Theoretical Computer Science 410 (2009) 2811–2822

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Continuant polynomials and worst-case behavior of Hopcroft's minimization algorithm

Jean Berstel [a,*], Luc Boasson [b], Olivier Carton [b]

[a] IGM, Université Paris-Est Marne-la-Vallée, France
[b] Liafa, Université Denis Diderot, France

ARTICLE INFO

Keywords:
Hopcroft's algorithm
Automata minimization
Standard Sturmian sequence
Continuant polynomials

ABSTRACT

This paper is concerned with the analysis of the worst case behavior of Hopcroft's algorithm for minimizing deterministic finite state automata. We extend a result of Castiglione, Restivo and Sciortino. They show that Hopcroft's algorithm has a worst case behavior for the automata recognizing Fibonacci words. In a previous paper, we have proved that this holds for all standard Sturmian words having an ultimately periodic directive sequence (the directive sequence for Fibonacci words is $(1, 1, \ldots)$).

We prove here that the same conclusion holds for all standard Sturmian words having a directive sequence with bounded elements.

More precisely, we obtain in fact a characterization of those directive sequences for which Hopcroft's algorithm has worst case running time. These are the directive sequences $(d_1, d_2, d_3, \ldots)$ for which the sequence of geometric means $(d_1 d_2 \cdots d_n)^{1/n}$ is bounded. As a consequence, we easily show that there exist directive sequences for which the worst case for the running time is not attained.

1. Introduction

An algorithm for the minimization of deterministic finite state automata that runs in time $O(n \log n)$ on automata with n states was given by Hopcroft [22]. It is, up to now, the most efficient algorithm known in the general case, although it has been proved quite recently that Moore's partition refinement algorithm has an $O(n \log n)$ average running time [4].

Hopcroft's algorithm is based on Moore's algorithm. Another family of algorithms is based on fusion of states. Such an algorithm working in linear time for cycle-free automata was given by Revuz [26]. This algorithm has been extended to a more general class of automata by Almeida and Zeitoun [3]. It has been demonstrated in [5] that minimization by state fusion, which is not always possible, works well for local automata. Another dynamic state minimization algorithm based on fusion has been described in Watson's taxonomy [27]. There is a third algorithm, seemingly based neither on splitting nor on fusion of states, namely Brzozowski's minimization algorithm that works by determinization of the reversal of the automaton. In fact, there is a connection with splitting algorithms, as shown in [18].

We address here the problem of showing that the running time $O(n \log n)$ for Hopcroft's algorithm is tight. This algorithm has a degree of freedom because, in each step of its main loop, it allows a free choice of a set of states to be processed. Berstel and Carton [10] introduced a family of finite automata based on de Bruijn words, and they showed that there exist some "unlucky" sequence of choices that slow down the computation to achieve lower bound $\Omega(n \log n)$.

In the papers [15,16], Castiglione, Restivo and Sciortino replace de Bruijn words by Fibonacci words. They show that, for this word, and more generally for all circular standard Sturmian words, there is no more choice in Hopcroft's algorithm,

* Corresponding author. Tel.: +33 143319675.
E-mail address: berstel@univ-mlv.fr (J. Berstel).

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.01.039

so that there is a unique execution of Hopcroft's algorithm. They show that, for Fibonacci words, the unique execution of Hopcroft's algorithm runs in time $\Omega(n \log n)$, so that the worst-case behavior is achieved for the automata recognizing Fibonacci words. The computation is carried out explicitly, using connections between Fibonacci numbers and Lucas numbers. In [17], they give a detailed analysis of the reduction process that is the basis of their computation, and they show that this process is isomorphic, for all standard Sturmian words, to the refinement process in Hopcroft's algorithm.

The uniqueness of the execution of Hopcroft's algorithm comes from the fact that automata for Sturmian words are what we have called "slow automata" in another context [9].

In a previous paper [7], we have presented a generalization of the result of [16] to any sequence of Sturmian words that is constructed with an eventually periodic directive sequence. The computation required a detailed analysis of some particular systems of rational functions.

In this paper, we generalize the result to any sequence of Sturmian words that is constructed with a directive sequence with bounded elements. More precisely, we will get this result as a straightforward consequence of a characterization of those directive sequences for which Hopcroft's algorithm has worst case running time. These are the directive sequences $(d_1, d_2, d_3, \ldots)$ for which the sequence of geometric means $((p_n)^{1/n})_{n\geq 1}$, where $p_n = d_1 d_2 \cdots d_n$, is bounded.

The steps that lead to the equations describing the running time of Hopcroft's algorithm are similar to those in [16], but we get a usually infinite system of equations instead of a single one.

We need not solve the system explicitly. We give estimates for the coefficients of the series which are the solution of the system. This leads to consider continuants, as they are usually introduced for continued fractions [20].

Remark. Although the analysis of Hopcroft's algorithm is carried out for automata over a single letter input alphabet, we do not claim that this algorithm is the best one in this case. On the contrary, there is a well-known linear-time algorithm [24] for minimizing these automata. Moreover, we consider a particular class of these automata, composed of a unique cycle. For such an automaton, associated canonically to some binary word, it is easily seen that it is minimal if and only if the corresponding word is primitive. We consider standard Sturmian words which all are primitive, so we know a priori that all our automata are minimal. However, as soon as one considers only Hopcroft's algorithm, one needs $\Omega(n \log n)$ steps to check that the automata are minimal.

Outline. The paper is organized as follows. After some definitions, the first section sketches Hopcroft's automata minimization algorithm in the case we are interested in, namely for alphabets formed of a single letter. We then derive the system of equations for the generating function of the running time of the algorithm. The last section is concerned with the evaluation of these systems of equations.

2. Definitions and notation

In this section, we recall some definitions concerning finite Sturmian words, and fix conventions concerning circular occurrences of factors. Expository texts about Sturmian words are [12,2,25]. For finite Sturmian words, see [11].

Directive sequence. A *directive sequence* is a sequence $d = (d_1, d_2, d_3, \ldots)$ of positive integers. If d is eventually periodic and has period k, that is if $d_{n+k} = d_n$ for $n > \ell$, then one writes also $d = (d_1 \ldots, d_\ell, \overline{d_{\ell+1}, \ldots, d_{\ell+k}})$. In particular, if d is purely periodic and has period k, then one writes $d = (\overline{d_1, \ldots, d_k})$.

Standard words. A directive sequence d generates a sequence of words denoted $(s_n)_{n\geq 0}$. The words s_n are called the *standard words* generated by d. They are defined as follows:

$$s_0 = 1, \qquad s_1 = 0, \qquad s_{n+1} = s_n^{d_n} s_{n-1} \quad (n \geq 1).$$

Thus in particular $s_2 = 0^{d_1}1$ and $s_3 = (0^{d_1}1)^{d_2}0$. We observe that in the literature on Sturmian words, the first term of a directive sequence is frequently supposed to be only non-negative. We exclude the case $d_1 = 0$ for convenience. It amounts merely to exchange symbols 0 and 1 in the sequence generated by d.

Example 1 (*Fibonacci*). For $d = (1, 1, \ldots) = (\overline{1})$, one gets $s_{n+1} = s_n s_{n-1}$ for $n \geq 1$, and the standard words generated by d are the well-known Fibonacci words 1, 0, 01, 010, 01001, ...

Example 2. For $d = (\overline{2, 3})$, one gets $s_{n+1} = s_n^2 s_{n-1}$ if n is odd, and $s_{n+1} = s_n^3 s_{n-1}$ if n is even. The standard words generated by d are the words 1, 0, 001, 0010010010, 00100100100010010010001, ...

Example 3. For $d = (1, 2, 3, \ldots)$, one gets $s_{n+1} = s_n^n s_{n-1}$ for $n \geq 1$, and the standard words generated by d are the rapidly growing words 1, 0, 01, 01010, 01010010100101001, ...

Shift. The *shift* $\tau(d)$ of a directive sequence $d = (d_1, d_2, d_3, \ldots)$ is defined by

$$\tau(d) = \begin{cases} (d_1 - 1, d_2, d_3, \ldots) & \text{if } d_1 > 1 \\ (d_2, d_3, \ldots) & \text{otherwise.} \end{cases}$$

The terminology is justified by the following observation. Consider the infinite word $a^{d_1-1}ba^{d_2-1}b\cdots$. Then the shift $\tau(d)$ corresponds to the shift on this word, that is to the erasing of its first letter. If d is periodic and has period k, then $\tau^{d_1+\cdots+d_k}(d) = d$.

Example 4. For $d = (1, 1, \ldots) = (\overline{1})$, one gets $\tau(d) = d$. For $d = (\overline{2, 3})$, one gets $\tau(d) = (1, 3, \overline{2, 3})$, $\tau^2(d) = (3, \overline{2, 3})$, $\tau^3(d) = (2, \overline{2, 3})$, $\tau^4(d) = (1, \overline{2, 3})$, and $\tau^5(d) = d$.

Circular factors. Two words x and y are *conjugate* if there exist words u, v such that $x = uv$ and $y = vu$. We write $x \sim y$ when x and y are conjugate. It is useful to see a class of the conjugacy relation $\sim$ as a single word as drawn on a circle, with no distinguished origin.

A word u is a *circular factor* of a word w if it is a factor of some conjugate of w. This is equivalent for u to be a prefix of some conjugate of w. It is also equivalent to the conditions that $|u| \leq |w|$ and u is a factor of ww. The set of circular factors of a word is denoted by $CF(w)$.

The *number of circular occurrences* of u in w is denoted by $|w|_u$ and is the number of factorizations $ww = pus$ with $|p| < |w|$ and $|u| \leq |w|$. This is precisely the number of occurrences of u in w viewed as drawn on a circle.

Example 5. Consider the word $w = 01001$. The word $u = 10$ has two circular occurrences in word w since $ww = 0|10|0101001 = 0100|10|1001$, although the word 10 has three occurrences in the word ww.

Given a word w over the alphabet $\{0, 1\}$, a word u is a *special (circular) factor* of w if $u0$ and $u1$ are both (circular) factors of w.

We define the *weight* of w by

$$\|w\| = \sum_{u \in CF(w)} \min(|w|_{u0}, |w|_{u1}).$$

The only circular factors which contribute to $\|w\|$ are circular special factors. Two conjugate words have the same weight.

Example 6. Consider the word $w = 01001$. The circular special factors are here the words 0, 10 and 010. Each of the factors 00, 100 and 0100 has one occurrence in w, so $\|w\| = 3$.

3. Hopcroft's algorithm

Hopcroft [22] has given an algorithm that computes the minimal automaton of a given deterministic automaton. The running time of the algorithm is $O(|A| \times n \log n)$ where $|A|$ is the cardinality of the alphabet and n is the number of states of the given automaton. The algorithm has been described and re-described several times [21,1,6,13,23].

3.1. Outline

The algorithm is outlined in the function HopcroftMinimization given below, and is explained then in some more detail.

It is convenient to use the shorthand $T^c = Q \setminus T$ when T is a subset of the set Q of states. We denote by $\min(B, C)$ the set of smaller size of the two sets B and C, and either one of them if they have the same size.

```
1:  P ← {F, F^c}
2:  for all a ∈ A do
3:    Add((min(F, F^c), a), W)
4:  while W ≠ ∅ do
5:    (C, a) ← Some(W)        ▷ choose some element and remove it from W
6:    for each B ∈ P split by (C, a) do
7:      B', B'' ← Split(B, C, a)
8:      Replace B by B' and B'' in P
9:      for all b ∈ A do
10:       if (B, b) ∈ W then
11:         Replace (B, b) by (B', b) and (B'', b) in W
12:       else
13:         Add((min(B', B''), b), W)
```

Algorithm 1: HopcroftMinimization

Given a deterministic automaton $\mathcal{A}$, Hopcroft's algorithm computes the coarsest congruence which saturates the set F of final states. It starts from the partition $\{F, F^c\}$ which obviously saturates F and refines it until it gets a congruence. These refinements of the partition are always obtained by splitting some class into two classes.

Before explaining the algorithm in more detail, some notation is needed. For a set B of states, we note by $B \cdot a$ the set $\{q \cdot a \mid q \in B\}$. Let B and C be two sets of states and let a be a letter. We say that the pair (C, a) *splits* the set B if both sets $(B \cdot a) \cap C$ and $(B \cdot a) \cap C^c$ are nonempty. In that case, the set B is split into the two sets $B' = \{q \in B \mid q \cdot a \in C\}$ and $B'' = \{q \in B \mid q \cdot a \notin C\}$ that we call the *resulting sets*. Note that a partition $\{Q_1, \ldots, Q_n\}$ is a congruence if and only if for any $1 \leq i, j \leq n$ and any $a \in A$, the pair (Q_i, a) does not split Q_j.

The algorithm proceeds as follows. It maintains a current partition $\mathcal{P} = \{B_1, \ldots, B_n\}$ and a current set $\mathcal{W}$ of pairs (C, a) that remain to be processed, where C is a class of $\mathcal{P}$ and a is a letter. The set $\mathcal{W}$ is called the *waiting* set. The algorithm stops

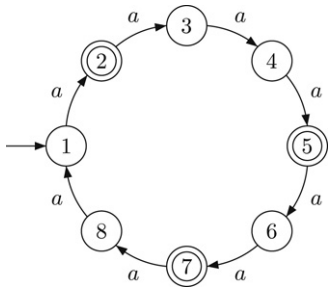

Fig. 1. Cyclic automaton $\mathcal{A}_w$ for $w = 01001010$.

when the waiting set $\mathcal{W}$ becomes empty. When it stops, the partition $\mathcal{P}$ is the coarsest congruence that saturates F. The starting partition is the partition $\{F, F^c\}$ and the starting set $\mathcal{W}$ contains all pairs $(\min(F, F^c), a)$ for $a \in A$.

The main loop of the algorithm takes one pair (C, a) out of the waiting set $\mathcal{W}$ and performs the following actions. Each class B of the current partition (including the class C) is checked whether it is split by the pair (C, a). If (C, a) does not split B, then nothing is done. Otherwise, the class B is replaced in the partition $\mathcal{P}$ by the two resulting sets B' and B'' of the split. For each letter b, if the pair (B, b) is in $\mathcal{W}$, it is replaced in $\mathcal{W}$ by the two pairs (B', b) and (B'', b), otherwise only the pair $(\min(B', B''), b)$ is added to $\mathcal{W}$.

The main ingredient in the analysis of the running time of the algorithm is that the splitting of all classes of the current partition according to a pair (C, a) takes a time proportional to the size of C. Therefore, the global running time of the algorithm is proportional to the sum of the sizes of the classes processed in the main loop. Note that a pair which is added to the waiting set $\mathcal{W}$ is not necessarily processed later because it can be split by the processing of another pair before it is considered.

It should be noted that the algorithm is not really deterministic because it has not been specified which pair (C, a) is taken from $\mathcal{W}$ to be processed at each iteration of the main loop. This means that for a given automaton, there are many executions of the algorithm. It turns out that all of them produce the right partition of the states. However, different executions may give rise to different sequences of splitting and also to different running times. Hopcroft has proved that the running time of any execution is bounded by $O(|A| \times n \log n)$.

3.2. Cyclic automata

The behavior of Hopcroft's algorithm is better understood on a family of automata that we introduce now, called *cyclic automata*.

Let $w = b_1 \cdots b_n$ be a word of length n over the binary alphabet $\{0, 1\}$. We define an automaton $\mathcal{A}_w$ over the unary alphabet $\{a\}$ as follows. The state set of $\mathcal{A}_w$ is $\{1, \ldots, n\}$ and the next state function is defined by $i \cdot a = i + 1$ for $i < n$ and $n \cdot a = 1$. Note that the underlying labeled graph of $\mathcal{A}_w$ is just a cycle of length n. The final states really depend on w. The set of final states of $\mathcal{A}_w$ is $F = \{1 \leq i \leq n \mid b_i = 1\}$.

For a binary word u, we define Q_u to be the set of states of $\mathcal{A}_w$ which are the starting positions of circular occurrences of u in w. If u is the empty word, then Q_u is by convention the set Q of all states of $\mathcal{A}_w$. By definition, the set F of final states of $\mathcal{A}_w$ is Q_1 while its complement F^c is Q_0.

Consider the automaton $\mathcal{A}_w$ for $w = 01001010$ given in Fig. 1. The sets Q_1, Q_{01} and Q_{11} of states are respectively $\{2, 5, 7\}$, $\{1, 4, 6\}$ and $\emptyset$.

Since cyclic automata are over the unary alphabet $\{a\}$, we say that a class C splits a class B when the pair (C, a) splits the pair (B, a).

The following statements appears in [16]:

Proposition 7. *Let w be a standard word. Hopcroft's algorithm on the cyclic automaton $\mathcal{A}_w$ is uniquely determined. In particular, at each step, the waiting set is a singleton.*

In fact, the execution is described in the next statement. The first part of the statement is from [14]. They prove that the number of circular (special) factors in fact characterizes standard words.

Proposition 8. *Let w be a standard word and set $m = |w|$. For $0 \leq i \leq m - 2$, the word w has exactly $i + 1$ circular factors of length i and exactly one circular special factor of length i.*

At each step i of the execution of Hopcroft's algorithm, the current partition is composed of the $i + 1$ classes Q_v indexed by the cyclic factors of length i, and the waiting set is a singleton. This singleton is the smaller of the sets Q_{u0}, Q_{u1}, where u is the unique cyclic special factor of length $i - 1$.

As already mentioned, the complexity of Hopcroft's algorithm is proportional to the sum of the sizes of the sets that are processed in the waiting set. As a consequence of the previous description, one gets the following corollary.

Corollary 9. *Let $d = (d_1, d_2, \ldots)$ be a directive sequence. Let $(s_n)_{n \geq 0}$ be the standard sequence defined by d. Then the complexity of Hopcroft's algorithm on the automaton $\mathcal{A}_{s_n}$ is proportional to $\|s_n\|$.*

Recall that $\|w\| = \sum_{u \in CF(w)} \min(|w|_{u0}, |w|_{u1})$. The main result of this paper is the following theorem. This is the basis for the description of those directive sequences for which Hopcroft's algorithm has a worst-case running time.

Theorem 10. *Let $d = (d_1, d_2, \ldots)$ be a directive sequence. Let $(s_n)_{n\geq 0}$ be the standard sequence defined by d. Then $\|s_n\| = \Theta(n|s_n|)$.*

This theorem is proved in [16] in the case of the Fibonacci words, for which the directive sequence is $d = (\overline{1})$.

4. The generating series of the complexity

Let $d = (d_1, d_2, \ldots)$ be a directive sequence. Let $(s_n)_{n\geq 0}$ be the standard sequence defined by d, and set

$$a_n = |s_n|_1, \qquad c_n = \|s_n\|.$$

By the definition of s_n, one gets, for $n \geq 1$,

$$a_{n+1} = d_n a_n + a_{n-1}. \tag{1}$$

These quantities depend of course on d, although this is not indicated in the notation. We have $a_0 = 1$, $a_1 = 0$, $a_2 = 1$ and $a_3 = d_2$ because $s_2 = 0^{d_1}1$ and $s_3 = (0^{d_1}1)^{d_2}0$. In particular, the value d_1 plays no role in the sequence $(a_n)_{n\geq 0}$. This will be used later.

Observe also that $c_0 = 0$, $c_1 = 0$ because $s_0 = 1$ and $s_1 = 0$, and that $c_2 = d_1$. Indeed, the circular special factors of s_2 are the words 0^ℓ for $0 \leq \ell < d_1$, and each factor $0^\ell 1$ for $0 \leq \ell < d_1$ has exactly one circular occurrence in s_2.

4.1. A first equation

The *generating series* $A_d(x)$ and $C_d(x)$ are defined by

$$A_d(x) = \sum_{n\geq 1} a_n x^n, \qquad C_d(x) = \sum_{n\geq 0} c_n x^n.$$

Note that the constant a_0 is not included in the series A_d. Observe also that the series A_d does not depend on the value of the first term d_1 of d.

Given a directive sequence $d = (d_1, d_2, \ldots)$, we define a shorthand notation $T(d)$ by $T(d) = \tau^{d_1}(d)$. Thus $T(d_1, d_2, d_3, \ldots) = (d_2, d_3, \ldots)$. Clearly $T^i(d) = (d_{i+1}, d_{i+2}, \ldots)$. We also define a *Kronecker symbol* δ by

$$\delta(d) = \begin{cases} 0 & \text{if } d_1 > 1, \\ 1 & \text{otherwise.} \end{cases}$$

The following equation results from the combinatorial study given below.

Proposition 11. *For any directive sequence $d = (d_1, d_2, \ldots)$, one has*

$$C_d(x) = A_d(x) + x^{\delta(d)} C_{\tau(d)}(x) + x^{1+\delta(T(d))} C_{\tau(T(d))}(x). \tag{2}$$

Example 12. Consider the directive sequence $d = (\overline{1})$ of the Fibonacci words. Since $\tau(d) = T(d) = d$, and $\delta(d) = 1$, Eq. (2) becomes

$$C_d(x) = A_d(x) + (x + x^2)C_d(x),$$

from which we get $C_d(x) = \frac{A_d(x)}{1-x-x^2}$. Clearly $a_{n+2} = a_{n+1} + a_n$ for $n \geq 0$, and since $a_0 = 1$ and $a_1 = 0$, one gets $A_d(x) = \frac{x^2}{1-x-x^2}$. Thus

$$C_d(x) = \frac{x^2}{(1-x-x^2)^2}.$$

4.2. Acceleration

Iterated application of Eq. (2) gives a system of equations which is finite when the directive sequence d is periodic. Indeed, call *suffix* of order m of d any of the directive sequences $\tau^m(d)$. If d is purely periodic with (minimal) period k, then $d = \tau^{d_1+\cdots+d_k}(d)$, so d has exactly $N = d_1 + \cdots + d_k$ distinct suffixes. Let U be this set of suffixes. Each of the suffixes u in U satisfies Eq. (2) with d replaced by u, so we get the following system of N equations in the variables $C_u(x)$, for $u \in U$:

$$C_u(x) = A_u(x) + x^{\delta(u)} C_{\tau(u)}(x) + x^{1+\delta(T(u))} C_{\tau(T(u))}(x).$$

Each of the C_u depends only linearly on $C_{\tau(u)}$ and $C_{\tau(T(u))}$, with coefficients which are monomials among $1, x, x^2$. We will show how to replace this system of N equations by a system of only k equations. This is done by collapsing appropriate equations of the previous system. We start with an example.

Example 13. Consider the directive sequence $d = (\overline{2, 3})$. Eq. (2) applied iteratively gives the following five equations:

$$\begin{aligned}
C_{(\overline{2,3})} &= A_{(\overline{2,3})} + C_{(1,\overline{3,2})} + xC_{(2,\overline{2,3})}\\
C_{(1,\overline{3,2})} &= A_{(1,\overline{3,2})} + xC_{(\overline{3,2})} + xC_{(2,\overline{2,3})}\\
C_{(2,\overline{2,3})} &= A_{(2,\overline{2,3})} + C_{(1,\overline{2,3})} + xC_{(1,\overline{3,2})}\\
C_{(\overline{3,2})} &= A_{(\overline{3,2})} + C_{(2,\overline{2,3})} + xC_{(1,\overline{3,2})}\\
C_{(1,\overline{2,3})} &= A_{(1,\overline{2,3})} + xC_{(\overline{2,3})} + xC_{(1,\overline{3,2})}.
\end{aligned}$$

There is no term with coefficient x^2 here because there is no $d_i = 1$ in the directive sequence. Observe that $A_{(\overline{2,3})} = A_{(1,\overline{3,2})}$ and $A_{(\overline{3,2})} = A_{(2,\overline{2,3})} = A_{(1,\overline{2,3})}$ because these series do not depend on the first term of the directive sequence. Setting $D_1 = C_{(1,\overline{3,2})}$ and $D_2 = C_{(2,\overline{2,3})}$, we get

$$C_{(\overline{2,3})} = A_{(\overline{2,3})} + D_1 + xD_2,$$

where D_1 and D_2 satisfy the equations

$$\begin{aligned}
D_1 &= A_{(\overline{2,3})} + xA_{(\overline{3,2})} + 2xD_2 + x^2D_1\\
D_2 &= 2A_{(\overline{3,2})} + xA_{(\overline{2,3})} + 3xD_1 + x^2D_2.
\end{aligned}$$

Thus the original system of 5 equations in the C_u is replaced by a system of 2 equations in D_1 and D_2.

Let $d = (d_1, d_2, \ldots)$ be a directive sequence, and for $i \geq 1$, set

$$e_i = T^{i-1}(d) = (d_i, d_{i+1}, \ldots), \qquad D_i = x^{\delta(e_i)}C_{\tau(e_i)}, \qquad B_i = (d_i - 1)A_{e_i} + xA_{e_{i+1}}.$$

With these notations, the following system of equation holds.

Proposition 14. *The following equations hold*

$$C_d = A_d + D_1 + xD_2 \tag{3}$$

$$D_i = B_i + d_ixD_{i+1} + x^2D_{i+2} \quad (i \geq 1). \tag{4}$$

Eqs. (3) and (4) hold for arbitrary, even non-periodic directive sequences. If a directive sequence d is periodic with period k, then $D_{k+1} = D_1$ and $D_{k+2} = D_2$, and the system is finite. Observe that the equations of the previous example have precisely this form.

4.3. Proof of the propositions

Proof of Proposition 14. Eq. (2) writes as

$$C_d = A_d + D_1 + xD_2,$$

as required. It is enough to prove (4) for $i = 1$, since indeed the equation for $i > 1$ reduces to the first one by replacing the directive sequence $(d_1, d_2, \ldots)$ by $(d_i, d_{i+1}, \ldots)$.

We distinguish two cases. Assume first $d_1 > 1$. For $m = 1, \ldots, d_1 - 1$, one has $\tau^m(d) = (d_1 - m, d_2, d_3, \ldots)$ and $A_{\tau^m(d)}(x) = A_d(x)$, since this series does not depend on d_1. Next one has $\delta(\tau^m(d)) = 0$ and $T(\tau^m(d)) = T(d) = (d_2, d_3, \ldots)$. Also, for $m = 1, \ldots, d_1 - 2$, one has $\delta(\tau^m(d)) = 0$ whereas $\delta(\tau^{d_1-1}(d)) = 1$. Eq. (2) gives, with d replaced by $\tau^m(d)$ for $m = 1, \ldots, d_1 - 1$

$$C_{\tau^m(d)} = A_{\tau^m(d)} + x^{\delta(\tau^m(d))}C_{\tau^{m+1}(d)} + x^{1+\delta(T(\tau^m(d)))}C_{\tau(T(\tau^m(d)))}(x).$$

It follows that for $m = 1, \ldots, d_1 - 2$,

$$C_{\tau^m(d)} = A_d + C_{\tau^{m+1}(d)} + xD_2,$$

and, for $m = d_1 - 1$,

$$C_{\tau^{d_1-1}(d)} = A_d + xC_{T(d)} + xD_2.$$

Summing these $d_1 - 1$ equations, we get

$$C_d = (d_1 - 1)A_d + xC_{T(d)} + (d_1 - 1)xD_2.$$

Since by (2) with d replaced by $T(d)$ we have

$$C_{T(d)} = A_{T(d)} + D_2 + xD_3,$$

substituting this expression gives

$$D_1 = C_{\tau(d)} = (d_1 - 1)A_d + xA_{T(d)} + d_1xD_2 + x^2D_3$$

as claimed.

Consider now the case $d_1 = 1$. Then $\tau(d) = T(d) = (d_2, d_3, \ldots)$, and

$$\begin{aligned} C_{\tau(d)} &= A_{\tau(d)} + x^{\delta(T(d))}C_{\tau(T(d))} + x^{1+\delta(T^2(d))}C_{\tau(T^2(d))} \\ &= B_1 + D_2 + xD_3. \end{aligned}$$

Consequently

$$D_1 = xC_{\tau(d)} = xB_1 + d_1xD_2 + x^2D_3.$$

This concludes the verification. □

We now turn to the derivation of expressions for the number of occurrences of circular special factors in the words generated by some directive sequence d in terms of the corresponding numbers associated to the directive sequence $\tau(d)$ and $T(d)$. For this, we will observe how circular special factors propagate through some particular morphisms.

Let $d = (d_1, d_2, \ldots)$ be a fixed directive sequence, and let $(s_n)_{n\geq 0}$ be the sequence of standard words generated by d. The following four lemmas hold. Lemmas 15 and 17 are similar to lemmas proved in [16], the proofs of Lemmas 16 and 18 are similar, so we prove only the last one.

Lemma 15. *Assume $d_1 = 1$, and let t_n be the sequence of standard words generated by $\tau(d) = (d_2, d_3, \ldots)$. Let φ be the morphism defined by $\varphi(0) = 01$ and $\varphi(1) = 0$. Then $s_{n+1} = \varphi(t_n)$ for $n \geq 1$. If v is a circular special factor of t_n, then $\varphi(v)0$ is a circular special factor of s_{n+1}. Conversely, if w is a circular special factor of s_{n+1} starting with 0, then w has the form $w = \varphi(v)0$ for some circular special factor v of t_n. Moreover, $|s_{n+1}|_{w0} = |t_n|_{v1}$ and $|s_{n+1}|_{w1} = |t_n|_{v0}$.*

Lemma 16. *Assume $d_1 > 1$, and let t_n be the sequence of standard words generated by $\tau(d) = (d_1 - 1, d_2, d_3, \ldots)$. Let ψ be the morphism defined by $\psi(0) = 0$ and $\psi(1) = 01$. Then $s_n = \psi(t_n)$ for $n \geq 1$. If v is a circular special factor of t_n, then $\psi(v)0$ is a circular special factor of s_n. Conversely, if w is a circular special factor of s_n starting with 0, then w has the form $w = \psi(v)0$ for some circular special factor v of t_n. Moreover, $|s_n|_{w0} = |t_n|_{v0}$ and $|s_n|_{w1} = |t_n|_{v1}$.*

Lemma 17. *Assume $d_2 = 1$, and let t_n be the sequence of standard words generated by $\tau T(d) = (d_3, d_4, \ldots)$. Let α be the morphism defined by $\alpha(0) = 10^{d_1+1}$ and $\alpha(1) = 10^{d_1}$. Then $s_{n+2}0^{d_1} = 0^{d_1}\alpha(t_n)$ for $n \geq 0$. If v is a circular special factor of t_n, then $\alpha(v)10^{d_1}$ is a circular special factor of s_{n+2}. Conversely, if w is a circular special factor of s_{n+2} starting with 1, then w has the form $w = \alpha(v)10^{d_1}$ for some circular special factor v of t_n. Moreover, $|s_{n+2}|_{w0} = |t_n|_{v0}$ and $|s_{n+2}|_{w1} = |t_n|_{v1}$.*

Lemma 18. *Assume $d_2 > 1$, and let t_n be the sequence of standard words generated by $\tau T(d) = (d_2 - 1, d_3, d_4, \ldots)$. Let β be the morphism defined by $\beta(0) = 10^{d_1}$ and $\beta(1) = 10^{d_1+1}$. Then $s_{n+1}0^{d_1} = 0^{d_1}\beta(t_n)$ for $n \geq 1$. If v is a circular special factor of t_n, then $\beta(v)10^{d_1}$ is a circular special factor of s_{n+1}. Conversely, if w is a circular special factor of s_{n+1} starting with 1, then w has the form $w = \beta(v)10^{d_1}$ for some circular special factor v of t_n. Moreover, $|s_{n+1}|_{w0} = |t_n|_{v1}$ and $|s_{n+1}|_{w1} = |t_n|_{v0}$.*

Proof. We first prove the relation between the s_n and the t_n. One has $0^{d_1}\beta(t_1) = 0^{d_1}\beta(0) = 0^{d_1}10^{d_1} = s_20^{d_1}$. Next

$$\begin{aligned} s_30^{d_1} &= (0^{d_1}1)^{d_2}0^{d_1+1} = 0^{d_1}1(0^{d_1}1)^{d_2-1}0^{d_1+1} = 0^{d_1}(10^{d_1})^{d_2-1}10^{d_1+1} \\ &= 0^{d_1}\beta(0^{d_2-1}1) = 0^{d_1}\beta(t_2) \end{aligned}$$

since by definition $t_2 = 0^{d_2-1}1$. By induction,

$$\begin{aligned} s_{n+1}0^{d_1} &= s_n^{d_n}s_{n-1}0^{d_1} = s_n^{d_n}0^{d_1}\beta(t_{n-2}) = \cdots \\ &= 0^{d_1}\beta(t_{n-1}^{d_n}t_{n-2}) = 0^{d_1}\beta(t_n) \end{aligned}$$

since by definition $t_n = t_{n-1}^{d_n}t_{n-2}$ for $n > 2$.

Next, let v be circular special factor of t_n. Then $v0$ and $v1$ are circular factors of t_n, and $|v0| = |v1| < |t_n|$ because if there were equality, then $v0$ and $v1$ would be conjugate words which is impossible since they do not have the same number of 0's. So $v00$ or $v01$ is a circular factor of t_n, and $\beta(v1) = \beta(v)10^{d_1+1}$ and one of $\beta(v00) = \beta(v)10^{d_1}10^{d_1}$ or $\beta(v01) = \beta(v)10^{d_1}10^{d_1+1}$ is a circular factor of s_{n+1}, so in both cases, $\beta(v)10^{d_1}1$ is a circular factor of s_{n+1}. This shows that $\beta(v)10^{d_1}$ is a special circular factor of s_{n+1}.

Conversely, let w be a circular special factor of s_{n+1} starting with 1. Then $w1$ is a circular factor of s_{n+1} so w ends with 10^{d_1} or 10^{d_1+1}, and since $w0$ is a circular factor of s_{n+1}, the second case is excluded because 0^{d_1+2} is not a factor of s_{n+1}. Thus $w = u10^{d_1}$ for some word u which is itself of the form $u = \beta(v)$ for some word v. Moreover, $w0 = \beta(v1)$ and $w = \beta(v0)$.

We show that $v0$ and $v1$ are circular factors of t_n. Indeed, every conjugate of s_{n+1} starting with the letter 1 is of the form $\beta t'$ for some word t' which is conjugate of t_n. We choose the conjugate of s_{n+1} starting with $w1$. Then $\beta(v1)$ is a prefix of $\beta(t')$ and $v1$ is a factor of t', so $v1$ is a circular factor of t_n. The same holds for $v0$. This shows that v is a special circular factor of t_n. The counting formulas are a consequence. □

Proof of Proposition 11. It will be convenient to refer explicitly to the directive sequence. Therefore, we write $c_n^{(d)}$ for c_n. Thus

$$c_n^{(d)} = \|s_n\| = \sum_{u \in CF(s_n)} \min(|s_n|_{u0}, |s_n|_{u1}).$$

Note that in this sum, only special circular factors contribute. We may therefore restrict the sum to special factors, and we consider three cases, namely whether a circular special factor starts with 0, starts with 1, or is the empty word. The last case is easy, since then the corresponding term is the smaller of the number of 0's or of 1's in s_n, and by the choice of $d_1 > 0$, this is always the number a_n of 1's in s_n. Thus, writing $c_{n,0}^{(d)}$ for the sum corresponding to special factors starting with 0 and $c_{n,1}^{(d)}$ for those starting with 1, one has

$$c_n^{(d)} = c_{n,0}^{(d)} + c_{n,1}^{(d)} + a_n.$$

Next, by Lemmas 15 and 16, one has $c_{n,0}^{(d)} = c_n^{(\tau(d))}$ if $d_1 > 1$ and $c_{n,0}^{(d)} = c_{n-1}^{(\tau(d))}$. Similarly, by Lemmas 17 and 18, one has $c_{n,1}^{(d)} = c_{n-1}^{(\tau T(d))}$ if $d_2 > 1$ and $c_{n,1}^{(d)} = c_{n-2}^{(\tau T(d))}$ otherwise. This proves the proposition. □

5. Evaluation of the complexity

5.1. Results

In this section, we derive bounds for the coefficients a_n and c_n of the series $A_d(x) = \sum a_n x^n$ and $C_d(x) = \sum c_n x^n$.

In the next statements, $d = (d_1, d_2, d_3, \ldots)$ denotes a directive sequence, $(s_n)_{n\geq 0}$ is the sequence of standard words generated by d, $a_n = |s_n|_1$ is the number of letters 1 in the word s_n, and $c_n = \|s_n\|$ is the running time of Hopcroft's algorithm on the automaton $\mathcal{A}_{s_n}$.

The first result is the following proposition that describes the dependence of c_n from a_n. This is precisely Theorem 10.

Proposition 19. *For any sequence d, one has $c_n = \Theta(na_n)$.*

We postpone the proof. We will get the bound $c_n = \Theta(a_n \log a_n)$ whenever $n = \Theta(\log a_n)$. The next proposition characterizes the directive sequences having this behavior.

Proposition 20. *One has $n = \Theta(\log a_n)$ and consequently $c_n = \Theta(a_n \log a_n)$ if and only if the sequence of geometric means $\left((d_1 d_2 \cdots d_n)^{1/n}\right)_{n\geq 1}$ of the directive sequence d is bounded.*

Proof. Since $a_{n+1} = d_n a_n + a_{n-1}$ and $a_3 = d_2$, one has

$$a_{n+1} \geq d_n a_n \geq \cdots \geq d_n \cdots d_4 a_4 \geq d_n \cdots d_2,$$

and since the sequence $(a_n)_{n\geq 1}$ is increasing, one has

$$a_{n+1} \leq (d_n + 1)a_n \leq \cdots \leq (d_n + 1)(d_{n-1} + 1) \cdots (d_2 + 1).$$

Set $p_n = d_2 d_3 \cdots d_{n-1} d_n$. Then $(d_n + 1)(d_{n-1} + 1) \cdots (d_2 + 1) \leq 2^n p_n$ and consequently

$$p_n \leq a_{n+1} \leq 2^n p_n.$$

If the sequence $(p_n^{1/n})$ is bounded, there exist numbers $k, k' > 0$ such that $k \leq p_n^{1/n} \leq k'$, and consequently

$$k^n \leq p_n \leq a_{n+1} \leq 2^n p_n \leq (2k')^n,$$

showing that $n = \Theta(\log a_n)$. Conversely, if $nk \leq \log a_n \leq nk'$, then $k^n \leq a_n \leq k'^n$ and $p_{n-1} \leq k'^n$ and $k^n \leq 2^{n-1} p_{n-1}$, showing that the sequence $(p_n^{1/n})$ is bounded. □

As a corollary, the relation $c_n = \Theta(a_n \log a_n)$ holds if the arithmetic mean $\frac{1}{n}(d_1 + \cdots + d_n)$ are bounded, since the geometric mean is always less than the arithmetic mean.

In particular, this relation holds when the elements of the sequence d themselves are bounded.

Observe also that there exist directive sequences d such that $c_n = O(a_n \log\log a_n)$. This holds if $d_n = 2^{2^n}$, since then $d_1 d_2 \cdots d_n = 2^{2^{n+1}-2}$ and consequently $a_n = \Theta(2^{2^{n+1}})$. Thus $n = \Theta(\log\log a_n)$. In fact, any running time close to a_n can be achieved by taking a rapidly growing directive sequence.

For the proof of Proposition 19, we will need some bounds for continuant polynomials.

5.2. Continuant polynomials

Continuant polynomials are used for continuant fractions, see for instance [20]. They are also used, in a noncommutative version, for the derivation of an analogue of Levi's lemma for polynomials, see for instance [19]. Here, we derive several upper and lower bounds that will be used later.

The *continuant polynomials* $K_n(x_1, \ldots, x_n)$, for $n \geq -1$ are a family of polynomials in the variables $x_1, \ldots, x_n$ defined by $K_{-1} = 0, K_0 = 1$ and, for $n \geq 1$, by

$$K_n(x_1, \ldots, x_n) = x_1 K_{n-1}(x_2, \ldots, x_n) + K_{n-2}(x_3, \ldots, x_n). \tag{5}$$

The first continuant polynomials are

$$K_1(x_1) = x_1$$
$$K_2(x_1, x_2) = x_1x_2 + 1$$
$$K_3(x_1, x_2, x_3) = x_1x_2x_3 + x_1 + x_3$$
$$K_4(x_1, x_2, x_3, x_4) = x_1x_2x_3x_4 + x_1x_2 + x_3x_4 + x_1x_4 + 1.$$

It can be shown that for $n \geq 1$, one has also the following recurrence formula.

$$K_n(x_1, \ldots, x_n) = K_{n-1}(x_1, \ldots, x_{n-1})x_n + K_{n-2}(x_1, \ldots, x_{n-2}). \tag{6}$$

These polynomials are related to continued fractions as follows (see e.g. [20]). Let $d = (d_1, d_2, d_3, \ldots)$ be a sequence of positive numbers. The *continued fraction* defined by d is denoted $\alpha = [d_1, d_2, d_3, \ldots]$ and is defined by

$$\alpha = d_1 + \frac{1}{d_2 + \frac{1}{d_3 + \cdots}}.$$

The finite initial parts $[d_1, d_2 \ldots, d_n]$ of d define rational numbers

$$d_1 + \frac{1}{d_2 + \frac{1}{d_3 + \cdots + \frac{1}{d_n}}} = \frac{K_n(d_1, \ldots, d_n)}{K_{n-1}(d_2, \ldots, d_n)}.$$

From (6), it follows for the standard words s_n defined by d that for $n \geq 3$

$$|s_n|_0 = K_{n-1}(d_1, \ldots, d_{n-1}), \qquad a_n = |s_n|_1 = K_{n-2}(d_2, \ldots, d_{n-1}).$$

It follows that

$$\alpha = \lim_{n\to\infty} \frac{|s_n|_0}{|s_n|_1},$$

and also $|s_n| = \Theta(|s_n|_1)$.

We now use these continuant polynomials to give parametrized expressions for the coefficients of the series A_d, B_i and D_i.

Lemma 21. *One has*

$$a_{n+2} = K_n(d_2, \ldots, d_{n+1}) \quad (n \geq -1) \tag{7}$$

and

$$A_d(x) = x^2 \sum_{n\geq 0} K_n(d_2, \ldots, d_{n+1})x^n. \tag{8}$$

Proof. One has $a_{n+1} = d_na_n + a_{n-1}$ for $n \geq 2$ and $a_0 = 1, a_1 = 0, a_2 = 1$. Thus $a_1 = K_{-1}$, $a_2 = K_0$, $a_3 = K_1(d_2)$ and (7) follows from (5). The expression for the series A_d follows. □

We now give a similar description of the series B_i and D_i. Define L_n by

$$L_n(x_1, \ldots, x_n) = K_n(x_1, \ldots, x_n) - K_{n-1}(x_2, \ldots, x_n). \tag{9}$$

Lemma 22. *For $i \geq 1$, one has*

$$B_i = x^2 \sum_{n\geq 0} \Big(K_{n+1}(d_i, \ldots, d_{n+i}) - K_n(d_{i+1}, \ldots, d_{n+i})\Big)x^n$$
$$= x^2 \sum_{n\geq 0} L_{n+1}(d_i, \ldots, d_{i+n})x^n.$$

Proof. Indeed, by definition one has $B_i = (d_i - 1)A_{e_i} + xA_{e_{i+1}}$, and using (8), applied to $(d_i, d_{i+1}, \ldots)$, one gets

$$A_{e_i}(x) = x^2 \sum_{n\geq 0} K_n(d_{1+i}, \ldots, d_{n+i})x^n,$$

and thus

$$B_i(x) - A_d(x) = x^2 \sum_{n\geq 0} \big(d_iK_n(d_{1+i}, \ldots, d_{n+i}) + K_{n-1}(d_{1+i}, \ldots, d_{n+i-1})\big)x^n,$$

which, by Eq. (5) applied to the values $d_i, d_{1+i}, \ldots, d_{n+i}$ gives the expression

$$B_i(x) - A_d(x) = x^2 \sum_{n\geq 0} K_{n+1}(d_i, \ldots, d_{n+i})x^n.$$

The expression for B_i follows. □

Finally, set

$$N_n(x_1, \ldots, x_n) = \sum_{i=0}^{n-1} K_i(x_1, \ldots, x_i)L_{n-i}(x_{i+1}, \ldots, x_n). \tag{10}$$

We now show that

Lemma 23. *One has*

$$D_1 = x^2 \sum_{n\geq 0} N_{n+1}(d_1, \ldots, d_{n+1})x^n,$$

and more generally

$$D_i = x^2 \sum_{n\geq 0} N_{n+1}(d_i, \ldots, d_{n+i})x^n.$$

Proof. We prove that for $i, m \geq 1$,

$$D_i = \sum_{\ell=0}^{m-1} K_\ell B_{i+\ell} + K_m x^m D_{i+m} + K_{m-1}x^{m+1}D_{i+m+1}$$

where we write K_h for $K_h(d_i, \ldots, d_{i+h-1})$. Indeed, for $m = 1$, this is (4), and by induction, substituting D_{i+m+1} we get

$$\begin{aligned} D_i &= \sum_{\ell=0}^{m-1} K_\ell x^\ell B_{i+\ell} + K_m x^m (B_{i+m} + d_{i+m}xD_{i+m+1} + x^2 D_{i+m+2}) + K_{m-1}x^{m+1}D_{i+m+1} \\ &= \sum_{\ell=0}^{m} K_\ell x^\ell B_{i+\ell} + (K_m d_{i+m}x^{m+1} + K_{m-1}x^{m+1})D_{i+m+1} + K_m x^{m+2}D_{i+m+2} \\ &= \sum_{\ell=0}^{m} K_\ell x^\ell B_{i+\ell} + K_{m+1}x^{m+1}D_{i+m+1} + K_m x^{m+2}D_{i+m+2}. \end{aligned}$$

It follows that, when $m \to \infty$,

$$D_1 = \sum_{\ell\geq 0} K_\ell(d_1, \ldots, d_\ell)x^\ell B_{\ell+1}.$$

Thus the coefficient of x^{2+n} in D_1 is as announced. □

Observe finally that the series C_d also has an expression with continuants since

$$C_d = x^2 \sum_{n\geq 0} (K_n(d_2, \ldots, d_{n+1}) + N_{n+1}(d_1, \ldots, d_{n+1}) + N_n(d_2, \ldots, d_{n+1}))x^n. \tag{11}$$

5.3. Bounds for continuant polynomials

The following bounds on the values of continuant polynomials will be used in Proposition 25.

Proposition 24. *The following inequalities hold for any sequence $(d_1, d_2, d_3, \ldots)$ of positive integers:*

(a) $K_{n-1}(d_2, \ldots, d_n) \leq K_n(d_1, \ldots, d_n) \leq (1 + d_1)K_{n-1}(d_2, \ldots, d_n)$.
(b) *For $n \geq 2$*

$$L_n(d_1, \ldots, d_n) \geq \begin{cases} \frac{1}{2+d_2}K_n(d_1, \ldots, d_n) & \text{if } d_1 = 1 \\ \frac{1}{2}K_n(d_1, \ldots, d_n) & \text{otherwise.} \end{cases}$$

(c) *For $n, m \geq 0$, one has $\frac{1}{2}K_{n+m}(d_1, \ldots, d_{n+m}) \leq K_n(d_1, \ldots, d_n)K_m(d_{n+1}, \ldots, d_{n+m}) \leq K_{n+m}(d_1, \ldots, d_{n+m})$.*

Proof. We use $K_{h,i}$ as a shorthand for $K_{h,i} = K_h(d_i, \ldots, d_{i+h-1})$, and similarly for L.

(a) By (5), one has $K_{n-1,2} \leq K_{n,1}$, and $K_{n,1} = d_1K_{n-1,2} + K_{n-2,3} \leq d_1K_{n-1,2} + K_{n-1,2} = (1 + d_1)K_{n-1,2}$.

(b) If $d_1 = 1$, then $K_{n,1} = K_{n-1,2} + K_{n-2,3} \leq (2 + d_2)K_{n-2,3}$ by use of (a) for $n - 1$. It follows that $L_{n,1} = K_{n,1} - K_{n-1,2} = K_{n-2,3} \geq 1/(2 + d_2)K_{n,1}$.

Assume now $d_1 \geq 2$. Since $d_1 - 1 \geq d_1/2$, we have $2L_{n,1} = 2(d_1 - 1)K_{n-1,2} + 2K_{n-2,3} \geq d_1K_{n-1,2} + K_{n-2,3} = K_{n,1}$.

For (c), we use the following identity which is easy to prove and which is also Equation (6.133) in [20]:

$$K_{n+m,1} = K_{n,1}K_{m,n+1} + K_{n-1,1}K_{m-1,n+2}.$$

Since $K_{n-1,1}K_{m-1,n+2} \leq K_{n,1}K_{m,n+1}$, one gets $K_{n+m,1} \leq 2K_{n,1}K_{m,n+1}$ which gives the first inequality. One gets the second by observing in the equation, the second term in the right-hand side satisfies $K_{n-1,1}K_{m-1,n+2} \geq 0$, and so $K_{n+m,1} \geq K_{n,1}K_{m,n+1}$. □

The inequalities just obtained allow us to give bounds on the coefficients of the series D_i:

Proposition 25. *For $n \geq 1$, one has*

$$N_n(d_1, \ldots, d_n) \leq nK_n(d_1, \ldots, d_n),$$

and for $n \geq 2$, one has

$$N_n(d_1, \ldots, d_n) \geq \frac{n-1}{12} K_n(d_1, \ldots, d_n).$$

Proof. Since $L_n(d_1, \ldots, d_n) \leq K_n(d_1, \ldots, d_n)$, we get by Eq. (9) the inequality $N_n(d_1, \ldots, d_n) \leq \sum_{i=0}^{n-1} K_i(d_1, \ldots, d_i)K_{n-i}(d_{i+1}, \ldots, d_n)$. By Proposition 24(c), it follows that $N_n(d_1, \ldots, d_n) \leq nK_n(d_1, \ldots, d_n)$.

For the other inequality, consider the partition of the set $J = \{1, 2, \ldots, n\}$ into the three sets $I = \{i \in J : d_i > 1\}$, $I' = \{i \in J : i < n, d_i = d_{i+1} = 1\}$, and $I'' = J \setminus (I \cup I')$. Observe if $i \in I''$ and $i < n$, then $d_{i+1} \in I$. This shows that $\mathrm{Card}(I'') \leq \mathrm{Card}(I) + 1$, and consequently that $\mathrm{Card}(I \cup I') \geq (n-1)/2$.

According to Proposition 24(b), one has

$$L_{n-i}(d_{i+1}, \ldots, d_n) \geq 1/2 K_{n-i}(d_{i+1}, \ldots, d_n) \quad \text{for } i+1 \in I,$$

and

$$L_{n-i}(d_{i+1}, \ldots, d_n) \geq 1/3 K_{n-i}(d_{i+1}, \ldots, d_n) \quad \text{for } i+1 \in I'.$$

It follows that

$$\begin{aligned} N_n(d_1, \ldots, d_n) &\geq \frac{1}{2} \sum_{i+1 \in I} K_i(d_1, \ldots, d_i)K_{n-i}(d_{i+1}, \ldots, d_n) + \frac{1}{3} \sum_{i+1 \in I'} K_i(d_1, \ldots, d_i)K_{n-i}(d_{i+1}, \ldots, d_n) \\ &\geq \frac{1}{4} \sum_{i+1 \in I} K_n(d_1, \ldots, d_n) + \frac{1}{6} \sum_{i+1 \in I'} K_n(d_1, \ldots, d_n) \\ &\geq \frac{1}{6} \sum_{i+1 \in I \cup I'} K_n(d_1, \ldots, d_n) \geq \frac{n-1}{12} K_n(d_1, \ldots, d_n), \end{aligned}$$

as claimed. □

We can now derive easily the estimation of the complexity of Hopcroft's algorithm stated in Proposition 19.

Proof of Proposition 19. By Eq. (11), one has $c_n = \Theta(K_{n-2}(d_2, \ldots, d_{n-1}) + N_{n-1}(d_1, \ldots, d_{n-1}) + N_{n-2}(d_2, \ldots, d_{n-1}))$. By Proposition 25, one has $N_n(d_1, \ldots, d_n) = \Theta(nK_n(d_1, \ldots, d_n))$. Thus $c_n = \Theta(nK_{n-1}(d_1, \ldots, d_{n-1}) + nK_{n-2}(d_1, \ldots, d_{n-2})) = \Theta(nK_{n-1}(d_1, \ldots, d_{n-1}))$. Finally, by Eq. (7), one gets $a_n = \Theta(K_{n-1}(d_1, \ldots, d_{n-1}))$. □

6. Conclusion

We have characterized a family of automata over a single letter alphabet that have a worst-case behavior for Hopcroft's minimization algorithm. These automata are defined by means of standard Sturmian words.

One may ask whether there exist families of automata over more than one letter that have the same behavior. The answer is in fact positive: There exist automata over a binary alphabet with worst-case behavior for Hopcroft's algorithm. These automata were discovered during the investigation of Sturmian trees [9]. A paper with the proof of their behavior is in preparation [8].

Acknowledgments

We thank Isabelle Fagnot for her helpful comments, and Wolfgang Steiner for his suggestion to look at arithmetic means. We also thank the reviewers for their recommendations.

References

[1] A. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.
[2] J.-P. Allouche, J. Shallit, Automatic Sequences, Cambridge University Press, Cambridge, 2003.
[3] J. Almeida, M. Zeitoun, Description and analysis of a bottom-up DFA minimization algorithm, Inform. Process. Lett. 107 (2) (2008) 52–59.
[4] F. Bassino, J. David, C. Nicaud, On the average complexity of Moore's state minimization algorithm, in: Symp. Theoret. Aspects Comput. Sci., STACS 2009, Freiburg, February 26–28 2009 (in press).
[5] M.-P. Béal, M. Crochemore, Minimizing local automata, in: G. Caire, M. Fossorier (Eds.), IEEE International Symposium on Information Theory, ISIT'07, IEEE, 2007, pp. 1376–1380. number 07CH37924C.
[6] D. Beauquier, J. Berstel, P. Chrétienne, Éléments d'algorithmique, Masson, 1992.
[7] J. Berstel, L. Boasson, O. Carton, Hopcroft's automaton minimization algorithm and Sturmian words, in: Discrete Mathematics and Theoretical Computer Science 2008 (Fifth Coll. Math. Comput. Sci.), in: DMTCS Proc. Series, vol. AI, 2008, pp. 355–366.

[8] J. Berstel, L. Boasson, O. Carton, Slow automata and Hopcroft's minimization algorithm, 2009, (in preparation).
[9] J. Berstel, L. Boasson, O. Carton, I. Fagnot, A first investigation of Sturmian trees, in: W. Thomas, P. Weil (Eds.), STACS'2007, in: Lect. Notes in Comput. Sci., vol. 4393, Springer Verlag, 2007, pp. 73–84.
[10] J. Berstel, O. Carton, On the complexity of Hopcroft's state minimization algorithm, in: Implementation and Application of Automata, in: Lect. Notes in Comput. Sci., vol. 3317, Springer-Verlag, 2004, pp. 35–44.
[11] J. Berstel, A. Lauve, F. Saliola, C. Reutenauer, Combinatorics on Words: Christoffel Words and Repetitions in Words, in: CRM Monograph Series, vol. 27, American Mathematical Society, 2008.
[12] J. Berstel, P. Séébold, Sturmian words, in: M. Lothaire (Ed.), Algebraic Combinatorics on Words, in: Encyclopedia of Mathematics and its Applications, vol. 90, Cambridge University Press, 2002, pp. 45–110. chapter 2.
[13] N. Blum, A $O(n \log n)$ implementation of the standard method for minimizing n-state finite automata, Inform. Proc. Lett. 57 (1996) 65–69.
[14] J.-P. Borel, C. Reutenauer, On Christoffel classes, Theor. Inform. Appl. 40 (1) (2006) 15–27.
[15] G. Castiglione, A. Restivo, M. Sciortino, Circular words and automata minimization, in: P. Arnoux, N. Bédaride, J. Cassaigne (Eds.), Words 2007, Institut de Mathématiques de Luminy, 17–21 september 2007, pp. 79–89.
[16] G. Castiglione, A. Restivo, M. Sciortino, Hopcroft's algorithm and cyclic automata, in: C. Martín-Vide, F. Otto, H. Fernau (Eds.), Language and Automata Theory and Applications, Second International Conference, LATA 2008, in: Lect. Notes in Comput. Sci., vol. 5196, Springer-Verlag, 2008, pp. 172–183. Tarragona, Spain, March 13–19, 2008. Revised Papers.
[17] G. Castiglione, A. Restivo, M. Sciortino, Circular Sturmian words and Hopcroft's algorithm, Theoret. Comput. Sci. (submitted for publication).
[18] J.-M. Champarnaud, A. Khorsi, T. Paranthoën, Split and join for minimizing: Brzozowski's algorithm, in: M. Balik, M. Simanek (Eds.), Prague Stringology Conference 2002, Research report DC-2002-03, Czech Technical University of Prague, 2002, pp. 96–104.
[19] P.M. Cohn, Free associative algebras, Bull. London Math. Soc. 1 (1969) 1–39.
[20] R.L. Graham, D.E. Knuth, O. Patashnik, Concrete Mathematics, Second ed., Addison-Wesley, 1994.
[21] D. Gries, Describing an algorithm by Hopcroft, Acta Inform. 2 (1973) 97–109.
[22] J.E. Hopcroft, An $n \log n$ algorithm for minimizing states in a finite automaton, in: Z. Kohavi, A. Paz (Eds.), Theory of Machines and Computations, Academic Press, 1971, pp. 189–196.
[23] T. Knuutila, Re-describing an algorithm by Hopcroft, Theoret. Comput. Sci. 250 (2001) 333–363.
[24] R. Paige, R. E. Tarjan, R. Bonic, A linear time solution for the single function coarsest partition problem, Theoret. Comput. Sci. 40 (1) (1985) 67–84.
[25] N. Pytheas Fogg, Substitutions in Dynamics, Arithmetics and Combinatorics, in: V. Berthé, S. Ferenczi, C. Mauduit, A. Siegel (Eds.), Lecture Notes in Mathematics, vol. 1794, Springer Verlag, 2002.
[26] D. Revuz, Minimisation of acyclic deterministic automata in linear time, Theoret. Comput. Sci. 92 (1992) 181–189.
[27] B. Watson, A taxonomy of finite automata minimization algorithms, Technical Report 93/44, Faculty of Mathematics and Computing Science, Eindhove Univeerisity of Technology, 1944.