary antidictionary $AD$ is connected, it contains at least $|AD|$ vertices of the outgoing degree 1. The missing edge of each such vertex ended in a leaf of the trie $T$ and was deleted on step 4 of Algorithm 2.

*Proof.* By lemma 4.1, for any leaf $X$ of the trie $T$ there is a path from some vertex $U$ of the C-graph to $X$ in the complete automaton built by step 3 of Algorithm 2. Choose one such path of minimum length for each leaf. Prove that all chosen paths start at different vertices.

Assume the contrary: two chosen paths start at the same vertex $U$. The first edges of these paths does not belong to the C-graph by the length argument. Since the alphabet is binary, $U$ has at most one outgoing edge
outside the C-graph. Hence, these paths has the common first edge, say, $(U, Y)$. Then $Y$ is not a leaf, hence, $Y$ is a vertex of the automaton $A$. Since the C-graph is connected, and $Y$ is reached from it, no paths from $Y$ back to the C-graph exist. Hence, the first reduction rule is applicable to $AD$.

Since all chosen paths start at different vertices, the number of these vertices is equal to the number $|AD|$ of leaves in the trie. The first statement of lemma follows from this.

To prove the second statement, consider one of the chosen paths again. Let $(U, Y)$ be its first edge. The vertex $Y$ is not in the C-graph by the length argument; if $U$ is in $A$ we obtain, similar to the above, that $AD$ is reducible. Thus, $Y$ is a leaf of $T$, whence the required statement.

We note that some connected C-graphs of reduced binary antidictionaries have more than $|AD|$ vertices of the outgoing degree 1. One such C-graph is presented in Example 2.1: the vertices $a$ and $bba$ both had an edge to the leaf $aa$ after step 3 of Algorithm 2.

5. Small C-graphs and growth rates for binary antidictionaries

According to Proposition 4.1 all possible growth rates are achieved by languages with reduced antidictionaries. The list of small C-graphs of reduced binary antidictionaries is given in tables 1–3. The following theorem claims that this list is complete.

**Theorem 5.1.** If the C-graph of a language with a reduced binary antidictionary has at most four vertices, then it is isomorphic to one of the graphs 1.1-1.9, 2.1-2.13, or 3.1-3.3.

Before proving this theorem we note that some different reduced antidictionaries do have isomorphic C-graphs (we do not mind the trivial case when these antidictionaries coincide up to renaming the letters). For example, the C-graph of the antidictionary $\{a^3, bab\}$ is isomorphic to the graph 2.1.

**Proof.** Summarizing the statements above, we see that any C-graph of a reduced binary antidictionary possesses the following properties:

1. all the components are non-trivial (the definition);
2. $1 \leq \deg^+(v) \leq 2$ for any vertex $v$ (the definitions of C-graph and $A$);
3. there is a vertex $v$ such that $\deg^+(v) = \deg^-(v) = 1$ (Proposition 4.2);
4. there are no multiple edges (Proposition 2.4);
5. the number of loops is at most 2 (Corollary 2.3);
6. all cycles of length 2 are disjoint (Corollary 2.2).
Table 1: Antidictionaries of one word

<table>
<thead>
<tr>
<th>No.</th>
<th>$AD$</th>
<th>C-graph</th>
<th>Characteristic polynomial</th>
<th>Growth rate</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>$a^2$</td>
<td><img src="image" alt="C-graph" /></td>
<td>$x^2 - x - 1$</td>
<td>1.618</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>$a^3$</td>
<td><img src="image" alt="C-graph" /></td>
<td>$x^3 - x^2 - x - 1$</td>
<td>1.839</td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>$aba$</td>
<td><img src="image" alt="C-graph" /></td>
<td>$x^3 - 2x^2 + x - 1$</td>
<td>1.755</td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>$a^4$</td>
<td><img src="image" alt="C-graph" /></td>
<td>$x^4 - x^3 - x^2 - x - 1$</td>
<td>1.928</td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>$aaba$</td>
<td><img src="image" alt="C-graph" /></td>
<td>$x^4 - 2x^3 + x - 1$</td>
<td>1.867</td>
<td>Cs: 1.7; 1.9</td>
</tr>
<tr>
<td>1.6</td>
<td>$aabb$</td>
<td><img src="image" alt="C-graph" /></td>
<td>$x^4 - 2x^3 + 1$ = $(x-1)(x^3-x^2-x-1)$</td>
<td>1.839</td>
<td></td>
</tr>
<tr>
<td>1.7</td>
<td>$abaa$</td>
<td><img src="image" alt="C-graph" /></td>
<td>$x^4 - 2x^3 + x - 1$</td>
<td>1.867</td>
<td>Cs: 1.5; 1.9</td>
</tr>
<tr>
<td>1.8</td>
<td>$abab$</td>
<td><img src="image" alt="C-graph" /></td>
<td>$x^4 - 2x^3 + x^2 - 2x + 1$ = $(x^2 - (1+\sqrt{2})x + 1) \cdot (x^2 - (1-\sqrt{2})x + 1)$</td>
<td>1.883</td>
<td></td>
</tr>
<tr>
<td>1.9</td>
<td>$abba$</td>
<td><img src="image" alt="C-graph" /></td>
<td>$x^4 - 2x^3 + x - 1$</td>
<td>1.867</td>
<td>Cs: 1.5; 1.7</td>
</tr>
<tr>
<td>No.</td>
<td>AD</td>
<td>C-graph</td>
<td>Characteristic polynomial</td>
<td>Growth rate</td>
<td>Remarks</td>
</tr>
<tr>
<td>-----</td>
<td>----------</td>
<td>---------</td>
<td>---------------------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>2.1</td>
<td>$a^2 aba$</td>
<td><img src="image1" alt="Diagram" /></td>
<td>$x^3 - x^2 - 1$</td>
<td>1,466</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>$a^2 b^3$</td>
<td><img src="image2" alt="Diagram" /></td>
<td>$x^3 - x - 1$</td>
<td>1,325</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>$a^2 abba$</td>
<td><img src="image3" alt="Diagram" /></td>
<td>$x^4 - x^3 - x^2 + x - 1$</td>
<td>1,513</td>
<td></td>
</tr>
<tr>
<td>2.4</td>
<td>$a^2 b^4$</td>
<td><img src="image4" alt="Diagram" /></td>
<td>$x^4 - x^2 - x - 1 = (x+1)(x^3 - x^2 - 1)$</td>
<td>1,466</td>
<td>2nd factor</td>
</tr>
<tr>
<td>2.5</td>
<td>$a^3 aaba$</td>
<td><img src="image5" alt="Diagram" /></td>
<td>$x^4 - x^3 - x^2 - 1 = (x+1)(x^3 - 2x^2 + x - 1)$</td>
<td>1,755</td>
<td>2nd factor</td>
</tr>
<tr>
<td>2.6</td>
<td>$a^3 aabb$</td>
<td><img src="image6" alt="Diagram" /></td>
<td>$x^4 - x^3 - x^2 - x + 1$</td>
<td>1,722</td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td>$a^3 aba$</td>
<td><img src="image7" alt="Diagram" /></td>
<td>$x^4 - x^3 - x - 1 = (x^2 + 1)(x^2 - x - 1)$</td>
<td>1,618</td>
<td>Cs: 2.13; Dv: no</td>
</tr>
<tr>
<td>2.8</td>
<td>$a^3 bab$</td>
<td><img src="image8" alt="Diagram" /></td>
<td>$x^4 - x^3 - 2x = x \cdot (x^3 - x^2 - 2)$</td>
<td>1,696</td>
<td>2nd factor</td>
</tr>
<tr>
<td>2.9</td>
<td>$a^3 babb$</td>
<td><img src="image9" alt="Diagram" /></td>
<td>$x^4 - x^3 - x^2 = x^2 \cdot (x^2 - x - 1)$</td>
<td>1,618</td>
<td>Dv: no</td>
</tr>
</tbody>
</table>
Table 2: Antidictionaries of two words (continued)

<table>
<thead>
<tr>
<th>No.</th>
<th>AD</th>
<th>C-graph</th>
<th>Characteristic polynomial</th>
<th>Growth rate</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.10</td>
<td>$a^3$</td>
<td><img src="image1" alt="C-graph" /></td>
<td>$x^4 - x^2 - 2x - 1 = (x^2-x-1)(x^2+x+1)$</td>
<td>1,618</td>
<td>Dv: 1.1</td>
</tr>
<tr>
<td>2.11</td>
<td>$aba$</td>
<td><img src="image2" alt="C-graph" /></td>
<td>$x^4 - 2x^3 + x^2 - 1 = (x^2-x-1)(x^2+x+1)$</td>
<td>1,618</td>
<td>Cs: 2.12 Dv: 1.1</td>
</tr>
<tr>
<td>2.12</td>
<td>$aba$</td>
<td><img src="image3" alt="C-graph" /></td>
<td>$x^4 - 2x^3 + x^2 - 1 = (x^2-x-1)(x^2+x+1)$</td>
<td>1,618</td>
<td>Cs: 2.11 Dv: no</td>
</tr>
<tr>
<td>2.13</td>
<td>$b^4$</td>
<td><img src="image4" alt="C-graph" /></td>
<td>$x^4 - x^3 - x - 1 = (x^2 + 1)(x^2 - x - 1)$</td>
<td>1,618</td>
<td>Cs: 2.7 Dv: 1.1</td>
</tr>
</tbody>
</table>

Comments to the “Remarks” column: Cs are cospectral graphs, Dv are divisors; “2nd factor” means that the characteristic polynomial of the recurrent formula for the complexity of this language is the second factor in the given factorization.

Table 3: Antidictionaries of three words

<table>
<thead>
<tr>
<th>No.</th>
<th>AD</th>
<th>C-graph</th>
<th>Characteristic polynomial</th>
<th>Growth rate</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>$a^2$</td>
<td><img src="image5" alt="C-graph" /></td>
<td>$x^4 - x^3 - 1$</td>
<td>1,380</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>$b^4$</td>
<td><img src="image6" alt="C-graph" /></td>
<td>$x^4 - x - 1$</td>
<td>1,221</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>$b^4$</td>
<td><img src="image7" alt="C-graph" /></td>
<td>$x^4 - x^2 - 1 = (x^2 - 1 + \sqrt{5}) (x^2 - 1 - \sqrt{5})$</td>
<td>1,272</td>
<td></td>
</tr>
</tbody>
</table>

Comments to the “Remarks” column: Cs are cospectral graphs, Dv are divisors; “2nd factor” means that the characteristic polynomial of the recurrent formula for the complexity of this language is the second factor in the given factorization.
So, it is enough to check only graphs with such properties. The properties (1)-(4) immediately yield that no C-graphs with two vertices exist except the graph 1.1. As to the C-graphs with 3 vertices, we first note that they are connected, since a one-vertex component has to be trivial by (4). Further, such connected graph contains at most one length 2 cycle by (6), and hence is gamiltonian. By (1)-(3), either one or two edges are to be added to the gamiltonian cycle to get a C-graph. Two non-isomorphic graphs can be obtained by adding one edge, and they are precisely the graphs 2.1 and 2.2. If two edges are added, they start in different vertices by (2), and neither one of them ends in the third vertex by (3). The only two remaining possibilities give us the graphs 1.2 and 1.3. Thus, we find all C-graphs with two or three vertices.

We see that the properties (1)-(6) are sufficient for a graph with at most three vertices to be the C-graph of some reduced binary antidictionary. It is not the case for the graphs with four vertices, so we need a careful study. First we note that the C-graph has 5 to 7 edges by (1)-(3), whence the antidictionary consists of at most three words by Lemma 4.4. Now prove that the C-graph is connected.

Consider a graph $G$ with the vertices $v_1$, $v_2$, $v_3$, and $v_4$ that satisfies (1)-(6) but is not connected. Both its components should be isomorphic to the graph 1.1. Let the vertices $v_1$ and $v_2$ induce one component, while $v_3$ and $v_4$ induce the other one, with $v_1$ and $v_3$ having loops. Assume that $G$ is a C-graph and reconstruct the antidictionary. Consider the vertices of $G$ as words. By Proposition 2.1, $|v_1|, |v_3| \leq 1$, and the labels of the loops differ in view of Proposition 2.3. Suppose that the letter $a$ labels the loop on $v_1$, while $b$ labels the loop on $v_3$. Then the labels of other known edges appear like this:

(Recall that one more edge can present, connecting the components.) If $v_1 = \lambda$, then the only possibility for $v_3$ is $b$. Hence, the edge labeled by $b$ must lead from $v_1$ to $v_3$, a contradiction. By a similar argument, $v_3 \neq \lambda$ also. Thus, $v_1 = a$ and $v_3 = b$. The edge labeled by $b$ leads from the vertex $a$ to a suffix of $ab$; since this edge does not lead to $v_3 = b$, it leads to the vertex $ab = v_2$. Similarly, $v_4 = ba$. The edge labeled by $a$ leads from $v_2$ to $a$, but $v_4$ is a longer suffix of $aba$; thus, we get a contradiction with step 3 of Algorithm 2. Therefore, the considered antidictionary does not exist, and $G$ is not a C-graph.
We begin the enumeration of 4-vertex C-graphs with the graphs of one-word antidictionaries. If \( AD = \{ W \} \), then the C-graph contains the vertex \( \lambda \) by Lemma 4.2 and the maximum length prefix of \( W \) by Remark 4.2. We already proved that the C-graph is connected, whence it coincides with the whole automaton \( A \). The number of vertices in \( A \) is \(| W |\). Thus, we need exactly a 4-letter word to produce a 4-vertex C-graph. Without loss of generality, assume that \( W \) begins with the letter \( a \). By Remark 4.1, we have six possible reduced antidictionaries with \(| W | = 4\). They produce the graphs 1.4-1.9.

Next we move to the antidictionaries of at least two words. By (1), (2), and Lemma 4.4, the C-graph contains five or six edges. We enumerate all possible graphs and prove that each of them either is contained in one of the tables 2, 3, or is not a C-graph.

We begin with graphs having five edges. If such a graph contains a hamiltonian cycle, then only one edge remains. All three locations of this edge, satisfying (4), are represented by the graphs 3.1-3.3. On the other hand, if such a graph is not hamiltonian, it contains a 3-cycle and two more edges to connect to the last vertex. In this way one can get only the graphs \( G_1 \) and \( G_2 \), represented below.

\[
\begin{align*}
G_1: & \quad v_1 \quad v_2 \\
& \quad v_3 \quad v_4 \\
G_2: & \quad v_1 \quad v_2 \\
& \quad v_3 \quad v_4
\end{align*}
\]

It will be proved that \( G_1 \) and \( G_2 \) are not C-graphs, but first we proceed with the graphs having six edges to complete the list of such “exceptions”.

Suppose that the considered graph with six edges contains a hamiltonian cycle. If both remaining edges are loops, then only two cases are possible, namely, the graphs 2.11 and 2.12. If only one loop exists, then the last edge starts in one of three other vertices. In each case there are two possible vertices to end the last edge, because this edge neither loops nor duplicates the existing edge. Therefore, we have six non-isomorphic gamiltonian graphs with one loop, namely, the graphs 2.3, 2.5, 2.7, 2.13, and the “exceptions” \( G_3 \) and \( G_4 \) below.

If the C-graph with six edges (hamiltonian or not) has no loops, then the antidictionary contains some powers of both \( a \) and \( b \) by Corollary 2.4. By Lemma 4.4, the antidictionary consists of exactly two words, say \( a^m \) and \( b^n \). The corresponding FAD-automaton \( A \) has the backward edges from any power of \( a \) to the vertex \( b \), and from any power of \( b \) to the vertex \( a \). Hence, the C-graph contains all vertices of \( A \) except \( \lambda \), namely, \((m−1) + (n−1)\) vertices. Thus, \( m + n = 6 \), and we get only two different C-graphs 2.4 and 2.10.

Now only non-hamiltonian graphs with loops remain. After removing the loop, the graph should still be connected, whence it is either \( G_1 \) or \( G_2 \). In
view of (2), there are three ways to add a loop to each of these graphs. We also mention that the loops added to the vertices $v_1$ and $v_4$ of $G_1$ give a pair of isomorphic graphs. Thus, we have five graphs, namely, the graphs 2.6, 2.8, 2.9, and the “exceptions” $G_5$ and $G_6$.

To prove the theorem it remains to show that neither of the graphs $G_1$–$G_6$ is a C-graph. For each graph we suppose that it is a C-graph of a reduced antidictionary $AD$, and refer to its vertices as to words, trying to recover the set $AD$. The preliminary remarks are:

(i) among two outgoing edges of a vertex at least one is forward;

(ii) the destination vertex of any edge is at most one letter longer than the starting vertex; whence, by Lemma 4.4,

(iii) the vertex $\lambda$ has two outgoing edges.

For $G_1$ the argument is easy. By Corollary 2.1, $|v_2| < 3$. Then, by Proposition 2.2, $v_2$ is a common suffix of the labels of both cycles, read from $v_2$. But the second letters of these labels are different, since the C-graph is extracted from a deterministic automaton. Hence, the length of this common suffix is at most 1. By (iii) we finally get $|v_2| = 1$, say, $v_2 = a$. By Remark 4.1 the missing edge ended in the forbidden word $a^2$. Note that $v_3$ has an outgoing edge labeled by $a$, say, $(v_3, v_4)$. By (iii), $v_4 \neq \lambda$, whence it ends with $a$. Then the word $v_4a$ ends with $a^2$. But $a^2$ is in the trie, whence the edge $(v_4, v_2)$ was added in a contradiction with step 3 of Algorithm 2.

The graph $G_2$ requires the longest consideration. The main reason is that $G_2$ can appear as a C-graph of a reducible antidictionary, or as a component of a C-graph of a reduced antidictionary, but not as a whole C-graph of a reduced antidictionary. In the following analysis we heavily use Lemma 4.4 to make conclusions of the following two types:

(*) if the outgoing edge from the vertex $U$ labeled by $c$ is missing, then the word $Uc$ is forbidden;

(**) if, moreover, the existing outgoing edge from $U$ is backward, then the word $Uc$ belongs to the antidictionary.

First we show that $G_2$ does not contain the $\lambda$ vertex. By (iii), the only candidate for $\lambda$ is $v_3$. Suppose that $v_3 = \lambda$. Then one of the vertices $v_1$, $v_4$,
is $a$ and the other one is $b$ by (ii). Hence, no backward edges can lead to $\lambda$
by Lemma 2.2, a contradiction.

Next we use Proposition 2.2 to get the only possible, up to renaming the
letters, labeling of $G_2$:

By Proposition 2.1 we have $|v_3| \leq 3$. Considering the labels of the edges,
we see that $v_3 \in \{a, ba, aba\}$ in view of Proposition 2.2.

Case 1: $v_3 = a$. Then $v_1$ is a suffix of $a^2$. Since $v_1 \neq \lambda$, $v_1 \neq a = v_3$, we have
$v_1 = a^2$. Further, $v_2$ is the longest suffix of $aab$ in the graph, while $v_4$ is the
longest suffix of $ab$. Since $v_2 \neq v_4$, we get $v_2 = aab$. Then $aab \in AD$, and
either $b^2$ or $abb$ also belongs to the antidictionary, depending on the value of $v_4$. In both cases the antidictionary is not antifactorial, a contradiction.

Case 2: $v_3 = ba$. Then $v_1$ is a suffix of $aab$.

Case 2.1: $v_1 = a$. Then $a^2$ is forbidden. Hence $aab$ is forbidden also, and the
dge $(v_3, v_1)$ is added incorrectly, a contradiction.

Case 2.2: $v_1 = a^2$. Then $v_4 = bab$ by (i), yielding $bab \in AD$. Depending
on the value of $v_2$, which is a suffix of $aab$, one of the words $b^2$, $aab$, $aab$ is
forbidden. Each of the first two words violates the definition of the antidictionary. As to the third one, such an antidictionary, if exists, can be reduced by replacing all words containing $aab$ with the word $abb$.

Case 2.3: $v_1 = bab$. Then the word $baaa$ is forbidden. By Remark 4.1 this
word cannot belong to a reduced antidictionary. On one hand, this fact
means that $a^3$ is forbidden. On the other hand, it implies that the edge
$(v_1, v_2)$ is forward, whence $v_2 = bab$ and $bab \in AD$. Now, both words
$aaabb$ and $bab$ are forbidden. Then the antidictionary can be reduced by
replacing all words containing $aab$ with the word $aab$.

Case 3: $v_3 = aba$. Then $v_1$ is a suffix of $baa$.

Case 3.1: $v_1 = a$ is similar to the case 2.1.

Case 3.2: $v_1 = a^2$. Then $a^3$ is forbidden. Further, $v_4 = abab$ by (i), yielding
$abab \in AD$. If $v_2$ is a proper suffix of $aab$, then $b^2$ or $abb$ is forbidden,
contradicting to the definition of the antidictionary. If $v_2 = aab$, then
$aab \in AD$. Now we see that $\lambda$ belongs to the $C$-graph by Lemma 4.2,
a contradiction.

Case 3.3: $v_1 = bab$ is similar to the case 2.3.

Case 3.4: $v_1 = abaa$. Then $abaa$ is forbidden, $v_2$ is a suffix of $abab$, $v_4$
is a suffix of $abab$, and $|v_2|, |v_4| \geq 2$ by (ii). Further, both words $v_2b$, $v_4b$
are forbidden (and certainly belong to the antidictionary when the length of the
vertex is at least 3). Hence, if one of these vertices is $ab$, we can not get an
antifactorial antidictionary. Assume that $|v_2|, |v_4| \geq 3$. If $|v_2| = |v_4| = 3$, then $aabb, babb \in AD$, whence the antidictionary can be reduced using the word $abb$. In all remaining subcases we get a contradiction, finding an additional component of the C-graph. In each of the pictures below, this component is drawn in boldface.

Case 3.4.1: $v_2 = aab$, $v_4 = abab$. All words in the antidictionary begin with $a$, whence $\lambda$ belongs to the C-graph by Lemma 4.2.

Case 3.4.2: $v_2 = baab$, $v_4 = bab$. $AD = \{abaaa, baabb, babb\}$.

Case 3.4.3: $v_2 = baab$, $v_4 = abab$. $AD = \{abaab, baabb, ababb\}$.

Case 3.4.4: $v_2 = abaab$, $v_4 = bab$. In this and the next cases the forbidden word $abaab$ does not need to be minimal. If it is not, then $a^3 \in AD$ ($baaa$ can be reduced, while $a^2$ is not forbidden). The first automaton is obtained from the antidictionary $AD = \{abaab, abaabb, bab\}$, and the second one is obtained from $AD = \{a^3, abaabb, bab\}$. 

24
Case 3.4.5: $v_2 = abaab$, $v_4 = abab$ is similar to the case 3.4.1. Thus, we proved that $G_2$ is not a C-graph.

Consider the graph $G_3$. The edge $(v_1, v_2)$ is forward by (i). All forward edges belong to a tree, whence $(v_3, v_2)$ is backward. Then $(v_4, v_1)$ is forward by (i). One has $|v_2| - |v_1| = 1$, $|v_1| - |v_4| = 1$, whence $|v_2| - |v_4| = 2$, a contradiction to (ii).

Now move to the graph $G_4$ and label its edges in accordance with Proposition 2.2. This can be done in a unique way, up to renaming the letters:

The length of $v_3$ is at most 3 by Proposition 2.1. By Proposition 2.2, $v_3$ is a suffix of $aabb$, and simultaneously a suffix of $(ab)^n$. Since $v_3 \neq \lambda$ by (iii), we have $v_3 = b$. Then the word $b^2$ is forbidden, and the path from $v_1$ to $v_3$ can not be labeled with a forbidden word.

For the graph $G_5$ exactly the same reasoning, as for $G_1$, is suitable.

For the last graph $G_6$ we need to examine a few cases. First, label the edges in accordance with Proposition 2.2.

Note that $v_3 \in \{a, ba, aba\}$; it can be shown repeating the argument for the graph $G_2$. We also have $v_1 \in \{\lambda, a, a^2\}$ by Propositions 2.1 and 2.2. The edge $(v_1, v_2)$ is forward by (i).

Case 1: $v_1 = \lambda$. Then $v_2 = b$, and $b^2$ is forbidden. The edge $(v_3, v_1)$ is backward for any value of $v_3$, whence $(v_3, v_4)$ is forward by (i). Then $v_3b \in AD$, which contradicts the definition of the antidictionary.

Case 2: $v_1 = a$. Then $v_2 = ab$, and $abb$ is forbidden. By Remark 4.1, $b^2$ is forbidden also. The argument of case 1 can be repeated to get a contradiction.

Case 3: $v_1 = a^2$. Then $v_2 = aab$, and $aabb \in AD$. One has $0 < |v_4| \leq 4$ by (ii) and (iii), whence $v_4 \in \{b, ab, bab, abab\}$ by Proposition 2.2. If $v_4 = b$ or $v_4 = ab$, the word $b^2$ is forbidden, so, we get a contradiction with the definition of the antidictionary. In two remaining cases one of the words $bab$, $ababb$ belongs to the antidictionary, and $AD$ can be reduced using the
We finished the examination of graphs and the proof of the theorem.

We conclude the paper with some remarks. First, the tables 1–3 give some evidence that the number of possible growth rates rapidly grows with the degree of the polynomial. A separate interesting problem is to determine, how fast this grow is. Is it exponential? Superexponential?

Another interesting problem concerns connected components of C-graphs. As we already have seen in the proof of Theorem 5.1, some C-graphs are not connected. Actually, a two-component C-graph can have only five (but not four, see the proof of Theorem 5.1) vertices, as for the antidictionary \{aabb, aabab\}. An interesting question is, whether the number of components can be arbitrarily large?

The third remark concerns the reversals of antidictionaries. The languages, antidictionaries of which are reversals of each other, are reversals of each other also. Hence, their complexity functions, and then growth rates, coincide. But the corresponding C-graphs are not necessarily isomorphic, or even cospectral! (Two graphs are cospectral, if their characteristic polynomials coincide.) The graphs 1.5 and 1.7 are cospectral, but not isomorphic, while the C-graph of the reversal of the graph 2.8 contains five vertices, whence their characteristic polynomials are of different degree.

The previous remark suggests the idea that one has to study some divisors of characteristic polynomials of C-graphs, rather than the characteristic polynomials themselves, to classify the growth rates. Such divisors should inherit the index of the graph. The tables 1–3 contains enough examples of this kind: for instance, the graphs 2.9, 2.10, 2.11 and 2.13 has different characteristic polynomials with the common divisor \(x^2 - x - 1\), which inherits their common growth rate.

In our last remark we consider two possible ways to find these essential divisors. First way uses the algebraic graph theory, namely, divisors of graphs. Some theory on divisors can be found in [3]. Here we only note that divisors are homomorphic images of the graphs with some additional properties. These properties yield, in particular, that characteristic polynomial of a graph is divisible by the ones of all its divisors.

The second way is based upon linear recurrent formulas for calculating complexity functions. Such formulas were used, for example, in [12]. The conjecture is that the characteristic polynomial of such a recurrence always divides the characteristic polynomial of the C-graph for the same antidictionary, inheriting the Frobenius root. We note that for some antidictionaries in the table 2 the characteristic polynomial of the recurrence divides the one of the C-graph properly.
References


