Theoretical Computer Science 410 (2009) 4372–4381

Contents lists available at ScienceDirect

Theoretical Computer Science

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

Circular sturmian words and Hopcroft's algorithm

G. Castiglione, A. Restivo *, M. Sciortino

University of Palermo, Dipartimento di Matematica ed Applicazioni, Via Archirafi 34, 90123 Palermo, Italy

ARTICLE INFO

Keywords:
Deterministic finite state automata
Hopcroft's minimization algorithm
Circular sturmian words
Sturmian morphisms

ABSTRACT

In order to analyze some extremal cases of Hopcroft's algorithm, we investigate the relationships between the combinatorial properties of a circular sturmian word (x) and the run of the algorithm on the cyclic automaton $\mathcal{A}_x$ associated to (x). The combinatorial properties of words taken into account make use of sturmian morphisms and give rise to the notion of *reduction tree* of a circular sturmian word. We prove that the shape of this tree uniquely characterizes the word itself. The properties of the run of Hopcroft's algorithm are expressed in terms of the *derivation tree* of the automaton, which is a tree that represents the refinement process that, in the execution of Hopcroft's algorithm, leads to the coarsest congruence of the automaton. We prove that the shape of the reduction tree of a circular sturmian word (x) coincides with that of the derivation tree $\mathcal{T}(\mathcal{A}_x)$ of the automaton $\mathcal{A}_x$. From this we derive a recursive formula to compute the running time of Hopcroft's algorithm on the automaton $\mathcal{A}_x$, expressed in terms of parameters of the reduction tree of (x). As a special application, we obtain the time complexity $\Theta(n \log n)$ of the algorithm in the case of automata associated to Fibonacci words.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

Minimization of deterministic finite automata is a largely studied problem of the theory of Automata and Formal Languages. It consists in finding the unique (up to isomorphism) minimal deterministic automaton recognizing a regular language.

The most efficient known algorithm was given by Hopcroft [15] and it runs in time $O(n \log n)$.

In [3], in order to analyze some extremal cases of Hopcroft's algorithm, Berstel and Carton introduced a family of unary automata associated to circular words. The circular words taken into account in [3] are the de Bruijn words and, by using the associated automata, it is shown that the complexity $O(n \log n)$ of the algorithm is tight.

Hopcroft's 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. In the family of automata in [3] it is shown that there exists some "unlucky" sequence of choices that slow down the computation to achieve the lower bound $\Omega(n \log n)$. However, there are also executions that run in linear time for the same automata. The authors of [3] leave open the problem whether there are automata on which all the executions of Hopcroft's algorithm do not run in linear time.

In [8] the authors of the present paper replace the de Bruijn words by circular Fibonacci words, which are particular circular sturmian words. They first show that, for automata associated to circular sturmian words, there is no more choice in Hopcroft's algorithm. Moreover, they give an answer to the open problem in [3], by proving that the unique execution of Hopcroft's algorithm on automata associated to circular Fibonacci words runs in time $\Theta(n \log n)$. The result is obtained by the exact computation of the running time of the algorithm, expressed in terms of the Fibonacci convolution sequence. Very

* Corresponding author.
E-mail addresses: giusi@math.unipa.it (G. Castiglione), restivo@math.unipa.it (A. Restivo), mari@math.unipa.it (M. Sciortino).

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.07.018

recently in [2] Berstel, Boasson and Carton presented a generalization of this result to any sequence of circular sturmian words that is constructed with an eventually periodic directive sequence.

In the present paper we deepen the relationships between the structure of circular sturmian words and the runs of Hopcroft's algorithm on the associated cyclic automata.

In the first part of the paper we investigate some combinatorial properties of circular sturmian words. In particular, by extending some results of [5], we study the action of sturmian morphisms on circular sturmian words. This part of the paper culminates on the definition of *reduction tree* which, roughly speaking, describes how each circular sturmian word is generated by composing some basic sturmian morphisms. The shape of such a tree (i.e. the corresponding unlabeled tree) uniquely characterizes the word and highlights some structural properties of the word itself.

In the second part of the paper we study the runs of Hopcroft's algorithm on cyclic automata associated to circular sturmian words. In order to analyze the behavior of the algorithm on $\mathcal{A}$, we use the notion of *derivation tree* $\mathcal{T}(\mathcal{A})$ of an automaton $\mathcal{A}$, introduced by Knuutila [17]. It is a tree that represents the refinement process that, during the execution of Hopcroft's algorithm, leads from the initial partition to the coarsest congruence of the automaton. Our main result states that the shape of the derivation tree of the automaton $\mathcal{A}_x$ associated to a circular sturmian word (x) coincides with the shape of the reduction tree of the word itself. From this we derive a recursive formula to compute the running time of Hopcroft's algorithm on the automaton $\mathcal{A}_x$, expressed in terms of parameters of the reduction tree of (x). The time complexity $\Theta(n \log n)$ of the algorithm in the case of Fibonacci words is then obtained as a special application.

2. Circular sturmian words

Let A be a finite alphabet and v, u be two words in A^*. We say that v and u are conjugate if for some words $z, w \in A^*$ one has that $v = zw$ and $u = wz$. It is easy to see that conjugation is an equivalence relation. Note that many combinatorial properties of words in A^* can be thought as properties of the respective conjugacy classes. So, in order to investigate some structural properties of conjugacy classes of words, in this section we consider a conjugacy class of words as a *circular word*. In particular, we denote by (w) the circular word corresponding to all the conjugates of the word w.

A word w in A^* is called primitive if all its conjugates are distinct. In this case we say that the circular word (w) is *primitive*. For instance, it is easy to verify that the circular word $(bcabcabca)$ is not primitive, while $(abaab)$ is primitive.

We say that a word $v \in A^*$ is a *factor* of a circular word (w) if v is a factor of some conjugate of w. Equivalently, a factor of (w) is a factor of ww of length not greater that $|w|$. Note that, while each factor of w is also a factor of (w), there exist circular words (w) having factors that are not factors of w. For instance, ca is a factor of (abc) without being factor of abc. One can give a notion of balancing for circular words. A circular word (w) is called *balanced* if for each u, v factors of (w), with $|u| = |v|$, and for each $a \in A$ one has that $||u|_a - |v|_a| \leq 1$. Remark that, w balanced (in the sense of linear words) does not imply that (w) is a balanced circular word. For instance, 1001 is balanced and (1001) is not balanced.

The following proposition can be easily obtained from a result in [6].

Proposition 1. *Let w be a word of length $n \geq 2$. The following statements are equivalent:*

(1) (w) *is primitive;*
(2) *for* $k = 0, \ldots, n-1$ *the circular word* (w) *has at least* $k+1$ *factors of length* k*;*
(3) (w) *has* n *factors of length* $n-1$.

We can define analogously the notion of special factor of a circular binary word. Given a circular word (w) defined over the binary alphabet $\{0, 1\}$, we say that u is a *special factor* of (w) if both $u0$ and $u1$ are factors of (w). For instance, 00 is a special factor of (01001100) because both 001 and 000 are factors.

In this section we focus on a particular class of circular words, called circular sturmian words. These have newly aroused lively interest in the field of Combinatorics on Words. The main paper on this topic have been very recently published (cf. [6]) where such words are named Christoffel classes. Such finite words inherit many structural properties of the infinite sturmian words. Nevertheless, in spite of their strong connection to the well known notions of infinite Sturmian words and standard words, they have different and distinctive combinatorial properties.

A word w on a binary alphabet is called a *Christoffel word* if it is obtained by discretizing a segment in the lattice $\mathbb{N} \times \mathbb{N}$ (cf. [6]). Given the pair of coprime integers p and q and the segment from the point $(0, 0)$ to the point (p, q), the Christoffel word having p occurrences of 0's and q occurrences of 1's is obtained by considering the path under the segment and by coding by 0 a horizontal step and by 1 a vertical step. Such words are conjugate of standard words (cf. [18, chap. 2]).

We say that a circular word, defined over a binary alphabet, is a *Christoffel class*, or equivalently a *circular sturmian word*, if some word in its conjugacy class is a Christoffel word. For instance, the circular word (10010) is sturmian because the 00101 is a Christoffel word.

The following proposition provides some characterizations of circular sturmian words (cf. [6,16]).

Proposition 2. *Let w a word of length $n \geq 2$. The following statements are equivalent:*

(1) (w) *is a circular sturmian word;*
(2) *for* $k = 0, \ldots, n-1$ *the circular word* (w) *has exactly* $k+1$ *factors of length* k*;*
(3) (w) *has* $n-1$ *factors of length* $n-2$ *and* w *is primitive;*
(4) (w) *is balanced.*

From such a proposition one can easily derive the following

Corollary 3. *Let w be a word of length n. The circular word (w) is sturmian if and only if for each $k = 0, \ldots, n-2$ there exists a unique special factor of (w) of length k.*

A circular word can be described as a necklace in which one can read the circular word in a fixed clockwise direction. Since the notion of balanced circular word is independent from any direction, an immediate consequence of the previous proposition is the following corollary.

Corollary 4. *If v is a Christoffel word and v^R its reverse, then $(v) = (v^R)$.*

The next corollary shows that sturmian circular words keep a structure that is a finite version of the structure of infinite sturmian words (cf. [12]).

Corollary 5. *If (w) is a sturmian circular word with $|w|_0 \geq |w|_1$ (resp. $|w|_1 > |w|_0$) then either $w = 0$ (resp. $w = 1$) or there exists an integer $p \geq 0$ such that (w) is a concatenation of the factors 10^p and 10^{p+1} (resp. 01^p and 01^{p+1}).*

Proof. Let us consider the case when $|w|_0 \geq |w|_1$. If $|w| = 1$ the result is trivial. We can suppose that $|w| > 1$. We have to prove that the number of 0's between two consecutive 1's in (w) is either p or $p+1$. Let us suppose, by contradiction, that (w) has factors 10^q1 and 10^r1 for some integers q, r with $q < r$ and $r - q \geq 2$. Then, the words $u = 10^q1$ and $v = 0^{q+2}$ are factors of (w) of the same length such that $|u|_1 - |v|_1 \geq 2$, against the hypothesis that (w) is balanced. □

We call the *signature* of a circular sturmian word (w) the integer $\sigma(w) = \lfloor \frac{\max\{|w|_0,|w|_1\}}{\min\{|w|_0,|w|_1\}} \rfloor$. Note that the integer p in Corollary 5 is equal to $\sigma(w)$. Remark that, $\sigma(w) = \sigma(\overline{w})$, where $\overline{w}$ is the word obtained from w by interchanging 0 and 1.

Example 6. Let us consider the circular sturmian word $(w) = (0010010010)$. We have that $(w) = (0^2 10^2 10^3 1)$ and $\sigma(w) = \lfloor \frac{7}{3} \rfloor = 2$.

3. Sturmian morphisms on circular words

In this section we focus on sturmian morphisms (cf. [18, chap. 2]), in particular we are interested in their action on circular sturmian words. Remark that, if f is a morphism and u,v are conjugate words, then $f(u)$ and $f(v)$ are conjugate too. Since the set of images of elements of a conjugacy class is a conjugacy class, then the action of a morphism f on a circular word (w) is well defined and it is denoted by $f((w))$. One has that $f((w)) = (f(w))$.

Recall that a morphism f in A^* is *sturmian* if, for each infinite sturmian word x, $f(x)$ is also sturmian. Here we report a characterization of sturmian morphisms. Consider the following morphisms:

$$E\colon \begin{matrix} 0 \to 1 \\ 1 \to 0 \end{matrix} \qquad \varphi\colon \begin{matrix} 0 \to 01 \\ 1 \to 0 \end{matrix} \qquad \widetilde{\varphi}\colon \begin{matrix} 0 \to 10 \\ 1 \to 0 \end{matrix}$$

In [18] it is proved that the morphisms above defined are the generators of the monoid of sturmian morphisms.

The following proposition shows that the action of morphisms φ and $\widetilde{\varphi}$ coincides on circular words.

Proposition 7. *For any circular word (w), $\varphi((w)) = \widetilde{\varphi}((w))$.*

Proof. It suffices to prove that, for any word $v \in A^*$, $\varphi(v)$ is conjugate to $\widetilde{\varphi}(v)$. Actually, we prove that $\varphi(v) = 0x$ and $\widetilde{\varphi}(v) = x0$, for some word $x \in A^*$. The proof is by induction on the length of the word v. For $|v| = 1$ the statement holds true by the definition of φ and $\widetilde{\varphi}$. Let us suppose that the statement is true for all words of length k, and consider a word w of length $k+1$. If the last letter of w is 1, i.e. $w = v1$, with $|v| = k$, one has

$$\varphi(v1) = \varphi(v)\varphi(1) = 0x0,$$
$$\widetilde{\varphi}(v1) = \widetilde{\varphi}(v)\widetilde{\varphi}(1) = x00.$$

By setting $x0 = y$, one has $\varphi(w) = 0y$ and $\widetilde{\varphi}(w) = y0$. If the last letter of w is 0, i.e. $w = v0$, with $|v| = k$, one has

$$\varphi(v0) = \varphi(v)\varphi(0) = 0x01,$$
$$\widetilde{\varphi}(v0) = \widetilde{\varphi}(v)\widetilde{\varphi}(0) = x010.$$

By setting $x01 = y$, one has $\varphi(w) = 0y$ and $\widetilde{\varphi}(w) = y0$. This concludes the proof. □

The following important proposition corresponds to Lemma 4.1 of [5] and is here reported without proof.

Proposition 8. *A morphism f is sturmian if and only if, for each circular sturmian word (w), $f((w))$ is a circular sturmian word.*

Next proposition shows a sort of weak converse of the previous one.

Proposition 9. *Let (w) be a circular word and let f be a sturmian morphism. If $f((w))$ is sturmian then (w) is sturmian.*

Proof. If f is sturmian then it can be obtained as composition of morphisms E, φ and $\widetilde{\varphi}$. Since we consider the action of f on circular words, then, by Proposition 7, f is a composition of E and φ. So it suffices to prove that if $\varphi((w))$ is sturmian then (w) is sturmian (the implication "if $E((w))$ is sturmian then (w) is sturmian" is trivial). Let us suppose, by contradiction, that (w) is not sturmian, i.e. by Proposition 2, (w) is not balanced. Then, for some word x, $0x0$ and $1x1$ are both factors of (w). One has that $\varphi(0x0) = 01\varphi(x)01$ and $\varphi(1x1) = 0\varphi(x)0$. Every occurrence of $\varphi(1x1)$ in $\varphi((w))$ is followed by the letter 0. As a consequence $1\varphi(x)01$ and $0\varphi(x)00$ are both factors of $\varphi((w))$, which is not balanced, against the hypothesis. □

In Section 4 it is shown how we can associate to each circular sturmian word (x) a tree that highlights the structure of the word itself.

We now define a morphism that plays an important role in the construction of such a tree. If p is a positive integer, we define the morphism ψ_p as follows:

$$\psi_p: \begin{array}{l} 0 \to 10^p \\ 1 \to 10^{p+1}. \end{array}$$

Remark that, $\psi_p = (\widetilde{\varphi}E)^{p-1}\widetilde{\varphi}\varphi E$ and then ψ_p is a sturmian morphism (we write the composition of morphisms as concatenation, i.e. we write fg instead of $f \circ g$).

The definition of the reduction tree, in Section 4, is based on the following proposition.

Proposition 10. *For any circular sturmian word (w) there exists a sturmian morphism f such that $(w) = f((0))$.*

Proof. It suffices to prove that, if $|w| \geq 2$, there exist a sturmian morphism g and a circular sturmian word (v), with $|v| < |w|$, such that $(w) = g((v))$. The thesis then follows by induction on the length of w.

If (w) is sturmian and $|w| \geq 2$, by Corollary 5 there exists an integer $p \geq 0$ such that (w) is concatenation of the factors 10^p and 10^{p+1}. It follows that there exists a circular word (v), with $|v| < |w|$, such that $(w) = \psi_p((v))$. By Proposition 9, it follows that (v) is a circular sturmian word and this concludes the proof. □

4. Reduction tree of circular sturmian words

Let (x) be a circular sturmian word with signature $\sigma(x)$. If $|x|_0 \geq |x|_1$ (resp. $|x|_0 < |x|_1$), by Corollary 5, (x) (resp. $(E(x))$) can be circularly factorized in $X = \{0, 01\}$ and $Y = \{10^{\sigma(x)}, 10^{\sigma(x)+1}\}$. Since X and Y are circular codes (cf. [4]) the factorizations are unique. In the factorization of (x) (or $(E(x))$) in elements of X, we can encode each block 01 by 0 and each block 0 by 1. In such a way we obtain a circular binary word that we denote $(L(x))$. In the factorization of (x) (or $(E(x))$) in elements of Y, we can encode each block $10^{\sigma(x)}$ by 0 and each block $10^{\sigma(x)+1}$ by 1. In such a way we obtain a circular binary word that we call $(R(x))$.

Example 11. Let $(x) = (01010010100101010)$ and $(y) = (1010110)$. We have that $(L(x)) = (0010010001)$, $(R(x)) = (1010100)$, $(L(y)) = (0010)$ and $(R(y)) = (001)$.

Remark 12. For each circular sturmian word (x) such that $|x|_0 \geq |x|_1$ (resp. $|x|_0 < |x|_1$) we have that

(1) $x = \varphi(L(x)) = \psi_{\sigma(x)}(R(x))$ (resp. $E(x) = \varphi(L(x)) = \psi_{\sigma(x)}(R(x))$). Since φ and $\psi_{\sigma(x)}$ are sturmian morphisms, by Proposition 9, $(L(x))$ and $(R(x))$ are circular sturmian words;
(2) $|x|_0 = |L(x)|$ (resp. $|x|_1 = |L(x)|$) and $|x|_1 = |R(x)|$ (resp. $|x|_0 = |R(x)|$), i.e. $|x| = |L(x)| + |R(x)|$.

To a circular sturmian word (x) we can associate a tree $\mathcal{T}(x)$, called *reduction tree* of (x), defined as follows:

- if $(x) = (0)$ or $(x) = (1)$, $\mathcal{T}(x)$ is a single node with label (0);
- if $|x| > 1$, $\mathcal{T}(x)$ is a tree with root labeled by (x) having respectively as left and right subtrees $\mathcal{T}(L(x))$ and $\mathcal{T}(R(x))$.

Example 13. Let us consider the circular sturmian word (0010010010). The tree $\mathcal{T}(0010010010)$ is shown in Fig. 1.

Let us denote by $\tau(x)$ the unlabeled tree obtained from $\mathcal{T}(x)$ by removing the labels. Practically speaking, $\tau(x)$ represents the shape of the reduction tree of (x). The following proposition states that each circular sturmian word is uniquely determined by a binary tree.

Proposition 14. *Let (x) and (y) be sturmian words, then*

$$\tau(x) = \tau(y) \quad \text{if and only if} \quad (x) = (y) \quad \text{or} \quad (E(x)) = (y).$$

Proof. If $\tau(x)$ and $\tau(y)$ are single nodes the proof is trivial. Let us consider $\tau(x)$ and $\tau(y)$. Let $\tau(x)_{sx}$, $\tau(x)_{dx}$, $\tau(y)_{sx}$ and $\tau(y)_{dx}$ be the right and left subtrees of $\tau(x)$ and $\tau(y)$, respectively. By definition, if $L(x)$, $R(x)$, $L(y)$ and $R(y)$ are the factorizations of (x) and (y), respectively, we have $\tau(x)_{sx} = \tau(L(x))$, $\tau(x)_{dx} = \tau(R(x))$, $\tau(y)_{sx} = \tau(L(y))$ and $\tau(y)_{dx} = \tau(R(y))$. If $\tau(x) = \tau(y)$ then $\tau(x)_{sx} = \tau(y)_{sx}$ and $\tau(x)_{dx} = \tau(y)_{dx}$. It follows that $\tau(L(x)) = \tau(L(y))$, $\tau(R(x)) = \tau(R(y))$.Then, by induction,

$$(L(x)) = (L(y)) \quad \text{or} \quad (E(L(x))) = (L(y))$$

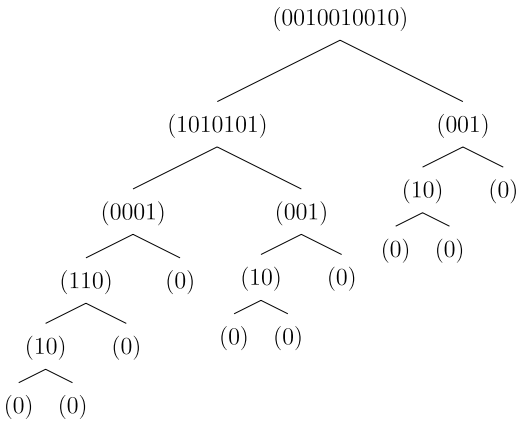

Fig. 1. The reduction tree of the word (0010010010).

and

$$(R(x)) = (R(y)) \quad \text{or} \quad (E(R(x))) = (R(y)).$$

It follows that $|x| = |L(x)| + |R(x)| = |L(y)| + |R(y)| = |y|$. Furthermore, let us suppose that $|x|_0 \geq |x|_1$. Then, either ($|x|_0 = |L(x)| = |L(y)| = |y|_0$ and $|x|_1 = |R(x)| = |R(y)| = |y|_1$) or ($|x|_0 = |L(x)| = |L(y)| = |y|_1$ and $|x|_1 = |R(x)| = |R(y)| = |y|_0$). That is, either $(x) = (y)$ or $(E(x)) = (y)$. □

The following proposition establishes a close relation among the special factors of the circular sturmian words (x), $(L(x))$ and $(R(x))$.

Proposition 15. *Let (x) be a circular sturmian word with $|x|_0 \geq |x|_1$. If v is a special factor of $(L(x))$ then $\varphi(v)0$ is a special factor of (x). Conversely, if w is a special factor of (x) starting with 0 then $w = \varphi(v)0$, where v is a special factor of $(L(x))$.*

Proof. Let v be a special factor of $(L(x))$, then both $v0$ and $v1$ are factors of $(L(x))$. Since $\varphi(v1) = \varphi(v)0$ then $\varphi(v)0$ is a factor of (x). Moreover $\varphi(v0) = \varphi(v)01$ is also a factor of (x). Note that since 11 is forbidden, $v10$ has to be a factor of $(L(x))$. Hence $\varphi(v10) = \varphi(v)001$ is a factor of (x). Consequently, it follows that $\varphi(v)0$ is a special factor of (x). Conversely, let w be a special factor of (x) starting from 0. It is easy to verify that w must be ended by 0, hence $w = z0$ where $z \in (0 + 01)^*$. Therefore, there exists v such that $z = \varphi(v)$ and v is a special factor of $(L(x))$ because $w1 = \varphi(v0)$ and $w0$ contains $\varphi(v1)$ as a prefix. □

Proposition 16. *Let (x) be a circular sturmian word with $|x|_0 \geq |x|_1$. If v is a special factor of $(R(x))$ then $\psi_{\sigma(x)}(v)10^{\sigma(x)}$ is a special factor of (x). Conversely, if w is a special factor of (x) starting with 1 then $w = \psi_{\sigma(x)}(v)10^{\sigma(x)}$, where v is a special factor of $(R(x))$.*

Proof. Let v be a special factor of $(R(x))$, then both $v0$ and $v1$ are factors of $(R(x))$. Since $\psi_{\sigma(x)}(v0) = \psi_{\sigma(x)}(v)10^{\sigma(x)}$ then $\psi_{\sigma(x)}(v)10^{\sigma(x)}$ is a factor of (x). Moreover $\psi_{\sigma(x)}(v1) = \psi_{\sigma(x)}(v)10^{\sigma(x)+1}$ is also a factor of (x). Note that since 11 is forbidden, $v10$ has to be a factor of $(R(x))$. Hence $\psi_{\sigma(x)}(v01) = \psi_{\sigma(x)}(v)10^{\sigma(x)}10^{\sigma(x)+1}$ is a factor of (x). Consequently, it follows that $\psi_{\sigma(x)}(v)10^{\sigma(x)}$ is a special factors of (x). Conversely, let w be a special factor of (x) starting from 1. It is easy to verify that w must be ended by $0^{\sigma(x)}$, hence $w = z10^{\sigma(x)}$ where $z \in (10^{\sigma(x)} + 10^{\sigma(x)+1})^*$. Therefore there exists v such that $z = \psi_{\sigma(x)}(v)$ and v is a special factor of $(R(x))$ because $w0 = \psi_{\sigma(x)}(v1)$ and $w1$ must have $\psi_{\sigma(x)}(v0)$ as prefix. □

5. Minimization of finite state automata

In this section we give some basics about methods for minimization of finite automata. We focus our attention to Hopcroft's algorithm.

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a *deterministic finite automaton* (*DFA*) over the finite alphabet Σ, where Q is a finite state set, δ is a *transition function* $Q \times \Sigma \to Q$, $q_0 \in Q$ is the *initial state* and $F \subseteq Q$ the set of *final states*. If C is a subset of Q and $a \in \Sigma$, with $\delta_a^{-1}(C)$ we denote the set $\{q \in Q | \delta(q, a) \in C\}$.

Two finite automata are *equivalent* if they recognize the same language.

A *DFA* is *minimal* if it has the minimum number of states among all its equivalent deterministic automata. For each regular language there exists a unique (up to isomorphism) minimal automaton recognizing it. It is computed by using the Nerode equivalence, as described below.

For any state $p \in Q$, it is considered the language

$$\mathcal{L}_p(\mathcal{A}) = \{v \in \Sigma^* | \delta(p, v) \in F\}.$$

The *Nerode equivalence* on Q, denoted by $\sim$, is defined as follows: for $p, q \in Q$, $p \sim q$ if $\mathcal{L}_p(\mathcal{A}) = \mathcal{L}_q(\mathcal{A})$.

We say that an equivalence relation $\sim$ defined on the set Q of the states of $\mathcal{A}$ is a *congruence* of $\mathcal{A}$ if it is compatible with the transitions of $\mathcal{A}$, i.e. for any $a \in \Sigma$, $p \sim q$ implies $\delta(p, a) \sim \delta(q, a)$.
It is also known (cf. [11]) that the Nerode equivalence is the coarsest congruence of $\mathcal{A}$ that saturates F, i.e. such that F is union of classes of the congruence.

The minimal automaton equivalent to a given *DFA* can be computed by merging states which are equivalent w.r.t. the Nerode equivalence. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a *DFA* that recognizes the regular language $\mathcal{L}$, and $\Pi = \{Q_1, Q_2, \ldots, Q_m\}$ the partition corresponding to Nerode equivalence.

For $q \in Q_i$, the class Q_i is denoted by $[q]$. Then the minimal automaton that recognizes $\mathcal{L}$ is $\mathcal{MA} = (\overline{Q}, \Sigma, \overline{\delta}, \overline{q_0}, \overline{F})$, where:

- $\overline{Q} = \{Q_1, Q_2, \ldots, Q_m\}$
- $\overline{q_0} = [q_0]$
- $\overline{\delta}([q], a) = [\delta(q, a)], \forall\ q \in Q$ and $\forall\ a \in \Sigma$
- $\overline{F} = \{[q] |\ q \in F\}$

The Nerode equivalence is commonly computed by the Moore construction (cf. [20]) that, for any integer $k \geq 0$, defines

$$\mathcal{L}_p^k = \{v \in \mathcal{L}_p || v| \leq k\}.$$

Then the equivalence $\sim_k$ on Q is defined as follows:

$$p \sim_k q \Leftrightarrow \mathcal{L}_p^k = \mathcal{L}_q^k.$$

Such a relation means that in order to distinguish the two states p and q, a word of length at least $k + 1$ is needed.

Theorem 17 (*Moore*). *The Nerode equivalence is equal to* $\sim_{|Q|-2}$.

Given an automaton $\mathcal{A}$, denote by $\mu(\mathcal{A})$ the integer defined as follows:

$$\mu(\mathcal{A}) = \min\{k |\ \sim_k = \sim_{k+1}\}.$$

By the theorem of Moore, $\mu(\mathcal{A}) \leq |Q| - 2$.
Note that $\mu(\mathcal{A})$ represents the minimal length of the words needed to determine the partition corresponding to the Nerode equivalence.

There exist several methods which can be used to compute the Nerode equivalence and the value $\mu(\mathcal{A})$ in order to minimize a finite automaton $\mathcal{A}$. Some of them operate by successive refinements of a partition of the states of a given *DFA* (cf. [20,15]). Moore's algorithm computes in time $O(|\Sigma||Q|^2)$ the minimal automaton, Hopcroft's algorithm is the most efficient in the worst case and its running time is $O(|\Sigma||Q| \log |Q|)$. Brzozowski's method (cf. [7]) operates by reversal and determinization repeated twice and it can be also applied to a non-deterministic finite automata. The time complexity is exponential in the worst case, but it has good performance in practice (cf. [9]). Other methods work only for a restricted class of automata, for instance for acyclic automata (cf. [22,10]) and local automata (cf. [1]).

In 1971 Hopcroft gave an algorithm for minimizing a finite state automaton with n states, over an alphabet Σ, in $O(|\Sigma| n \log n)$ time (c.f. [15]). This algorithm has been widely studied and described by many authors (see for example [14, 17,19,24]) cause of the difficult to give its theoretical justification, to prove correctness and to compute running time.

Here we give a brief description of the algorithm's running.

```
MINIMIZATION (A = (Q, Σ, δ, q0, F))
 1. Π ← {F, Q \ F}
 2. for all a ∈ Σ do
 3.     W ← {(min(F, Q \ F), a)}
 4. while W ≠ ∅ do
 5.     choose and delete any (C, a) from W
 6.     for all B ∈ Π do
 7.         if B is split from (C, a) then
 8.             B' ← δ_a^{-1}(C) ∩ B
 9.             B'' ← B \ δ_a^{-1}(C)
10.             Π ← Π \ {B} ∪ {B', B''}
11.             for all b ∈ Σ do
12.                 if (B, b) ∈ W then
13.                     W ← W \ {(B, b)} ∪ {(B', b), (B'', b)}
14.                 else
15.                     W ← W ∪ {(min(B', B''), b)}
```

Given an automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, it computes the coarsest congruence that saturates F. Let us observe that the partition $\{F, Q \setminus F\}$, trivially, saturates F.

Given a partition $\Pi = \{Q_1, Q_2, \ldots, Q_m\}$ of Q, we say that the pair (Q_i, a), with $a \in \Sigma$, *splits* the class Q_j if $\delta_a^{-1}(Q_i) \cap Q_j \neq \emptyset$ and $Q_j \not\subseteq \delta_a^{-1}(Q_i)$. In this case the class Q_j is split into $Q_j' = \delta_a^{-1}(Q_i) \cap Q_j$ and $Q_j'' = Q_j \setminus \delta_a^{-1}(Q_i)$. Let us note that if Π saturates F then $\Pi \setminus \{Q_j\} \cup \{Q_j', Q_j''\}$ saturates F and it is coarser than Π. Furthermore, we have that a partition Π is a congruence if and only if for any $1 \leq i, j \leq m$ and any $a \in \Sigma$, the pair (Q_i, a) does not splits Q_j.

The main idea of the algorithm is the following. It starts from the partition $\{F, Q \setminus F\}$ and refines it by means of splitting operations until it obtains a congruence, i.e. until no more split is possible. To do that it maintains the current partition Π and a set $\mathcal{W} \subseteq \Pi \times \Sigma$, called *waiting set*, that contains the pairs for which it has to check whether some classes of the current partition is split.

The main loop of the algorithm takes and deletes one pair (C, a) from $\mathcal{W}$ and, for each class B of Π, checks if it is split by (C, a). If it is the case, the class B in Π is replaced by the two sets B' and B'' obtained from the split. For each $b \in \Sigma$, if $(B, b) \in \mathcal{W}$, it is replaced by (B', b) and (B'', b), otherwise the pair $(\min(B', B''), b)$ is added to $\mathcal{W}$ (with the notation $\min(B', B'')$ we mean the set with minimum cardinality between B' and B''). Let us observe that a class is split by (B', b) if and only if it is split by (B'', b), hence, the pair $(\min(B', B''), b)$ is chosen for convenience.

We point out that the algorithm is not deterministic because the pair (C, a) to be processed at each step is freely chosen. Furthermore, when a set B is split into B' and B'' with the same size and it is not present in $\mathcal{W}$ the algorithm can, indifferently, add to $\mathcal{W}$ either (B', b) or (B'', b). This implies that for each automaton there can be many different executions that produce the same partition and, as consequence, different running time.

The refinement process that, during each execution of the algorithm, leads from the initial partition $\Pi = \{F, Q \setminus F\}$ to the coarsest congruence of the automaton $\mathcal{A}$ can be represented by a tree $\mathcal{T}(\mathcal{A})$, also called *derivation tree* of $\mathcal{A}$. Each node of $\mathcal{T}(\mathcal{A})$ is labeled with a subset of Q that we find as a class of the partition Π during the execution of the algorithm. In particular, the root of the tree is labeled by Q and its immediate descendants are F and $Q \setminus F$. At each step, the leaves are labeled with the classes of the current partition obtained by the algorithm. Each intermediate node B has two descendants labeled by B' and B'' if the class B is split into the sets B' and B''. Note that the shape of the tree $\mathcal{T}(\mathcal{A})$ is strongly affected from the non-deterministic choices of Hopcroft's algorithm. Both the order in which the pairs are extracted from the waiting set and the insertion of B' or B'' (when the cardinality of such sets is the same) can lead to the construction of different derivation trees, although with the same leaves.

The notion of derivation tree equipped with a suitable coloring for the nodes is used in [17] as a tool both in the correctness proof and the time analysis of the algorithm. In the paper the author assigns a color to a node B depending on whether the pair (B, x) is inserted, removed or replaced in the waiting set, for any $x \in \Sigma$. The running time of the algorithm is expressed and computed in terms of the coloring of the nodes of the derivation tree and it is proved that the worst case is obtained when no pair is replaced in the waiting set. Moreover, we can observe that the splitting of classes of the partition, with respect to the pair (C, a), takes a time proportional to the cardinality of the set C. Hence, the running time of the algorithm is proportional to the sum of the cardinality of all sets extracted from the waiting set.

Hopcroft proved that the running time is bounded by $O(|\Sigma||Q| \log |Q|)$. In [3] the authors prove that this bound is tight, in the sense that they provide a class of unary automata for which there exist executions of Hopcroft's algorithm that run in time $O(|\Sigma||Q| \log |Q|)$. Such a bound is reached by using a non-splitting strategy in choosing the class to add at each step in $\mathcal{W}$. Other strategy could produce executions that run in linear time for the same automata.

6. Minimization of cyclic automata

In this section we deepen the connection between the refinements strategy of Hopcroft's algorithm on unary cyclic automata and the combinatorial properties of circular words. Finally, we investigate the relationship between the derivation tree describing the execution of the algorithm on a unary cyclic automata associated to a circular sturmian word and the reduction tree representing the structure of such a word.

Following the idea introduced in [3], we associate to a circular word (w) an automaton $\mathcal{A}_w$ as follows.

Definition 18. Let (w) be a circular word, where $w = a_1 a_2 \ldots a_n$ be a word of length n over the binary alphabet $A = \{0, 1\}$. The *cyclic automaton associated to* (w), denoted by $\mathcal{A}_w$, is the automaton (Q, Σ, δ, F) such that:

- $Q = \{1, 2, \ldots, n\}$
- $\Sigma = \{a\}$
- $\delta(i, a) = (i + 1), \forall\ i \in Q \setminus \{n\}$ and $\delta(n, a) = 1$
- $F = \{i \in Q |\ a_i = 1\}$

See Fig. 2 for example. We do not specify the initial state because for our aim it does not matter.

We want to observe how combinatorial properties of the word w are closed to the properties of the states of the automaton $\mathcal{A}_w$. Firstly, we recall some results proved in [8].

Remark 19. For any $i \in Q$ and any k, $0 \leq k \leq n$, the path starting from i and having label a^k corresponds to the factor $a_i a_{i+1} \ldots a_{i+k}$ of (w), i.e. the factor of (w) of length $k + 1$ starting from the position i. From the definition of the equivalence $\sim_k$ one has that $i \sim_k j$ if the factors of w of length $k + 1$ starting from i and j, respectively, are equal.

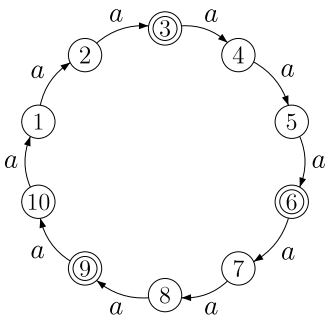

Fig. 2. Cyclic automaton $\mathcal{A}_w$ for $w = 0010010010$.

Theorem 20. *$\mathcal{A}_w$ is minimal iff (w) is primitive.*

Proof. By the Theorem of Moore, $\mathcal{A}_w$ is minimal if and only if each class of $\sim_{|Q|-2}$ is a singleton. By the previous remark, this corresponds to the fact that all the factors of (w) of length $|Q| - 1$ have a unique occurrence in (w). This means, by Proposition 1, that w is primitive. □

Now we consider the cyclic automaton $\mathcal{A}_w$ associated to a circular sturmian word. The following theorem gives a characterization of circular sturmian words in terms of the automaton $\mathcal{A}_w$.

Theorem 21. *(w) is a circular sturmian word iff $\mu(\mathcal{A}_w) = |w| - 2$.*

Proof. By Proposition 2, (w) is standard iff (w) has $|w| - 1$ factors of length $|w| - 2$ and w is primitive. It follows that for $k \leq |w| - 2$ there exists a factor of (w) of length k that occurs at least twice in (w). This means, by the previous remark, that for each $k \in \{1, \ldots, |w| - 2\}$ there exist two states $i, j \in \{1, \ldots, |w|\}$, $i \neq j$, such that $i \sim_{k-1} j$. Since all factors of (w) of length $|w| - 1$ occur once in (w), the classes of $\sim_{|w|-2}$ are singleton, i.e. $\sim_{|w|-2}$ is equal to Nerode equivalence. □

Note that, as proved in [8], given a circular binary word (w),

$$\mu(\mathcal{A}_w) = \max\{|u| | u \text{ is a special factor of } (w)\}.$$

In order to stress the difference between the automata considered in [3] and the ones studied in this paper, note that if (w) is a circular de Bruijn word of length n, $\mu(\mathcal{A}_w) = \log n - 1$. On the contrary, if (w) is a circular sturmian word of length n, $\mu(\mathcal{A}_w) = n - 2$.

In the following subsection we analyze the executions of Hopcroft's algorithm on unary circular automata by using combinatorial properties of the associated binary words. Moreover, we consider a cyclic automaton associated to a circular sturmian word. We show the relation between the derivation tree of Hopcroft's algorithm and the reduction tree of the associated word.

6.1. Hopcroft's algorithm, derivation trees and circular sturmian words

For a word $u \in A^*$, we define a subset Q_u of states of $\mathcal{A}_w$ as the set of positions of circular occurrences of the factor u in (w). Trivially, we have that $Q_\epsilon = Q$, $Q_1 = F$ and $Q_0 = Q \setminus F$. Let u be a factor of (w) such that Q_u is a class of the partition of Q. We say that Q_u is a *splitting* subset of states if there exists $v \in A^*$ such that Q_v is a class of the partition and it splits Q_u.

We report the following proposition, proved in [8], that establishes a close relation between the execution of Hopcroft's algorithm on a cyclic automata and the notion of special factor of a circular word.

Proposition 22. *Let Q_u and Q_v be classes of the partition. If Q_v splits Q_u and $|u| = |v|$ then u is a special factor of (w) and the resulting sets are Q_{u0} and Q_{u1}.*

The classes that appear during each execution of Hopcroft's algorithm on cyclic automaton $\mathcal{A}_w$ are all of the form Q_u for some factor u of (w). Then the nodes of the associated derivation tree are labeled with Q_u.

The following proposition, proved in [8], describes the current partition of the set of states at each step of Hopcroft's algorithm on a cyclic automaton associated to a circular sturmian word.

Let us denote by Π_k and $\mathcal{W}_k$ the partition and the waiting set at the kth step of the algorithm.

Proposition 23. *The executions of Hopcroft's algorithm on a cyclic standard automaton $\mathcal{A}_w$, where (w) is a circular sturmian word, are uniquely determined. At each step $1 \leq k \leq n - 2$, $\Pi_k = \{Q_v | v \text{ is a factor of length } k\}$ and $|\mathcal{W}_k| = 1$.*

Hence, the execution of Hopcroft's algorithm on cyclic automata associated to circular sturmian words is deterministic since $\mathcal{W}$ contains only one element at each step. Furthermore, if the class Q_u is split into Q_{u0} and Q_{u1} of the same size, the execution does not change whatever of the two classes is added to the waiting set $\mathcal{W}$.

The partial derivation tree obtained after each step of the main loop of the algorithm on $\mathcal{A}_w$, where (w) is a circular sturmian word, is uniquely determined and its leaves are labeled by the elements of the set Π_k.

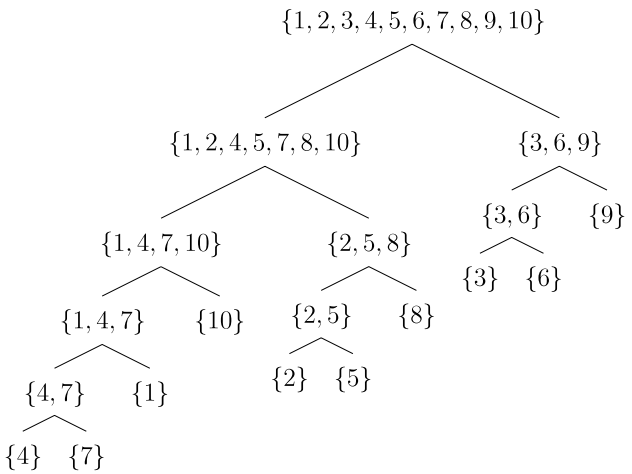

Fig. 3. The derivation tree of the automaton $\mathcal{A}_x$ with $(x) = (0010010010)$.

Example 24. Let us consider the cyclic automaton $\mathcal{A}_x$ with $(x) = (0010010010)$. The tree $\mathcal{T}(\mathcal{A}_x)$ is shown in Fig. 3

Let us denote by $\tau(\mathcal{A}_x)$ the unlabeled tree obtained from $\mathcal{T}(\mathcal{A}_x)$ by removing the labels. The following proposition shows that the unlabeled derivation tree obtained by the execution of Hopcroft's algorithm on a cyclic automaton associated to a circular sturmian word is equal to the unlabeled reduction tree representing the structure of the word itself.

Theorem 25. *If (x) is a circular sturmian word then*

$$\tau(\mathcal{A}_x) = \tau(x).$$

Proof. Let us prove the result by induction on the length of the word x. If $|x| = 1$ then both $\tau(\mathcal{A}_x)$ and $\tau(x)$ are singular nodes, so the proof is trivial. Let us suppose now that $|x| > 1$ and $|x|_0 \geq |x|_1$. Note that if $|x|_0 < |x|_1$ we can consider $E(x)$ because the derivation tree is the same. We can associate to (x) the reduction tree $\mathcal{T}(x)$ having (x) as label of the root. Note that from Proposition 22 we can deduce that the internal nodes of $\mathcal{T}(\mathcal{A}_x)$ are labeled with all the sets $\{Q_u^{(x)} \mid u \text{ special factor of } x\}$ and the leaves are labeled with all the sets $\{Q_v^{(x)} \mid v \text{ factor of } x \text{ of length } n-1\}$. Moreover, each internal node $Q_u^{(x)}$ is split into two states $Q_{u0}^{(x)}$ and $Q_{u1}^{(x)} = Q_{u10^{\sigma(x)}}^{(x)}$, where $u0$ and $u10^{\sigma(x)}$ are special factors of (x). Moreover, the label of the root of $\mathcal{T}(\mathcal{A}_x)$ is $Q_\varepsilon^{(x)} = \{1, 2, \ldots, |x|\}$ and it is split into $Q_0^{(x)}$ and $Q_1^{(x)}$ containing the occurrences of 0's and 1's, respectively. $Q_0^{(x)}$ and $Q_1^{(x)}$ are roots of the left and right subtrees of $\mathcal{T}(\mathcal{A}_x)$, denoted by $\mathcal{T}(\mathcal{A}_x)_{sx}$ and $\mathcal{T}(\mathcal{A}_x)_{dx}$, respectively. On the other hand, the reduction tree $\mathcal{T}(x)$ rooted in (x) has $\mathcal{T}(L(x))$ and $\mathcal{T}(R(x))$ as left and right subtree, respectively. We prove the isomorphism between $\mathcal{T}(\mathcal{A}_x)_{sx}$ and $\mathcal{T}(\mathcal{A}_{L(x)})$ and between $\mathcal{T}(\mathcal{A}_x)_{dx}$ and $\mathcal{T}(\mathcal{A}_{R(x)})$. Consider the first one. In particular we show that there exists a bijection that maps nodes of $\mathcal{T}(\mathcal{A}_{L(x)})$ to nodes of $\mathcal{T}(\mathcal{A}_x)_{sx}$. Moreover, for each child of a node in a tree there is a child of the correspondent node in the other tree. Recall that (x) is uniquely circular factorized in $\{0, 01\}$. Let $x = w_1 w_2 \ldots w_k$ be its decomposition, where $w_j = x_{i_j} x_{i_j+1} \ldots x_{i_{j+1}-1}$. It is easy to see that $k = |x|_0$ and the indexes i_j are exactly the positions of 0's in x. Hence we can define the map λ from $\{1, 2, \ldots, |x|_0\}$ to $\{1, 2, \ldots, |x|\}$ such that $\lambda(j) = i_j$. It is easy to see that λ is injective and $\lambda(Q_\varepsilon^{L(x)}) = Q_0^{(x)}$. By Proposition 15, each internal node of $\mathcal{T}(\mathcal{A}_x)_{sx}$ has the form $Q_{\varphi(v)0}^{(L(x))}$, where v is a special factor of $L(x)$. Since φ is a morphism we have that $\lambda(Q_v^{(L(x))}) = Q_{\varphi(v)0}^{(x)}$ and such a correspondence is bijective. Moreover the leaves of $\mathcal{T}(\mathcal{A}_x)_{sx}$ have the form $Q_{\varphi(v)00}^{(x)}$ and $Q_{\varphi(v)01}^{(x)}$, resulting from the split of $Q_{\varphi(v)0}^{(x)}$ where $\varphi(v)0$ has length $|x|-2$. Each leaf of $\mathcal{T}(\mathcal{A}_{L(x)})$ has the form $Q_{v0}^{(L(x))}$ and $Q_{v1}^{(L(x))}$, where v is a special factor of $(L(x))$ of length $|x|_0 - 2$. Then $v0$ and $v1$ are factors that occur only once in $(L(x))$. So, $\varphi(v)01$ and $\varphi(v)00$ occur only once in (x). Hence $\mathcal{T}(\mathcal{A}_x)_{sx} \simeq \mathcal{T}(\mathcal{A}_{L(x)})$. By induction $\tau(\mathcal{A}_x)_{sx} = \tau(L(x))$. Analogously we can prove that $\mathcal{T}(\mathcal{A}_x)_{dx} \simeq \mathcal{T}(\mathcal{A}_{R(x)})$ by using the bijection ρ from $\{1, 2, \ldots, |x|_1\}$ to $\{1, 2, \ldots, |x|\}$. In fact, (x) is uniquely circular factorized in $\{10^{\sigma(x)}, 10^{\sigma(x)+1}\}$. Let $x = w_1 w_2 \ldots w_p$ be its decomposition, where $w_j = x_{i_j} x_{i_j+1} \ldots x_{i_{j+1}-1}$. It is easy to see that $p = |x|_1$ and the indexes i_j are exactly the positions of 1's in x. Hence we can define the map ρ from $\{1, 2, \ldots, |x|_1\}$ to $\{1, 2, \ldots, |x|\}$ such that $\rho(j) = i_j$. As proved for the left tree we obtain the thesis. □

6.2. On the complexity of Hopcroft's algorithm

As mentioned in Section 5, the running time of the algorithm is proportional to the sum of the size of all sets processed in the waiting set. Theorem 25 and its proof provide some structural information about such sets when we consider the automaton $\mathcal{A}_x$ with (x) circular sturmian word. Moreover, we can deduce a recursive method to compute the running time, denoted by $\mathbf{c}(x)$, of the execution of Hopcroft's algorithm on $\mathcal{A}_x$.

Proposition 26. *Let (x) a circular sturmian word. If $|x| = 1$ then $\mathbf{c}(x) = 0$, else*

$$\mathbf{c}(x) = \mathbf{c}(L(x)) + \mathbf{c}(R(x)) + \min\{|L(x)|, |R(x)|\}.$$

Proof. We know that $c(x)$ is given by the sum of the size of those classes that are extracted from the waiting set. In the case of circular sturmian word, such classes are exactly the minimal classes resulting from each split. Note that the size of each node is the length of the word in the corresponding node of the reduction tree of (x). By the recursive structure of the derivation tree the thesis follows. □

It is interesting to analyze two extremal cases. The first one is $(0^p 1)$. In this case the running time of the algorithm on the cyclic automaton associated is, trivially, $\Theta(|Q|)$. In fact, in this case, each internal node P, of the derivation tree, is split into two classes having size 1 and $|P| - 1$, respectively. From such an example, we get that there exists an infinite sequence of words for which the execution of Hopcroft's algorithm is linear. This fact does not contradict the result in [2] because such words are not generated by an eventually periodic directive sequence. The second extremal example is the case of circular sturmian words represented by the nth Fibonacci finite words.

In [3] the authors pose the open problem whether there are automata on which all the executions of Hopcroft's algorithm do not run in linear time. At the same time in [21] the author conjectures that there is a strategy for implementing the waiting set such that the minimization of all unary languages will be realized in linear time by Hopcroft's algorithm. With this example we give a solution to both the above questions by giving an infinite family of automata for which the running time is $\Theta(n \log n)$, whatever implementation strategy is used.

The infinite Fibonacci word f, over the alphabet $\{0, 1\}$ is the limit of the infinite sequence $\{f_n\}_{n\geq 0}$ of binary words inductively defined as $f_0 = 1, f_1 = 0, f_{n+1} = f_n f_{n-1}, n \geq 2$. Words f_n are called finite Fibonacci words. We denote by f_n also each circular Fibonacci word. The numbers $F_n = |f_n|$, for $n \geq 1$, are the Fibonacci numbers defined by the recurrence equation $F_{n+1} = F_n + F_{n-1}$, with $F_0 = F_1 = 1$. For each n we have that $(L(f_n)) = (f_{n-1})$ and $(R(f_n)) = (f_{n-2})$, respectively. From the previous proposition it follows that

$$c(f_n) = c(f_{n-1}) + c(f_{n-2}) + F_{n-2}.$$

Then the sequence $\{\mathbf{c}(f_n)\}_{n\geq 0}$ of running time of Hopcroft's algorithm on automaton associated to finite Fibonacci words is the Fibonacci convolution sequence (sequence A001629 in [23]). The nth term is $\mathbf{c}(f_n) = \frac{1}{5}((n-1)F_n + 2nF_{n-1})$ (cf. [13]). We know that $F_n = [\frac{\phi^n}{\sqrt{5}}]$, where $[x]$ is the nearest integer function and ϕ is the golden ratio $\frac{1+\sqrt{5}}{2}$. By simple computations, one can prove that definitively we have

$$\frac{k}{\phi} F_n \log F_n \leq c(F_n) \leq k F_n \log F_n,$$

where $k = \frac{3}{5 \log \phi}$.

References

[1] M.-P. Béal, M. Crochemore, Minimizing local automata, in: G. Caire, M. Fossorier (Eds.), IEEE International Symposium on Information Theory, ISIT'07, IEEE Catalog, 2007, pp. 1376–1380. number 07CH37924C.
[2] J. Berstel, L. Boasson, O. Carton, Hopcroft's automaton minimization algorithm and Sturmian words, in: Proceedings of Fifth Colloquium on Mathematics and Computer Science, DMTCS proc., vol. AI, 2008, pp. 355–366.
[3] J. Berstel, O. Carton, On the complexity of Hopcroft's state minimization algorithm, in: M. Domaratzki, A. Okhotin, K. Salomaa, S. Yu (Eds.), CIAA, in: Lecture Notes in Computer Science, vol. 3317, Springer, 2004, pp. 35–44.
[4] J. Berstel, D. Perrin, Theory of Codes, Academic Press, Inc., Orlando, FL, USA, 1985.
[5] V. Berthé, A. de Luca, C. Reutenauer, On an involution of christoffel words and sturmian morphisms, Eur. J. Combin. 29 (2) (2008) 535–553.
[6] J.P. Borel, C. Reutenauer, On Christoffel classes, RAIRO-Theoret. Inform. Appl. 450 (2006) 15–28.
[7] J.A. Brzozowski, Canonical regular expressions and minimal state graphs for definite events, Mathematical Theory of Automata, MRI Symposia Series.
[8] G. Castiglione, A. Restivo, M. Sciortino, Hopcroft's algorithm and cyclic automata, in: C. Martín-Vide, F. Otto, H. Fernau (Eds.), LATA, in: Lecture Notes in Comput. Sci., vol. 5196, Springer, 2008, pp. 172–183.
[9] J.-M. Champarnaud, A. Khorsi, T. Paranthon, Split and join for minimizing: Brzozowskis algorithm, in: Proceedings of PSC02 (Prague Stringology Conference), 2002 pp. 96–104.
[10] J. Daciuk, R. Watson, B. Watson, Incremental construction of acyclic finite-state automata and transducers, in: Finite State Methods in Natural Language Processing, Bilkent University, Ankara, Turkey, 1998.
[11] S. Eilenberg, Automata, Languages, and Machines, vol. A, 1974.
[12] F. Franek, A. Karaman, W. Smyth, Repetitions in Sturmian strings, Theoret. Comput. Sci. 249 (2000) 289–303.
[13] R.L. Graham, D.E. Knuth, O. Patashnik, Concrete Mathematics, Addison-Wesley, Reading, MA, 1989.
[14] D. Gries, Describing an algorithm by Hopcroft, Acta Inform. 2 (1973) 97–109.
[15] J.E. Hopcroft, An $n \log n$ algorithm for minimizing the states in a finite automaton, in: Z. Kohavi, A. Paz (Eds.), Theory of machines and computations (Proc. Internat. Sympos. Technion, Haifa, 1971), Academic Press, New York, 1971, pp. 189–196.
[16] O. Jenkinson, L. Zamboni, Characterizations of balanced words via orderings, Theoret. Comput. Sci. 310 (2004) 247–271.
[17] T. Knuutila, Re-describing an algorithm by Hopcroft, Theoret. Comput. Sci. 250 (2001) 333–363.
[18] M. Lothaire, Algebraic Combinatorics on Words, in: Encyclopedia of Mathematics and its Applications, vol. 90, Cambridge University Press, 2002.
[19] O. Matz, A. Miller, A. Potthoff, W. Thomas, E. Valkema, Report on the program AMoRE, Technical Report, Inst. f. Informatik u.Prakt. Math., CAU Kiel, 1995.
[20] E.F. Moore, Automata Studies, 1956, pp. 129–153 (chapter Gedaken experiments on sequential machines).
[21] A. Paun, On the Hopcroft's minimization algorithm, CoRR, 2007, abs/0705.1986.
[22] D. Revuz, Minimisation of acyclic deterministic automata in linear time, Theoret. Comput. Sci. 92 (1) (1992) 181–189.
[23] N.J.A. Sloane, The On-line Encyclopedia of Integer Sequences. Published electronically at http://www.research.att.com/~njas/sequences/.
[24] B. Watson, A taxonomy of finite automata minimization algorithms, Technical Report 93/44, Eindhoven University of Technology, Faculty of Mathematics and Computing Science, 1994.