Theoretical Computer Science 411 (2010) 3414–3422

Contents lists available at ScienceDirect

Theoretical Computer Science

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

On extremal cases of Hopcroft's algorithm

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

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

A R T I C L E   I N F O

Keywords:
Deterministic finite state automata
Hopcroft's minimization algorithm
Standard trees
Word trees

A B S T R A C T

In this paper we consider the problem of minimization of deterministic finite automata (DFA) with reference to Hopcroft's algorithm. Hopcroft's algorithm has several degrees of freedom, so there can exist different executions that can lead to different sequences of refinements of the set of the states up to the final partition. We find an infinite family of binary automata for which such a process is unique, whatever strategy is chosen. Some recent papers (cf. Berstel and Carton (2004) [3], Castiglione et al. (2008) [6] and Berstel et al. (2009) [1]) have been devoted to find families of automata for which Hopcroft's algorithm has its worst execution time. They are unary automata associated with circular words. However, automata minimization can be achieved also in linear time when the alphabet has only one letter (cf. Paige et al. (1985) [14]), but such a method does not seem to extend to larger alphabet. So, in this paper we face the tightness of Hopcroft's algorithm when the alphabet contains more than one letter. In particular we define an infinite family of binary automata representing the worst case of Hopcroft's algorithm, for each execution. They are automata associated with particular trees and we deepen the connection between the refinement process of Hopcroft's algorithm and the combinatorial properties of such trees.

1. Introduction

A deterministic finite automaton (DFA) is a recognizer of a regular language and provides a compact representation of the language itself. Among the equivalent deterministic finite automata (i.e. recognizing the same regular language), there exists a unique one (up to isomorphism) with minimal number of states, called the minimal automaton of the language. Describing a regular language by its minimal automaton is important in many applications, such as, for instance, text searching, lexical analysis or coding systems, where space considerations are prominent.

Finding the minimal automaton equivalent to a given DFA is a classical and largely studied problem in Theory of Automata and Formal Languages, also called the automata minimization problem. Several methods have been developed to minimize a deterministic finite automaton. Some of them operate by successive refinements of a partition of the states. For instance, we recall the well known algorithm proposed by Moore in 1956 (cf. [13]) with time complexity $O(kn^2)$, where n is the number of states of the DFA and k is the cardinality of the alphabet. More efficient is the algorithm provided by Hopcroft in 1971 (cf. [9]) where the refinements are computed in $O(kn \log n)$. Furthermore, such an algorithm is the fastest known solution to the automata minimization problem.

A taxonomy of finite automata minimization algorithms is given in [15]. Very recently, many papers on experimental comparison of minimization algorithms have been published.

The general complexity of the automata minimization problem is still an open question, but there are families of automata for which Hopcroft's algorithm runs effectively in $\Theta(n \log n)$ (cf. [3,6,7]). Such families are unary automata associated with circular words. However, automata minimization can be achieved also in linear time when the alphabet has only one letter

[*] Corresponding author. Tel.: +39 09123891051.
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 © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.05.025

```
Hopcroft Minimization ($\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$)
1.  $\Pi \leftarrow \{F, Q \setminus F\}$
2.  for all $a \in \Sigma$ do
3.      $\mathcal{W} \leftarrow \{(min(F, Q \setminus F), a)\}$
4.  while $\mathcal{W} \neq \emptyset$ do
5.      choose and delete any $(C, a)$ from $\mathcal{W}$
6.      for all $B \in \Pi$ do
7.          if $B$ is split from $(C, a)$ then
8.              $B' \leftarrow \delta_a^{-1}(C) \cap B$
9.              $B'' \leftarrow B \setminus \delta_a^{-1}(C)$
10.             $\Pi \leftarrow \Pi \setminus \{B\} \cup \{B', B''\}$
11.             for all $b \in \Sigma$ do
12.                 if $(B, b) \in \mathcal{W}$ then
13.                     $\mathcal{W} \leftarrow \mathcal{W} \setminus \{(B, b)\} \cup \{(B', b), (B'', b)\}$
14.                 else
15.                     $\mathcal{W} \leftarrow \mathcal{W} \cup \{(min(B', B''), b)\}$
```

Fig. 1. Hopcroft's algorithm.

(cf. [14]), but the solution does not seem to extend to a larger alphabet. In this paper we will focus on finding families of automata defined on a more than one letter alphabet representing the worst case of Hopcroft's algorithm. Actually, we provide an infinite family of binary automata defined by binary labeled trees and relate the execution of Hopcroft's algorithm on such automata with some combinatorial properties of the associated binary tree. Recall that, in general, Hopcroft's algorithm has several degrees of freedom since it leaves several choices to the programmer. Hence, there can exist different executions that could produce different sequences of refinements of the set of states leading to the final partition. Even so, for binary automata defined here the refinement process leading from the initial partition of the set of states to the final one is uniquely determined, whatever strategy is used. However, different executions, while producing the same partitions of the states, may have different running time. The main result of this paper is that there exists an infinite subfamily of such automata representing the worst case of Hopcroft's algorithm, for each execution. A preliminary version of this paper appeared in [8].

The paper is organized as follows. Section 2 contains the description of Hopcroft's algorithm by focusing on its degrees of freedom. Section 3 introduces the notion of standard binary tree and standard tree-like automaton. The uniqueness of the refinement process of Hopcroft's algorithm on standard tree-like automata is studied in Section 4. In Section 5 we define the notion of word tree and deepen the problem of tightness of Hopcroft's algorithm, by providing an infinite family of binary automata representing the worst case of the algorithm for each execution. Section 6 is devoted to some conclusions.

2. Hopcroft's algorithm

In 1971 Hopcroft proposed an algorithm for minimizing a deterministic finite state automaton with n states, over an alphabet Σ, in $O(kn \log n)$ time (cf. [9]). This algorithm has been widely studied and described by many authors (see for example [10,12,15]) because of the difficult to give its theoretical justification, to prove correctness and to compute running time.

In Fig. 1 we give a brief description of the algorithm.

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 $Q_j'' = Q_j \setminus \delta_a^{-1}(Q_i)$. Furthermore, the 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 split Q_j.

Hopcroft's algorithm operates by a sequence $\Pi_1, \Pi_2, \ldots, \Pi_l$ of successive refinements of a partition of the states and it is based on the so-called "smaller half" strategy. Actually, it starts from the partition $\Pi_1 = \{F, Q \setminus F\}$ and refines it by means of splitting operations until it obtains a congruence, i.e. until no split is possible. To do that it maintains the current partition Π_i and a set $\mathcal{W} \subseteq \Pi_i \times \Sigma$, called *waiting set*, that contains the pairs for which it has to be checked whether some classes of the current partition are split. The main loop of the algorithm takes and deletes one pair (C, a) from $\mathcal{W}$ and, for each class B of Π_i, checks if it is split by (C, a). If it is the case, the class B in Π_i 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 $\mathcal{W}$ (where $\min(B', B'')$ stands for the smaller of the two sets). 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 has a degree of freedom because the pair (C, a) to be processed at each step is freely chosen. Another free choice intervenes when a set B is split into B' and B'' with the same size and it is not present in $\mathcal{W}$. In this case, the algorithm can, indifferently, add to $\mathcal{W}$ either B' or B''.

Such considerations imply that there can be several executions and several sequences of successive refinements that starting from the initial partition $\Pi_1 = \{F, Q \setminus F\}$ lead to the coarsest congruence of the input automaton $\mathcal{A}$.

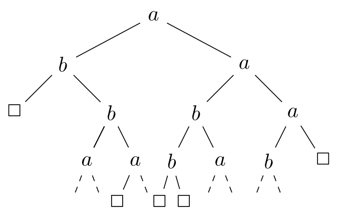

Fig. 2. Binary infinite labeled tree.

As regards the running time of the algorithm 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 processed. Hopcroft proved that the running time is bounded by $O(k|Q|\log|Q|)$. In [3] the authors proved that this bound is tight, in the sense that they provided a family of unary automata for which there exist a sequence of refinements such that the time complexity of the algorithm is $\Theta(k|Q|\log|Q|)$. However, for the same automata there exist other sequences of refinements producing executions that run in linear time. In [6] we presented a family of unary automata for which there is a unique sequence of refinements and a unique execution. Moreover, we defined a subclass of such automata for which the running time is $\Theta(k|Q|\log|Q|)$. Such a subclass of unary automata was extended in [1]. Actually, unary automata represent a very special case for the automata minimization problem. In fact, the minimization can be achieved also in linear time when the alphabet has only one letter (cf. [14]). So, we are interested in facing both the problem of the uniqueness of the refinements and the tightness of the algorithm when the alphabet contains more than one letter.

The next sections are devoted to defining an infinite family of binary automata for which the running time of Hopcroft's algorithm is $\Theta(k|Q|\log|Q|)$, for each execution.

3. Standard trees and tree-like automata

In this section we present a class of binary automata defined by using the notion of binary labeled tree.

Let $\Sigma = \{0, 1\}$ and $A = \{a, b\}$ be two binary alphabets. A *binary labeled tree* over A is a map $\tau : \Sigma^* \to A$ whose domain $dom(\tau)$ is a prefix-closed subset of Σ^*. The elements of $dom(\tau)$ are called *nodes*, if $dom(\tau)$ has a finite (resp. infinite) number of elements we say that τ is *finite* (resp. *infinite*). The *height* of a finite tree τ, denoted by $h(\tau)$, is defined as $\max\{|u| + 1, u \in dom(\tau)\}$. Actually, in the literature, factors of height h are sometimes considered of height $h - 1$ (cf. [5]). In this paper, the height of a tree is the number of nodes along a maximal branch and not the length of the path. Our convention is motivated by the fact that we are interested in the nodes and their labels. We say that a tree $\bar{\tau}$ is a *prefix* of a tree τ if $dom(\bar{\tau}) \subseteq dom(\tau)$ and $\bar{\tau}$ is the restriction of τ to $dom(\bar{\tau})$. A *complete infinite tree* is a tree whose domain is Σ^*. Besides, a *complete finite tree* of height n is a tree whose domain is Σ^{n-1}. The *empty tree* is the tree whose domain is the empty set.

If $x, y \in dom(\tau)$ are nodes of τ such that $x = yi$ for some $i \in \Sigma$, we say that y is the *father* of x and in particular, if $i = 0$ (resp. $i = 1$) x is the *left son* (resp. *right son*) of y. A node without sons is called a *leaf* and the node ϵ is called *the root* of the tree. Given a tree τ, the *outer frontier* of τ is the set $Fr(\tau) = \{xi | x \in dom(\tau), i \in \Sigma, xi \notin dom(\tau)\}$.

Example 1. In Fig. 2 an example of an infinite tree γ is depicted. We have, for instance, 0111, 1011 $\in dom(\gamma)$ and 0110, 1001, 1000 $\in Fr(\gamma)$. In Fig. 3 the finite tree τ is depicted.

The nodes belonging to the outer frontier are represented by a box.

Let τ and τ' be two binary labeled trees. We say that τ is a *subtree* of τ' if there exist a node $v \in dom(\tau')$ such that:

(i) $v \cdot dom(\tau) = \{vu | u \in dom(\tau)\} \subseteq dom(\tau')$
(ii) $\tau(u) = \tau'(vu)$ for all $u \in dom(\tau)$.

In this case we say that τ is a subtree of τ' that *occurs at* node v.

In [11] operations among trees have been introduced. Here, we are interested in the concatenation among trees. Roughly speaking, we can say that in order to concatenate two trees τ_1 and τ_2 we attach the root of τ_2 to one of the element of the outer frontier of τ_1. Obviously, since the outer frontier of τ_1 can have more than one element, by concatenating τ_1 and τ_2 we obtain a set of trees. We define the *simultaneous concatenation* of τ_1 and τ_2 the tree $\tau_1 \circ \tau_2$ such that:

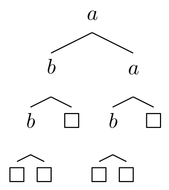

Fig. 3. A finite tree τ.

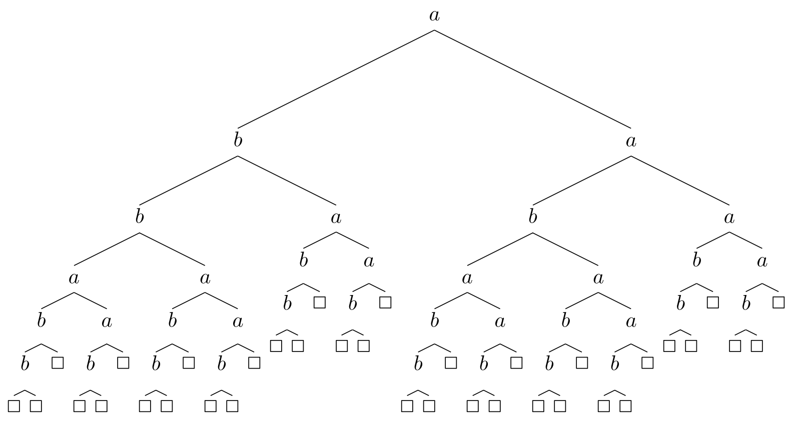
Fig. 4. The tree $\tau \circ \tau$.

(i) $dom(\tau_1 \circ \tau_2) = dom(\tau_1) \cup Fr(\tau_1)dom(\tau_2)$;

(ii) $\forall x \in dom(\tau_1 \circ \tau_2)$, $\tau_1 \circ \tau_2(x) = \begin{cases} \tau_1(x) & \text{if } x \in dom(\tau_1) \\ \tau_2(y) & \text{if } x = zy, z \in Fr(\tau_1), y \in dom(\tau_2). \end{cases}$

Let τ be a tree, with τ^ω we denote the infinite simultaneous concatenation $\tau \circ \tau \circ \tau \circ \ldots$. Notice that, by infinitely applying the simultaneous concatenation, we obtain a complete infinite tree. In Fig. 4 $\tau \circ \tau$ is depicted.

We define a *factor* of a tree a finite complete subtree of the tree, and in the following we are interested in particular factors we define by using some notations given in [5].

Let τ be a tree, σ and $\bar{\sigma}$ two factors of τ such that $\bar{\sigma}$ is the complete prefix of σ of height $h(\sigma) - 1$, then σ is called an *extension of* $\bar{\sigma}$ in τ. A factor σ of a tree τ is *extendable* in τ if there exists at least one extension of σ in τ.

A factor σ of τ is 2-*special* if there exist exactly two different extensions of σ in τ.

We say that γ is a *circular factor* of τ if it is a factor of τ^ω with $h(\gamma) \leq h(\tau)$. A circular factor γ of τ is a *special circular factor* if there exist at least two different extensions of γ in τ^ω (that we can call *circular extensions* or simply extensions). A special factor is called 2-*special circular factor* if it has exactly two different extensions.

Example 2. In Fig. 5 three factors of the tree τ (see Fig. 3) are depicted. The single node labeled by b is a 2-special circular factor, indeed it has two different extensions depicted in Fig. 5(a) and (b). The single node labeled by a has a unique extension depicted in Fig. 5(c).

The concept of circular factor can be easily understood by noting that in the case of unary tree it coincides with the well-known notion of circular factor of a word.

With reference to a characterization of the notion of circular standard word given in [4], we say that a finite tree τ is a *standard tree* if for each $0 \leq h \leq h(\tau) - 2$ it has only a 2-special circular factor of height h.

Remark 1. In fact, one can easily prove that a tree τ is standard if and only if, for each $k = 1, \ldots, h(\tau) - 1$, τ has exactly $k + 1$ circular factors of height k.

Example 3. An example of standard tree, called *finite uniform tree*, is a complete tree defined by labeling all the nodes at the same level with the same letter taken in the same order it occurs in a given standard word. In Fig. 6 we give the uniform tree from the word *abaab*.

Example 4. In Fig. 7(a) a tree that is not standard is depicted. Indeed the trees in Fig. 7(b) and (c) have three different extensions and the tree in Fig. 7(d) is a 2-special circular factor.

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*, $q_0 \in Q$ is the *initial state* and $F \subseteq Q$ the set of *final states*. Let $G = (V, E)$ be the transition directed graph associated with $\mathcal{A}$. We say that $\mathcal{A}$ is a *tree-like automaton* if $G = (V, E)$ has a subgraph $G_t = (V, E_t)$, containing all nodes V, which is a tree (called *skeleton*) with root q_0, and such that all edges of $E \backslash E_t$ are edges from a node to an ancestor.

Given a finite binary labeled tree τ we can uniquely associate a tree-like automaton $\mathcal{A}_\tau$ having τ as skeleton and such that for each missing edge we add a transition to the root of the tree. Moreover, the root is the initial state and the states corresponding to nodes labeled by a (resp. b) are non-final (resp. final) states.

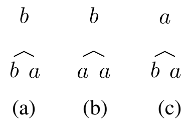
Fig. 5. Three factors of the tree τ depicted in Fig. 3.

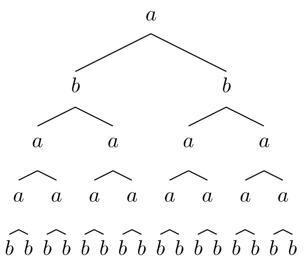

Fig. 6. The finite uniform tree defined from the word *abaab*.

Example 5. In Fig. 8 a finite labeled tree and the corresponding tree-like automaton are depicted. In the automaton, the initial state labeled by 1 corresponds to the root of the tree.

4. Hopcroft's algorithm on standard tree-like automata

In this section we deepen the connection between the refinement process of Hopcroft's algorithm when applied on a tree-like automaton associated with a standard tree and the combinatorial properties of the tree itself. By using such properties we prove that in such a case the refinement process is uniquely determined.

We define *standard tree-like automaton* a tree-like automaton $\mathcal{A}_\tau$ associated with a standard tree τ. The main theorem of this section gives a characterization of the partitions representing the refinement process of Hopcroft's algorithm on standard tree-like automata.

Let $\mathcal{A}_\tau = (Q, \Sigma, \delta, q_0, F)$ be a tree-like automaton. For any circular factor σ of τ, we define the subset Q_σ of states of $\mathcal{A}_\tau$ that are occurrences of σ in τ. Trivially, we have that $Q_\epsilon = Q$, $Q_b = F$ and $Q_a = Q \setminus F$.

The following proposition and corollary establish a close relation between the split operation during the execution of Hopcroft's algorithm on a standard tree-like automaton and the notion of circular special factor of a standard tree.

Proposition 1. *Let $\mathcal{A}_\tau$ be a standard tree-like automaton. Let Q_σ and Q_γ be classes of a partition of Q. If (Q_γ, x) splits Q_σ, for some $x \in \Sigma$, then $h(\gamma) \geq h(\sigma)$ and σ is a prefix of a circular 2-special factor of τ.*

Proof. Let us suppose $(Q_\gamma, 0)$ splits Q_σ (analogously for $(Q_\gamma, 1)$). We have that $\delta_0^{-1}(Q_\gamma) \cap Q_\sigma \neq \emptyset$ and $\delta_0^{-1}(Q_\gamma) \cap Q_\sigma^c \neq \emptyset$. If $h(\gamma) < h(\sigma)$ then γ is a prefix of the left subtree of σ i.e. $Q_\sigma \supseteq \delta_0^{-1}(Q_\gamma)$, i.e. $(Q_\gamma, 0)$ does not cause a split of Q_σ. Hence, we proved that $h(\gamma) \geq h(\sigma)$. In the case that $h(\gamma) = h(\sigma)$, if $(Q_\gamma, 0)$ splits Q_σ in Q'_σ and Q''_σ we have that Q'_σ is the set of occurrences of σ such that γ is the extension of height $h(\sigma)$ of the left complete subtree of σ. On the contrary, Q''_σ is the set of occurrences of σ such that the extension of height $h(\sigma)$ of the left complete subtree of σ is different from γ then we can deduce that σ has at least two different extensions. Since τ is standard, σ is a circular 2-special factor of τ. We denote by σ' and σ'' the two extensions therefore $Q'_\sigma = Q_{\sigma'}$ and $Q''_\sigma = Q_{\sigma''}$. In the case $h(\gamma) > h(\sigma)$, if $(Q_\gamma, 0)$ splits Q_σ in Q'_σ and Q''_σ then the occurrences of σ that are in Q'_σ have γ as complete left subtree, and the occurrences of σ that are in Q''_σ have, as complete left subtree, $\gamma' \neq \gamma$. Obviously, γ and γ' differ at a level greater than $h(\sigma) - 1$. Hence, σ is a prefix of a tree with two different extensions, i.e. since τ is standard, σ is a prefix of a 2-special factor of τ. $\square$

From the proof of the previous proposition it is possible to state the following corollary.

Corollary 2. *Let $\mathcal{A}_\tau$ be a standard tree-like automaton. Let Q_σ and Q_γ be classes of a partition of Q. If (Q_γ, x) splits Q_σ, for some $x \in \Sigma$, with $h(\gamma) = h(\sigma)$, then σ is a 2-special circular factor of τ. The resulting classes are $Q_{\sigma'}$ and $Q_{\sigma''}$, where σ' and σ'' are the only two possible extensions of σ in τ. Viceversa, if σ is a 2-special circular factor of τ and Q_σ is a class of a partition of Q then Q_σ is split in $Q_{\sigma'}$ and $Q_{\sigma''}$ where σ' and σ'' are the unique two extensions of σ in τ.*

Remark 2. If the circular factor σ is not special, i.e. it has only one extension σ' then $Q_\sigma = Q_{\sigma'}$.

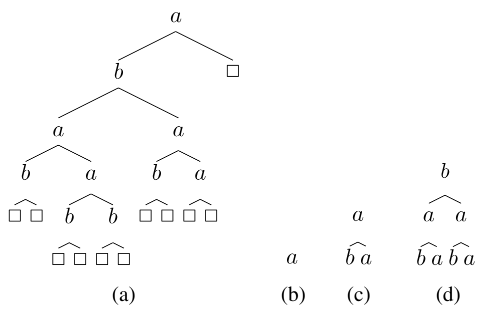

Fig. 7. A not standard tree τ.

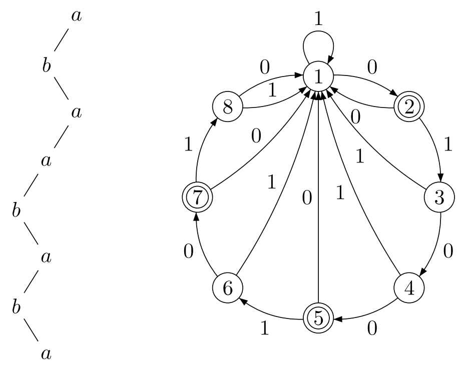

Fig. 8. A tree τ and tree-like automaton $\mathcal{A}_\tau$.

At each iteration of the main loop of the algorithm, the pair extracted from the waiting set can either cause some splits or not. Hence, at each step, the current partition can be either equal to or different from that one of the previous iteration. We call *refinement process* the sequence $\Pi_1, \Pi_2, \ldots, \Pi_m$ of the different partitions produced during an execution of the algorithm, where $\Pi_1 = \{F, Q \setminus F\}$ and Π_m is the Nerode equivalence. The following theorem states that, in the case of standard tree-like automata, the sequence of partitions created during the refinement process of Hopcroft's algorithm is unique whatever strategy is used for choosing and deleting any pair from the waiting set. The statement of the theorem characterizes the partition after each split operation of Hopcroft's algorithm on a standard tree-like automaton.

Theorem 3. *Let $\mathcal{A}_\tau$ be a standard tree-like automaton. The refinement process $\Pi_1, \ldots, \Pi_m$ is uniquely determined. Furthermore, $m = h(\tau) - 1$ and for each $1 \le k \le h(\tau) - 1$,*

$$\Pi_k = \{Q_\sigma \,|\, \sigma \text{ is a circular factor of } \tau \text{ with } h(\sigma) = k\}.$$

Proof. Let $\mathcal{A}_\tau$ be a standard tree-like automaton and Π_k the k-th partition of the refinement process. We prove the statement by induction on k.
If $k = 1$, $\Pi_1 = \{F = Q_b, Q \setminus F = Q_a\}$.
We suppose $\Pi_{k-1} = \{Q_\sigma \,|\, \sigma$ is a circular factor of τ with $h(\sigma) = k - 1\}$, with $k - 1 < h(\tau) - 1$. Since τ is standard and $k-1 < h(\tau)-1$ then there exists a 2-special circular factor γ of height $k-1$. $Q_\gamma \in \Pi_{k-1}$ then Π_{k-1} is not the final partition, i.e. there exists a pair in the waiting set that splits Q_γ in $Q_{\gamma'}$ and $Q_{\gamma''}$, where γ' and γ'' are the only two extensions of γ. By Proposition 1, Q_γ is the unique element of Π_{k-1} that is split. It follows that each $\sigma \neq \gamma$ has a unique extension σ' (i.e. $Q_\sigma = Q_{\sigma'}$), then

$$\begin{aligned}\Pi_k &= \{Q_{\gamma'}, Q_{\gamma''}\} \cup \{Q_{\sigma'} |\ \sigma \text{ is a circular factor of } \tau \text{ with } h(\sigma) = k-1,\ \sigma \neq \gamma\}\\ &= \{Q_\sigma \,|\, \sigma \text{ is a circular factor of } \tau \text{ with } h(\sigma) = k\}.\end{aligned}$$

The final partition is obtained when $k = h(\tau) - 1$, because no circular factor of height $h(\tau) - 1$ can be 2-special. □

Remark 3. The theorem states also that the partition Π_k of the set of states has exactly $k + 1$ classes.

One can pose the problem to characterize the family of all automata for which the refinement process is uniquely determined. It is still an open question whether there exist some tree-like automata that are not standard for which the refinement process is unique. For instance, one can verify that for the tree-like automaton associated with the non-standard tree depicted in Fig. 7 the refinement process produced by Hopcroft's algorithm is not unique.

5. Word trees

The aim of this section is to compute the running time of Hopcroft's algorithm on the automata associated with the word trees here defined. The main results of the section make use of a close relation between the notion of circular factors of words and trees.

Recall that a *circular factor* of a word w, over the alphabet $A = \{a, b\}$, is a factor of ww of length not greater than the length of w denoted by $|w|$. Moreover, a circular factor u of w is said to be *special* if both ua and ub are circular factors of w.

Let $\Sigma = \{0, 1\}$ and $A = \{a, b\}$. Given two words $v = v_1v_2 \ldots v_{n-1} \in \Sigma^*$ and $w = w_1w_2 \ldots w_n \in A^*$, by $\tau_{v,w}$ we denote the labeled tree such that $dom(\tau_{v,w})$ is the set of prefixes of v and the map is defined as follows:

$$\begin{cases}\tau_{v,w}(\epsilon) = w_1\\ \tau_{v,w}(v_1v_2 \ldots v_i) = w_{i+1} \quad \forall 1 \le i \le n-1.\end{cases}$$

We call *word tree* the finite labeled tree $\tau_{v,w}$. When v is obtained by taking the prefix of length $n-1$ of w and by substituting a's with 0's and b's with 1's, we use the simpler notation τ_w. In Fig. 8 a word tree τ_w with $w = abaababa$ is depicted.

Our investigation is focused on word trees associated with standard words.

We recall the well known notion of standard word. Let $d_1, d_2, \ldots, d_n, \ldots$ be a sequence of natural integers, with $d_1 \geq 0$ and $d_i > 0$, for $i = 2, \ldots, n, \ldots$. Consider the following sequence of words $\{s_n\}_{n \geq 0}$ over the alphabet A: $s_0 = b$, $s_1 = a$, $s_{n+1} = s_n^{d_n} s_{n-1}$ for $n \geq 1$. Each finite word s_n in the sequence is called a *standard word*. It is uniquely determined by the (finite) directive sequence $(d_0, d_1, \ldots, d_{n-1})$. In the special case where the directive sequence is of the form $(1, 1, \ldots, 1, \ldots)$ we obtain the sequence of Fibonacci words.

Proposition 4. *Let τ_w be a word tree. There exists a one-to-one correspondence ψ between the set of circular factors of w and the set of circular factors of τ_w. Furthermore, if u is a circular factor of w then $|Q_u| = |Q_{\psi(u)}|$.*

Proof. Let φ be the application from A to Σ such that $\varphi(a) = 0$ and $\varphi(b) = 1$. Let σ be a circular factor of τ_w of height h. It uniquely individuates a circular factor $u_1 u_2 \ldots u_n$ of w as follows. If x is the label of the root of σ, $u_1 = \sigma(\varphi(x))$ and $u_i = \sigma(\varphi(u_{i-1}))$, with $1 < i \leq h$. Roughly speaking, we follow the root to leaf path such that for each node we take the left son if the node is labeled by a and the right son otherwise.

Conversely, if u is a circular factor of w of length h then $\psi(u) = \tau_u$ is a subtree of τ_w and then it uniquely determines a circular factor of τ_w of height h.

We have that each occurrence of u in w corresponds to a state of the automaton that is an occurrence of τ_u in τ_w and viceversa. Hence the thesis. □

Hence, we have the following corollaries.

Corollary 5. *w is a standard word if and only if the word tree τ_w is a standard tree.*

Proof. It follows by the fact that u is a circular special factor of w if and only if $\psi(u)$ is a 2-special circular factor of τ_w. □

Corollary 6. *Two words w and w' are conjugates if and only if the trees τ_w and $\tau_{w'}$ have the same circular factors.*

The standard word trees represent an instance of standard trees that is opposite to the finite uniform tree described in Section 3. The difference consists in the fact that the two extensions of each 2-special circular factor differ only by a leaf, while in the case of a uniform tree all the leaves are involved.

Lemma 7. *Let τ_w be a standard word tree and $\mathcal{A}_{\tau_w}$ the associated automaton. Let σ and γ be two circular factors of τ_w having the same height. If $(Q_\gamma, 0)$ (resp. $(Q_\gamma, 1)$) splits Q_σ then $(Q_\gamma, 1)$ (resp. $(Q_\gamma, 0)$) does not split Q_σ.*

Proof. If $(Q_\gamma, 0)$ splits Q_σ then σ is a 2-special circular factor and by Proposition 4 σ individuates a special circular factor u of w that, trivially has two possible extensions $u0$ and $u1$. Each of them determines the two extensions σ' and σ'' of σ that differ only in the leaf reached by the path labeled $u0$ and $u1$, respectively. Furthermore, we have that these differences occur in the left subtrees of σ' and σ''. By definition of word tree τ_w, we have that the right subtrees of σ' and σ'' are equal. It means that Q_γ can not split Q_σ with respect to the letter 1. □

We know that for each automaton several executions of the algorithm can exist. We denote by $\mathbf{c}(\mathcal{A})$ the running time of Hopcroft's algorithm to minimize $\mathcal{A}$ in the worst execution. Note that, in order to evaluate the running time of an execution of the algorithm we compute the sum of the sizes of the classes extracted from the waiting set.

Proposition 8. *Let τ_w be a standard word tree. Then*

$$\mathbf{c}(\mathcal{A}_{\tau_w}) = 2 \sum_{\sigma \in sp(\tau_w)} \min(|Q_{\sigma'}|, |Q_{\sigma''}|),$$

where with $sp(\tau_w)$ we denote the set of 2*-special circular factor of τ_w.*

Proof. Firstly, we recall that the running time is proportional to the cardinality of the classes processed of the waiting set. So, in order to make the computation of $\mathbf{c}(\mathcal{A})$ we have to establish which classes each time go into the waiting set and in particular, which classes are extracted and processed. We know, by Corollary 2, that a class Q_σ is split when it is the set of occurrences of a 2-special circular factor of τ_w. Then, at each step we have at most a unique split of Q_σ in $Q_{\sigma'}$ and $Q_{\sigma''}$. If (Q_σ, x) is in the waiting set there is a substitution of (Q_σ, x) with $(Q_{\sigma'}, x)$ and $(Q_{\sigma''}, x)$ and an addition of $(\min(Q_{\sigma'}, Q_{\sigma''}), y)$, with $x \neq y$. Otherwise, if (Q_σ, x) does not belong to the waiting set there is the addition of $(\min(Q_{\sigma'}, Q_{\sigma''}), 0)$ and $(\min(Q_{\sigma'}, Q_{\sigma''}), 1)$. Then, the worst execution is obtained when after each split the minimal class is added to the waiting set each time with both the two letters of the alphabet. We prove that such an execution can be achieved. By Lemma 7 if the waiting set contains both $(C, 0)$ and $(C, 1)$, for some class C, the worst execution is obtained by extracting the splitting pair after the other one. Indeed, by following such a strategy, starting from the first step, it is easy to see that, each time a splitter pair is extracted, the waiting set is empty. Hence, after the split we add to the waiting set the $(\min(Q_{\sigma'}, Q_{\sigma''}), 0)$ and $(\min(Q_{\sigma'}, Q_{\sigma''}), 1)$. □

In [7,1] the authors consider the unary cyclic automaton $\mathcal{A}_w$ associated with a word w and give an exact computation of the running time of the algorithm on these automata, when w is a standard word. In particular, in [7], the authors prove that there is a unique execution of the algorithm for these automata and that $\mathbf{c}(\mathcal{A}_w) = \sum_{u \in sp(w)} \min(|Q_{u0}|, |Q_{u1}|)$, where $sp(w)$ is the set of special factors of w. For sake of brevity we refer to [3] for the definition of the notion of unary cyclic automaton. Intuitively, such an automaton is a tree-like automaton associated with a word tree $\tau_{v,w}$ such that v is a word in a unary alphabet.

The following result relates the worst running time $\mathbf{c}(\mathcal{A}_{\tau_w})$ of Hopcroft's algorithm on the automaton $\mathcal{A}_{\tau_w}$ and the running time $\mathbf{c}(\mathcal{A}_w)$ when w is a standard word. Recall that, since in the unary case the execution is unique, $\mathbf{c}(\mathcal{A}_w)$ is the exact running time of the algorithm on $\mathcal{A}_w$.

Proposition 9. *Let τ_w be a standard word tree and w the corresponding standard word.*

$$\mathbf{c}(\mathcal{A}_{\tau_w}) = 2\mathbf{c}(\mathcal{A}_w).$$

Proof. Remark that, by Proposition 4 and Corollary 5, there is a one-to-one correspondence between the set of the 2-special circular factors of τ_w and the set of the special factors of w. It follows that if u is a circular special factor of w and σ the corresponding 2-special circular factor of τ_w, then $|Q_u| = |Q_\sigma|$. □

The previous proposition states that Hopcroft's algorithm on a binary standard tree-like automaton inherits all the worst cases obtained when it is applied on unary cyclic automata associated with standard words. The unary automata representing the worst cases of Hopcroft's algorithm are the unary cyclic automata $\mathcal{A}_{f_n}$ where f_n is the Fibonacci word of order n. Let $F_n = |f_n| = |Q|$, in [7] the exact computation of $\mathbf{c}(\mathcal{A}_{f_n})$ is given as follows.

Theorem 10. *Let $\mathcal{A}_{f_n}$ be a unary cyclic automaton associated with the standard word f_n, with $n \geq 0$, then*

$$\frac{K}{\phi} F_n \log F_n \leq \mathbf{c}(\mathcal{A}_{f_n}) \leq K F_n \log F_n,$$

where $K = \frac{3}{5 \log \phi}$ and ϕ is the golden ratio $\frac{1+\sqrt{5}}{2}$.

Note that for these unary automata both the refinement process and the execution is unique, the inequality has the simple arithmetic justification that $F_n = [\frac{\phi^n}{\sqrt{5}}]$, where $[x]$ is the nearest integer function.

The extension to the binary case is formalized in the following theorem.

Theorem 11. *Hopcroft's algorithm on tree-like automata $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ associated with standard word trees runs in time $\Theta(|\Sigma||Q| \log |Q|)$.*

Hence, the theorem states that there exist automata on a binary alphabet for which the upper bound for the time complexity of Hopcroft's algorithm is achieved. From Proposition 9 and Theorem 10, an infinite family of such automata is obtained by considering binary automata associated with the word trees τ_{f_n}, where f_n is the Fibonacci word of order n, for each $n \geq 0$. Although for these standard automata the refinement process is unique, there may be different executions that produces the same sequence of partitions of the states. Such executions may have different running time. In the following theorem we prove that each execution has a time complexity $\Theta(|Q| \log |Q|)$. With $t(F_n)$ we denote the running time of an execution of Hopcroft's algorithm on $\mathcal{A}_{\tau_{f_n}}$.

Remark 4. One can prove that during the refinement process of Hopcroft's algorithm, each class of any partition has as cardinality a Fibonacci number and, when it is split, the cardinalities of the resulting classes are the two preceding Fibonacci numbers, respectively.

Theorem 12. *Let $\mathcal{A}_{\tau_{f_n}}$ be a standard automaton associated with the standard tree τ_{f_n}, with $n \geq 0$. Each execution of Hopcroft's algorithm on this automaton has a running time that satisfies the following inequalities:*

$$\frac{K}{\phi} F_n \log F_n + F_n - 1 \leq t(F_n) \leq 2K F_n \log F_n,$$

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

Proof. The second inequality follows from Theorem 10 and Proposition 9. In order to prove the first one we compute the running time of the best execution of the algorithm on the automaton.
Let $\Pi_1, \Pi_2, \ldots, \Pi_{F_n - 1}$ be the unique sequence of refinements of the set of the states. There exists a unique set of classes $P = \{Q_{\sigma_1}, Q_{\sigma_2}, \ldots, Q_{\sigma_{F_n-1}}\}$ such that for each $1 \leq i \leq F_n - 1$ one has that Q_{σ_i} is never added to the waiting set and $Q_{\sigma_i} \subset Q_{\sigma_{i+1}}$. By considering Remark 4, one can note that such classes have, as cardinality, $F_n, F_{n-1}, \ldots, F_1$, respectively.

As soon as a class $Q_{\sigma_i} \in P$ is split, the minimal class between $Q_{\sigma_i'}$ and $Q_{\sigma_i''}$ is added to the waiting set paired both with 0 and 1. Each of these resulting classes has as cardinality $F_{n-2}, F_{n-3}, \ldots, F_0$, respectively.

All the classes not belonging to P and included in some partition of the set of states will be added to the waiting set after some split. The best execution of the algorithm is obtained by extracting at each step the unique splitter class. By following such a strategy, if (Q_σ, x) is in the waiting set and it is not a splitter then it will be never extracted but, when Q_σ is split, (Q_σ, x) will be substituted by $(Q_{\sigma'}, x)$ and $(Q_{\sigma''}, x)$, that are not splitter too, and the pair $(\min(Q_{\sigma'}, Q_{\sigma''}), y)$, with $y \neq x$, will be added to the waiting set. On the contrary, the pair added after the split of a class in P contributes one more time to the size of the waiting set. At each step of the refinement process during this execution, if σ is a 2-special circular factor either (Q_σ, x) is in the waiting set for some x or Q_σ belongs to P. In any case the pair $(\min(Q_{\sigma'}, Q_{\sigma''}), y)$ is added with $y \neq x$ and, in particular, if $Q_\sigma \in P$, $(\min(Q_{\sigma'}, Q_{\sigma''}), x)$ is added too. Then the running time is $\sum_{\sigma \in Sp(\tau_{f_n})} \min(|Q_{\sigma'}|, |Q_{\sigma''}|) + \sum_{i=0}^{n-2} F_i = c(\mathcal{A}_{f_n}) + F_n - 1 \geq \frac{K}{\phi} F_n \log F_n + F_n - 1$. The last inequality follows by Theorem 10. $\square$

6. Conclusions

In this paper we face the problem of minimization of deterministic finite automata with reference to Hopcroft's algorithm. We consider the refinement processes of the algorithm and we exhibit an infinite family of binary automata for which there is a unique process. It would be interesting to give a characterization of the automata for which the sequence of successive refinements is uniquely determined.

Moreover we deepen the tightness of the algorithm when the alphabet contains more than one letter. In particular we define an infinite family of binary automata representing the worst case of Hopcroft's algorithm, whatever execution is considered. Such a family is defined by using the notion of word tree.

Remark that both the notions of word tree and standard tree here defined can arouse an independent interest because they are closely related to a class of infinite trees, called *Sturmian Trees* (cf. [2]). Actually the word trees allow one to construct an infinite family of Sturmian trees having some interesting combinatorial properties as, for instance, the balance.

References

[1] J. Berstel, L. Boasson, O. Carton, Continuant polynomials and worst-case behavior of Hopcroft's minimization algorithm, Theoretical Computer Science 410 (2009) 2811–2822.
[2] J. Berstel, L. Boasson, O. Carton, I. Fagnot, Sturmian trees, Theory of Computing Systems 46 (3) (2010) 443–478.
[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.P. Borel, C. Reutenauer, On Christoffel classes, RAIRO-Theoretical Informatics and Applications 450 (2006) 15–28.
[5] A. Carpi, A. de Luca, S. Varricchio, Special factors and uniqueness conditions in rational trees, Theory of Computing Systems 34 (4) (2001) 375–395.
[6] G. Castiglione, A. Restivo, M. Sciortino, Hopcroft's algorithm and cyclic automata, in: Carlos Martin-Vide, Friedrich Otto, Henning Fernau (Eds.), LATA, in: Lecture Notes in Computer Science, vol. 5196, Springer, 2008, pp. 172–183.
[7] G. Castiglione, A. Restivo, M. Sciortino, Circular Sturmian words and Hopcroft's algorithm, Theoretical Computer Science 410 (2009) 4372–4381.
[8] G. Castiglione, A. Restivo, M. Sciortino, On extremal cases of Hopcroft's algorithm, in: Sebastian Maneth (Ed.), CIAA, in: Lecture Notes in Computer Science, vol. 5642, Springer, 2009, pp. 14–23.
[9] J.E. Hopcroft, An $n \log n$ algorithm for mimimizing 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.
[10] T. Knuutila, Re-describing an algorithm by Hopcroft, Theoretical Computer Science 250 (2001) 333–363.
[11] S. Mantaci, A. Restivo, Codes and equations on trees, Theoretical Computer Science 255 (1–2) (2001) 483–509.
[12] O. Matz, A. Miller, A. Potthoff, W. Thomas, E. Valkema, Report on the program AMoRE. Technical Report 9507, Inst. f. Informatik u.Prakt. Math., CAU Kiel, 1995.
[13] E.F. Moore, Automata Studies, in: Gedaken Experiments on Sequential Machines, 1956, pp. 129–153.
[14] R. Paige, R.E. Tarjan, R. Bonic, A linear time solution to the single function coarsest partition problem, Theoretical Computer Science 40 (1985) 67–84.
[15] B. Watson, A taxonomy of finite automata minimization algorithms, Technical Report 93/44, Eindhoven Univ. of Tech., Faculty of Math. and Comp. Sc., 1994.