Information Sciences 370–371 (2016) 204–220

Contents lists available at ScienceDirect

Information Sciences

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

A sufficient condition to polynomially compute a minimum separating *DFA*

Manuel Vázquez de Parga, Pedro García, Damián López*

Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Spain

ARTICLE INFO

Article history:
Received 25 July 2015
Revised 15 July 2016
Accepted 21 July 2016
Available online 22 July 2016

Keywords:
Minimal separating *DFA*
Minimal consistent *DFA*
Model checking
Minimization of incompletely specified automata

ABSTRACT

The computation of a minimal separating automaton (MSA) for regular languages has been studied from many different points of view, from synthesis of automata or Grammatical Inference to the minimization of incompletely specified machines or Compositional Verification. In the general case, the problem is NP-complete, but this drawback does not prevent the problem from having a real application in the above-mentioned fields. In this paper, we propose a sufficient condition that guarantees that the computation of the MSA can be carried out with polynomial time complexity.

1. Introduction

In this work, we study the problem of computing a minimal separating automaton (MSA) for regular languages. This problem has been studied from many different points of view. We note that, in the general case, the decision problem is NP-complete. This complexity result can be derived from the results on synthesis of automata by Trakhtenbrot and Barzdin [30], who also state a (strict) condition that guarantees polynomial computation. In the Grammatical Inference (GI) framework, Gold proves in [14] that the decision problem of obtaining a *DFA* with a given number of states that is compatible with a finite (positive and negative) sample is NP-complete. In the same GI framework, Angluin proves in [1] that even a small modification of the condition stated by Trakhtenbrot and Barzdin implies that the problem is NP-complete. Also, in [25], Pfleeger studies the complexity of the *minimization of incompletely specified finite state machines* obtaining the same complexity bound.

Briefly speaking, all of these problems can be enunciated as the problem of computing (given any two (regular) languages L_+ and L_-) an automaton with the smallest number of states that accepts the strings in L_+ and rejects all the strings in L_- (the behavior of the automaton with respect to the strings not in $L_+ \cup L_-$ is irrelevant). Despite the exponential time complexity in the worst case of the general problem, in our work we prove a sufficient condition that guarantees that the computation can be carried out with polynomial time complexity.

One of the first approaches to the problem was presented by Trakhtenbrot and Barzdin [30], where the authors study the problem of computing the minimum deterministic finite automaton (*DFA*) which is consistent with respect to a finite set of strings of a target language and its complement. In their work, the authors prove that the *DFA* can be obtained with

[*] Corresponding author. Fax: 34 963877359.
E-mail addresses: mvazquez@dsic.upv.es (M. Vázquez de Parga), pgarcia@dsic.upv.es (P. García), dlopez@dsic.upv.es (D. López).

http://dx.doi.org/10.1016/j.ins.2016.07.053
0020-0255/© 2016 Elsevier Inc. All rights reserved.

polynomial complexity whenever a *uniformly-complete* sample is available (a sample that exclusively contains every string over the alphabet up to a given length).

Several authors study the computation of the *minimal cover-automaton* as a compact representation of a finite set of strings over an alphabet [6,7,18]. Taking into account the results by Trakhtenbrot and Barzdin, the computation of the minimal cover-automaton can be stated as the problem of obtaining the minimum *DFA* such that L_+ is finite and L_- is the language that contains the strings not in L_+ whose length is lower than or equal to an integer n that denotes the length of the longest string in L_+. This allows the finite set L_+ to be described by using the cover-automaton obtained together with n.

As mentioned above, another problem that is related to the computation of the MSA is the minimization of incompletely specified state machines, where incomplete means that either the transition function or the membership of some states to the set of accepting states is undefined. Thus, by taking into account the result of the analysis of any given string using an incompletely specified machine, it is possible to distinguish a set of strings that are accepted, a set of strings that are rejected, and a third set of strings that can either be accepted or rejected. Among the different approaches to the problem, in [22], the authors address the task by enumerating the possible reductions of the input machine and selecting the minimum one. Despite its complexity, the method has been used recently (i.e., in [8]). In the context of circuit design, in [27] propose exact methods (based on the computation of sets of *compatible* states) as well as heuristics. In [23], in a more general approach, Pena and Oliveira propose a method that takes into account previous GI methods. With the exception of the heuristics proposed, none of the enumerated works bypass the exponential complexity of the problem.

In some circumstances, the use of state machines in the modeling of algorithms, sequential logic circuits and communication protocols allows the verification of the system to be reduced to the computation of a MSA. *Compositional verification* is one of these approaches and is considered to be a way to scale up Model Checking [9]. Once two components M_1 and M_2 and the property P (to hold) are characterized by means of regular languages, it is possible to check that the composition of M_1 and M_2 (the intersection of the component languages) fulfills the property (the intersection is a sublanguage of $L(P)$) if there is a contextual assumption A such that the following inference rule is satisfied:

$$\frac{L(M_1)\cap L(A)\subseteq L(P) \qquad L(M_2)\subseteq L(A)}{L(M_1)\cap L(M_2)\subseteq L(P)}$$

The main drawback of applying this *assume-guarantee* rule is the need for expert knowledge in order to obtain the contextual assumptions, while the minimality of the assumption model is important in terms of performance [8]. When regular models are considered, this approach to Model Checking can be reduced to the problem of finding the MSA for the languages $L(M_2)$ (which plays the role of L_+) and $L(M_1)-L(P)$ (which plays the role of L_-).

Among the results in this field, in [15], the authors address the task in the context of the design of logic circuits and propose a heuristic that iteratively constructs a *contradicting sequence* that is used to find incompatible states. In [8], the authors use a version of the L^* algorithm by Angluin [2] in order to obtain an incompletely specified state machine that is then reduced by using the algorithm proposed in [22]. In [21], Neider addresses the problem by representing the desired properties of the *DFA* in terms of a logical formula and using standard SAT or SMT solvers to find a solution.

As mentioned above, the goal of Grammatical Inference is to obtain the MSA in the special case of two finite languages L_+ and L_-. Despite the negative results related to the complexity of the problem [1,14], recent work in this field proves that it is possible to relax the uniformly-complete criterion in order to compute the minimal consistent *DFA* with polynomial complexity [31]. Related work in the same field of Grammatical Inference allows to propose a heuristic to compute a *small DFA* that is consistent with a finite input by inferring a team of automata using different order-criteria on the prefix tree acceptor for the sample and selecting the smallest *DFA* obtained [12].

In this paper, we study the conditions that allow the MSA with polynomial complexity to be obtained. We prove a sufficient condition over the whole set of strings that are involved in the problem that guarantees that the process can be carried out with polynomial time complexity. Because the condition we propose takes into account the strings in the union of the sets L_+ and L_-, for the sake of simplicity (and without lack of generality), we state this problem as the following task: given two regular languages L_+ and $L_\cup$, where $L_+\subseteq L_\cup$, compute a minimal *DFA* that accepts the strings in L_+ and rejects the strings in $L_\cup - L_+$ with polynomial time complexity (i.e., the search of a *DFA* that separates L_+ and $L_\cup - L_+$). Any automaton that fulfills these conditions is considered to be *consistent with respect to* L_+ *and* $L_\cup - L_+$.

2. Notation and definitions

In this section, we summarize the main definitions used in the paper. We recommend [16] to the reader for further notions or definitions.

Let Σ be a finite alphabet and let Σ^* be the set of strings over Σ, where λ denotes the empty word and $|x|$ denotes the length of x (thus, $|\lambda|=0$). A *language* L over Σ is any subset of Σ^*. Here we recall the definition of the *canonical order* over Σ^* as being the order that first classifies the shorter strings and considers the alphabetic order for those strings of the same length.

A *(non-deterministic) finite automaton* is a 5-tuple $A=(Q,\Sigma,\delta,I,F)$, where Q is a finite set of states, Σ is an alphabet, $I\subseteq Q$ is the set of initial states, $F\subseteq Q$ is the set of final states and δ: $Q\times\Sigma\rightarrow 2^Q$ is the transition function, which can also be seen as a subset of $Q\times\Sigma\times Q$. The transition function can be extended in a natural way to Σ^* as well as to 2^Q. Given a finite automaton A, we say it is accessible if, for each $q \in Q$, there exists a string x such that $q \in \delta(p, x)$ for some $p \in I$. The *right language* of a state q of a finite automaton A is defined as $L_q^A = \{x \in \Sigma^* \ : \ \delta(q, x) \cap F \neq \emptyset\}$. The language accepted by the finite automaton, which we will denote as $L(A)$, is the union of the right languages of the initial states.

An automaton is called *deterministic* (*DFA*) if, for every state q and every symbol a, the number of transitions is at most one, and where q_0 is the only initial state. Because of the restriction on the set of initial states, a *DFA* is usually denoted as $A = (Q, \Sigma, \delta, q_0, F)$. A *DFA* is said to be complete when the transition function is always defined.

Given a language L and a finite automaton $A = (Q, \Sigma, \delta, I, F)$ such that $L = L(A)$, the reverse automaton for A is defined as the automaton $R(A) = (Q, \Sigma, \delta^R, F, I)$, where $q \in \delta^R(p, a)$ if and only if $p \in \delta(q, a)$. Given any language L, we will denote the reverse language as L^r.

For any finite automaton $A = (Q, \Sigma, \delta, I, F)$, it is possible to obtain an equivalent *DFA* A' using the well-known subset construction, which outputs the automaton $A' = (2^Q, \Sigma, \delta', I, F')$, where $F' = \{P \in 2^Q \ : \ P \cap F \neq \emptyset\}$ and $\delta'(P, a) = \cup_{p \in P} \delta(p, a)$. Let us denote the accessible version of A' by $D(A)$. For the sake of clarity, we will reduce the parentheses to denote the composition of determinization and reverse operations; for instance, we will use $DR(A)$ instead of $D(R(A))$.

Given any language L over an alphabet Σ, we denote the quotient of L by the string u as the language $u^{-1}L = \{v \in \Sigma^* \ : uv \in L\}$. We stress that, given any state $q \in Q$ of a *DFA* and any string $u \in \Sigma^*$ such that $\delta(q_0, u) = q$, the right language L_q^A equals $u^{-1}L(A)$. We also recall that any *DFA* $A = (Q, \Sigma, \delta, q_0, F)$ defines a right-invariant equivalence relation over Σ^*, where $x \equiv_A y$ if and only if $\delta(q_0, x) = \delta(q_0, y)$.

A *partition* π of a set Q is a set $\{P_1, P_2, \ldots, P_k\}$ of pairwise disjoint non-empty subsets of Q such that the union of all the P_i equals Q. We will refer to the subsets of a partition as *blocks*, and we will denote the block of π which contains p with B_p^π. A partition π_1 is refined by π_2 (π_1 is coarser than π_2) if each class in π_2 is contained in some class in π_1.

A Moore machine is a 6-tuple $M = (Q, \Sigma, \Delta, \delta, q_0, \Phi)$, where Σ (resp. Δ) is the input (resp. output) alphabet, δ is a partial function that maps $Q \times \Sigma$ in Q, and Φ is a function that maps Q in Δ called *output function*. The behavior of M is given by the partial function t_M: $\Sigma^* \to \Delta$ defined as $t_M(x) = \Phi(\delta(q_0, x))$, for every $x \in \Sigma^*$ such that $\delta(q_0, x)$ is defined.

In the following, it will be useful to simulate any given *DFA* using a Moore machine. In order to do this, given any *DFA* $A = (Q, \Sigma, \delta, q_0, F)$, it is possible to construct the machine $M = (Q, \Sigma, \{0, 1\}, \delta, q_0, \Phi)$, where $\Phi(q) = 1$ if $q \in F$ and $\Phi(q) = 0$ otherwise. Thus, the language defined by M is $L(M) = \{x \in \Sigma^* : \Phi(\delta(q_0, x)) = 1\} = L(A)$.

In order to propose our method which outputs a minimal *DFA* that separates two regular languages L_+ and L_-, we consider Moore machines whose output alphabet is {0, 1, ?}. Thus, the analysis of words in L_+ and in L_- will return 1 and 0 respectively, and where the analysis of words that are not in $L_+ \cup L_-$ will have *undefined* output (represented by the symbol ?). Therefore, we say that a Moore machine $M = (Q, \Sigma, \{0, 1, ?\}, \delta, q_0, \Phi)$ is *consistent* with respect to L_+ and L_- if, for every string x in L_+ we have that $t_M(x) = 1$, and for every string x in L_- we have $t_M(x) = 0$. Note that a consistent machine is allowed to return *defined* output (values of either 0 or 1) for some strings not in $L_+ \cup L_-$.

3. Similarity relationships

In our work, we consider binary relations that are weaker than equivalence relations. Definition 1 describes the properties of these relations. Fact 2 states a consequence of the definition that will be of importance in what follows. These concepts are based on the notion of *preorder*, which is defined as any reflexive and transitive relation over the domain.

Definition 1. Let $\leq$ be a total preorder on Σ^*. We call a relation $\sim$over Σ^* a *similarity relation*, if the following properties hold:

1. $x \sim x$ for every $x \in \Sigma^*$
2. If $x \sim y$, then $y \sim x$ for every $x, y \in \Sigma^*$
3. Given any $x, y, z \in \Sigma^*$ such that $x \leq y \leq z$, the relation $\sim$holds that:
   - If $x \sim y$ and $x \sim z$, then $y \sim z$.
   - If $x \sim y$ and $y \sim z$, then $x \sim z$.

Fact 2. Any equivalence relation is a similarity relation.

Previous studies that tackle the computation of the minimum cover automaton for finite languages [6,7,18] use a *similarity relation* as a relation that generalizes Nerode's equivalence relation for any regular language L ($x \equiv_L y$ if, for every $z \in \Sigma^*$, $xz \in L$ if and only if $yz \in L$). The similarity relation that is considered in these works takes into account the preorder that is established by considering the length of the strings (i.e., for any two strings, $x \leq y$ whenever $|x| \leq |y|$). Thus, given a finite language L and the maximum length of the strings in L denoted by l, the similarity relation in [6,7,18] states that $x \sim y$ when, for any $z \in \Sigma^*$ such that $|xz|$ and $|yz|$ are lower than or equal to l, it holds that $xz \in L$ if and only if $yz \in L$.

Before defining an extension of the similarity relation used in [6,7,18], which is key in the remainder of our paper, we define the preorder that will be taken into account.

Definition 3. Given any language $L \subseteq \Sigma^*$, we define the *preorder induced* by L in Σ^* as $x \leq y$ if and only if $x^{-1}L \supseteq y^{-1}L$.

In other words, a string x comes before y in the $\leq$ preorder when all the words that can be concatenated to y to obtain words in L can also be concatenated to x and also obtain words in L.

Note that, regardless of the language L, Definition 3 establishes a preorder in Σ^*. We want the order induced by a language to be total. Definition 4 provides a sufficient condition that assures this.

Definition 4. Given any regular language $L\subseteq\Sigma^*$, we say it is *well-structured* if and only if, for every strings x and y over Σ, either $x^{-1}L \subseteq y^{-1}L$ or $y^{-1}L \subseteq x^{-1}L$.

Obviously the preorder induced by a well-structured regular language in Σ^* is total.

Now, in Definition 5, we extend the relation used in previous papers on the computation of the minimal cover-automaton. The modification of the relation is twofold: first, we substitute L_+ (a finite set of *positive* strings) with any regular language; and second, we substitute the restriction to strings with a length lower than or equal to a positive integer with a membership criterion to a (well-structured) regular language.

Definition 5. Let $L_\cup$ be a well-structured regular language over Σ and let L_+ be a regular language such that $L_+ \subseteq L_\cup$. Let $\leq$ be the preorder induced by $L_\cup$ in Σ^*.

For any two strings x and y over Σ, we say that the strings are related with respect to L_+ and $L_\cup$ (denoted with $x \sim_{L_+ L_\cup} y$) if and only if, for every string $w \in \Sigma^*$ such that both xw and yw are in $L_\cup$, then $xw \in L_+$ if and only if $yw \in L_+$ (there is no string w such that xw and yw are conflicting in L_+).

According to the definition, it is clear that $\sim_{L_+,L_\cup}$ is reflexive and symmetric. In order to prove that it is a similarity relation, we prove that it is semi-transitive. Given two strings $x \leq y$, according to the $\leq$ preorder definition, if $x \sim_{L_+,L_\cup} y$, then there is no string w in $x^{-1}L_\cup \cap y^{-1}L_\cup = y^{-1}L_\cup$ such that xw and yw are conflicting in L_+.

Let us consider any three strings x, y and z such that $x \leq y \leq z$. Note that the definition of the $\leq$ preorder, and the fact that $L_\cup$ is well-structured imply that the quotients are such that $z^{-1}L_\cup \subseteq y^{-1}L_\cup \subseteq x^{-1}L_\cup$. We will look for a contradiction in order to show that if $x \sim_{L_+,L_\cup} y$ and $x \sim_{L_+,L_\cup} z$ then $y \sim_{L_+,L_\cup} z$. Let us assume that there exists v in $z^{-1}L_\cup$ such that yv is conflicting with zv in L_+. On the one hand, if xv is not conflicting with yv ($x \sim_{L_+,L_\cup} y$), it implies that xv is conflicting with zv (which is a contradiction). On the other hand, if xv is not conflicting with zv ($x \sim_{L_+,L_\cup} z$), it implies that xv is conflicting with yv (which is again a contradiction).

Similarly it is possible to show that if $x \sim_{L_+,L_\cup} y$ and $y \sim_{L_+,L_\cup} z$ then $x \sim_{L_+,L_\cup} z$. Therefore, the relation $\sim_{L_+,L_\cup}$ satisfies the conditions in Definition 1 and it is a similarity relation. In the following, we use $x\sim y$ instead of $x \sim_{L_+,L_\cup} y$ if no confusion is possible.

The relation stated in [6,7,18] (with respect to the definition of the minimal cover *DFA*) is an instance of the relation that we define because L_+ and $L_\cup$ generalize the finite set of strings, and the set of strings whose length is lower than a given value, respectively.

Definition 6. Let $L_\cup$ be a well-structured regular language over Σ and let L_+ be a regular language such that $L_+ \subseteq L_\cup$. Also let $\sim$be the similarity relation with respect to L_+ and $L_\cup$.

1. A non-empty set of strings is a *similarity set* if it contains only strings that are similar to each other.
2. A *similarity covering* of Σ^* according to $\sim$is a set of similarity sets whose union is Σ^*
3. A *canonical-covering* of Σ^* according to $\sim$is a similarity covering such that the union of any pair of similarity sets is not a similarity set.
4. A *partition* of Σ^* induced by $\sim$is a partition where each block of the partition is a similarity set.
5. A *canonical-partition* induced by $\sim$is a partition where there exists no pair of blocks of the partition such that their union is a similarity set.

Proposition 7 proves that the number of blocks of a canonical-covering is the minimum of any covering according to $\sim$.

Proposition 7. *Let $L_\cup$ be a well-structured regular language over Σ and let L_+ be a regular sublanguage of $L_\cup$. Let $\sim$be the similarity relation with respect to L_+ and $L_\cup$. A covering of Σ^* according to $\sim$is canonical if and only if it has the minimum number of similarity sets.*

Proof. First, given a covering of Σ^*, if it has the minimum number of similarity sets, then it is not possible to unite two similarity sets and obtain another similarity set (this means that the covering is not minimal). Therefore, any minimal covering of Σ^* (i.e., a covering with the minimum number of sets) fulfills the condition of being canonical.

Second, we prove that any canonical-covering is also minimal. Here, for any given covering of Σ^* and any block $B_i = \{u_{i1}, u_{i2}, \ldots\}$, we consider the *first element* of the i-th block as the string u_{i1}, assuming that it is such that $u_{i1} \leq u_{ik}$ for any k. We recall that the $\leq$ preorder is total in this case because $L_\cup$ is well-structured.

Let us consider any canonical-covering of Σ^* according to $\sim$and let u_{i1} and u_{j1} be the first elements of the blocks B_i and B_j of the covering. We assume that $u_{i1} \leq u_{j1}$ without lack of generality. Because the covering is canonical, there exist u_{ik} and u_{jm} such that $u_{ik} \not\sim u_{jm}$. Also $u_{i1} \leq u_{ik}$ and $u_{j1} \leq u_{jm}$.

We consider that $u_{i1} \sim u_{j1}$, and we look for a contradiction. If this is assumed, then $u_{i1} \sim u_{jm}$ because $u_{j1} \sim u_{jm}$ and $u_{i1} \leq u_{j1} \leq u_{jm}$. Furthermore (no matter if $u_{ik} \leq u_{jm}$ or vice versa), $u_{i1} \sim u_{ik}$ and $u_{i1} \sim u_{jm}$ imply that $u_{ik} \sim u_{jm}$. Therefore, the covering is not canonical, which is a contradiction, and thus $u_{i1} \not\sim u_{j1}$.

Finally, since the first elements of each block in a canonical-covering are not similar to each other, the covering is also minimal. □

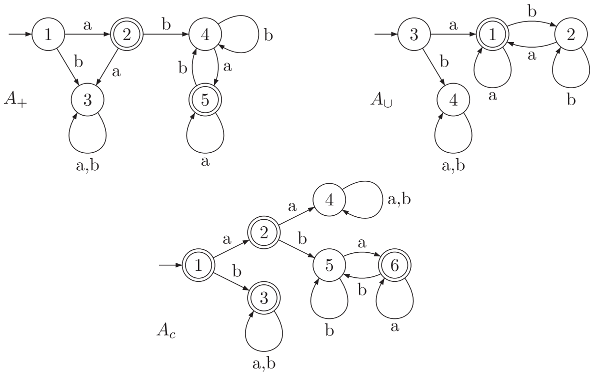


Fig. 1. Two *DFA* examples and an automaton A_c that is consistent with respect to $L(A_+)$ and $L(A_\cup) - L(A_+)$.

We mention here that, given any covering of Σ^* according to a similarity relation, it is trivial to obtain a canonical-covering by substituting any pair of similar sets in the covering by their union whenever it is possible. Similarly, given a covering, it is trivial to obtain a partition induced by that covering by deleting any element that is present in at least two blocks from all but one of the blocks in which the element is contained. Corollary 8 follows directly from these facts and Proposition 7.

Corollary 8. *Let $L_\cup$ be a well-structured regular language over Σ and let L_+ be a regular language such that $L_+ \subseteq L_\cup$. Also let $\sim$be the similarity relation with respect to L_+ and $L_\cup$.*

A canonical-partition induced by $\sim$has the minimum number of blocks of any partition induced by $\sim$.

4. Minimum consistent *DFA*

In this section, we study the relationship between the equivalence relation defined by a *DFA* A and the similarity relation with respect to $L(A)$ and some L such that $L(A)\subseteq L$.

We consider any well-structured regular language $L_\cup$ and any regular language L_+. Proposition 9 allows us to conclude in Corollary 10 that any *DFA* that is consistent with respect to L_+ and $L_\cup - L_+$ defines an equivalence relation $\equiv_A$ (described in Section 2) that refines a canonical-partition induced by the similarity relation with respect to L_+ and $L_\cup$. Example 11 exemplifies this result. Corollary 12 relates the minimum number of states in any consistent *DFA* with the number of similarity sets in a canonical-covering.

Proposition 9. *Let $L_\cup$ be a well-structured regular language over Σ and L_+ be a regular language such that $L_+ \subseteq L_\cup$. Let also $\sim$be the similarity relation with respect to L_+ and $L_\cup$.*

Let A be an automaton consistent with respect to L_+ and $L_\cup - L_+$. The equivalence relation $\equiv_A$ refines the similarity relation $\sim$.

Proof. Given any two strings x and y such that $x \equiv_A y$, it holds that $\delta(q_0, x) = \delta(q_0, y)$, and, for any $z \in \Sigma^*$, $\delta(q_0, xz) = \delta(q_0, yz)$.

Recall that A is consistent with respect to L_+ and $L_\cup - L_+$. Therefore, when both xz and yz are in $L_\cup$, two situations arise: on the one hand, if $\delta(q_0, xz) = \delta(q_0, yz) \in F$, then xz and yz are in L_+; on the other hand, if $\delta(q_0, xz) = \delta(q_0, yz) \notin F$, then both xz and yz are in $L_\cup - L_+$. Therefore, if xz and yz are both in $L_\cup$, then $xz \in L_+$ if and only if $yz \in L_+$, and $x \sim y$. □

The result of Proposition 9 also follows from the fact that, given any consistent *DFA* A with respect to L_+ and $L_\cup - L_+$, the equivalence relation $\equiv_A$ defines a consistent partition (and therefore a covering) of Σ^* that is also consistent with respect to the similarity relation $\sim$.

Corollary 10. *Let $L_\cup$ be a well-structured regular language over Σ and let L_+ be a regular language such that $L_+ \subseteq L_\cup$. Let $\sim$be the similarity relation with respect to L_+ and $L_\cup$.*

Let A be an automaton that is consistent with respect to L_+ and $L_\cup - L_+$. The equivalence relation $\equiv_A$ refines a canonical-partition induced by the similarity relation $\sim$.

Example 11. Let us consider the automata in Fig. 1. The automaton on the right accepts the language of strings over $\{a, b\}$ that begin and end with a. The automaton on the left accepts the language of strings that begin and end with the symbol a but do not begin with aa.

Note that A_c is consistent with respect to $L(A_+)$ and $L(A_\cup) - L(A_+)$. Let $\sim$be the similarity relation with respect to $L(A_+)$ and $L(A_\cup)$. In this example, we denote the set of strings equivalent to x according to the equivalence relation $\equiv_A$ as $[x]_A$.

To exemplify that $\equiv_{A_c}$ refines $\sim$, we show that the strings equivalent in $[ab]_{A_c}$ (strings that begin with ab and end with the symbol b) are similar to those in $[aba]_{A_c}$ (strings that begin with ab and end with the symbol a). To do so, and according

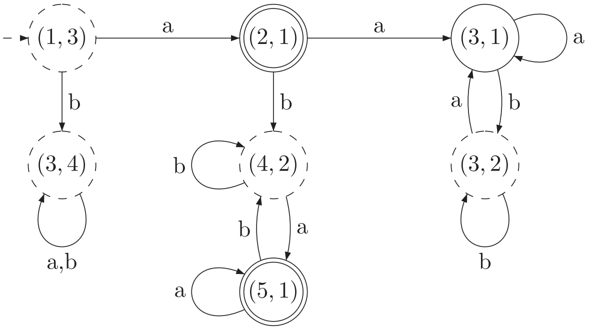


Fig. 2. The Moore machine obtained from the automata shown in Fig. 1. Single-circled states have output 0, double-circled states have output 1, and dashed states have output ?.

to the definition, given any pair of strings $x \in [ab]_{A_c}$ and $y \in [aba]_{A_c}$, the set of strings $Z = \{z \in \{a, b\}^* \,:\, xz \in L(A_\cup) \wedge yz \in L(A_\cup)\}$ is taken into account.

Note that any string in $[ab]_{A_c}$ (resp. in $[aba]_{A_c}$) reaches state 2 in the *DFA* $A_\cup$ (resp. state 1 in the *DFA* $A_\cup$). Therefore, the set Z is $L_1^{A_\cup} \cap L_2^{A_\cup}$, which contains the set of words that end with the symbol a.

The strings in $[ab]_{A_c}$ reach the state 4 of A_+, and the strings in $[aba]_{A_c}$ reach the state 5 of A_+. Since only the strings in Z (strings that end with symbol a) are considered, the strings are all analyzed in the same way from the states 1 and 2 in A_+ (all of them are accepted). Therefore, the strings in $[ab]_{A_c}$ are similar to those in $[aba]_{A_c}$.

Also, note that $[b]_{A_c} \cup [ab]_{A_c} \cup [aba]_{A_c}$ is also a similarity set. In this case, since the strings in $[b]_{A_c}$ reach state 4 in $A_\cup$, the set $Z = L_1^{A_\cup} \cap L_2^{A_\cup} \cap L_4^{A_\cup}$ is empty, and the similarity conclusion is reached by vacuity.

Corollary 12. *Let $L_\cup$ be a well-structured regular language over Σ and let L_+ be a regular sublanguage of $L_\cup$. Let A be a DFA that is consistent with respect to L_+ and $L_\cup - L_+$.*

If A is minimal (it has the minimum number of states among the consistent DFA s), then it has as many states as similarity sets in a canonical-covering induced by the similarity relation $\sim$.

5. Computation of the relation ~

In this section, we propose a method to compute the similarity relation. Once the $\sim$relation is computed, it is possible to obtain every canonical-partition that is induced by the similarity relation.

Taking into account a well-structured language $L_\cup$ and a sublanguage L_+ of $L_\cup$, the method considers *DFA* s for both L_+ and $L_\cup$ in order to obtain a finite state machine (a Moore machine) that is consistent with respect to L_+ and $L_\cup - L_+$. From now on, we consider the automaton for L_+ to be complete.

The Moore machine that the method constructs defines an equivalence relation over its states that allows a relation over Σ^* to be defined. We prove that this relation equals the similarity relation that is induced by L_+ and $L_\cup$.

Definition 13. Let L_+ and $L_\cup$ be two regular languages over Σ and let $A_+ = (P, \Sigma, \delta_+, p_0, F_+)$ and $A_\cup = (Q, \Sigma, \delta_\cup, q_0, F_\cup)$ be two *DFA* s such that $L(A_+) = L_+$ and $L(A_\cup) = L_\cup$. We define the Moore machine $M = (P \times Q, \Sigma, \{0, 1, ?\}, \delta, (p_0, q_0), \Phi)$, where $\delta((p, q), a) = (\delta_+(p, a), \delta_\cup(q, a))$ (when both are defined), and the output function is defined as follows:

$$\Phi((p,q)) = \begin{cases} 1 & \text{if } p \in F_+ \text{ and } q \in F_\cup, \\ 0 & \text{if } p \notin F_+ \text{ and } q \in F_\cup, \\ ? & \text{if } q \notin F_\cup \end{cases}$$

Example 14 depicts the construction of a Moore machine according to Definition 13.

Example 14. Consider the automata in Fig. 1. Note that $L(A_\cup)$ is well-structured and that the labeling of the states denotes their order according to the preorder induced by $L(A_\cup)$ in $\{a, b\}^*$. For instance, $L_1^{A_\cup} = (a+b)^*a + \lambda$ and $L_2^{A_\cup} = (a+b)^*a$, and, therefore, $L_1^{A_\cup} \supseteq L_2^{A_\cup}$. For the sake of brevity, we abuse the notation and say that $1 \le 2$.

The Moore machine obtained from A_+ and $A_\cup$ is shown in Fig. 2.

We define the relation $\equiv_M$ (which we will refer to as the *indistinguishability relation*) over $P \times Q$, where $(p_1, q_1) \equiv_M (p_2, q_2)$ if, for every string z over Σ, whenever both $\Phi(\delta((p_1, q_1), z))$ and $\Phi(\delta((p_2, q_2), z))$ are in $\{0, 1\}$ (defined), then both are equal.

The indistinguishability relation $\equiv_M$ allows a relation over Σ^* to be defined where, for any two strings, $x \sim_M y$ if and only if $\delta((p_0, q_0), x) \equiv_M \delta((p_0, q_0), y)$. Proposition 15 proves that the relation $\sim_M$ coincides with the similarity relation induced by L_+ and $L_\cup$.

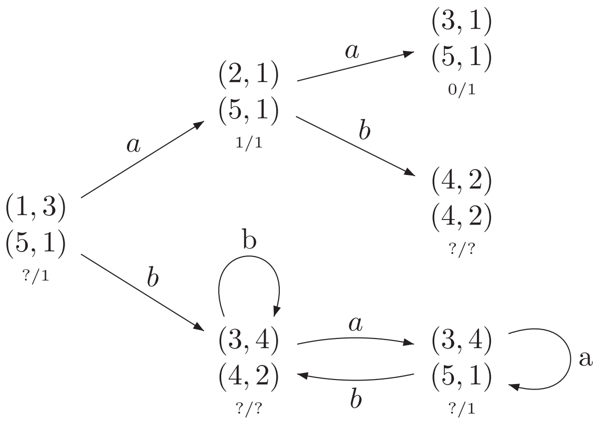


Fig. 3. Analysis of the relation (1, 3) $\equiv_M$(5, 1). The small symbols represent the output of the states in the pair.

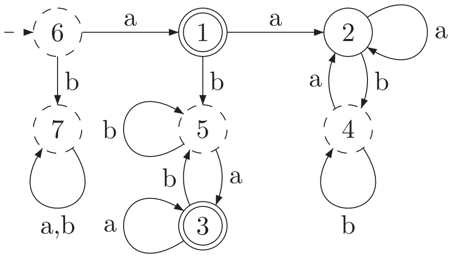


Fig. 4. The Moore machine obtained from the automata shown in Fig. 1. The machine is isomorphic to the one in Fig. 2, where the numbering of the states follows the order induced by $L(A_\cup)$ in Σ^*.

Proposition 15. *Let $L_\cup$ be a well-structured regular language over Σ, and let L_+ be a regular language such that $L_+ \subseteq L_\cup$. Let $A_+ = (P, \Sigma, \delta_+, p_0, F_+)$ and $A_\cup = (Q, \Sigma, \delta_\cup, q_0, F_\cup)$ be such that they accept L_+ and $L_\cup$, respectively. Let M be the Moore machine obtained from A_+ and $A_\cup$ according to Definition 13.*

For any two strings x and y over Σ, the relation defined as $x \sim_M y$ if and only if $\delta((p_0, q_0), x) \equiv_M \delta((p_0, q_0), y)$ coincides with the similarity relation $\sim$induced by L_+ and $L_\cup$.

Proof. We first prove that, if $x \sim_M y$ and $z \in \Sigma^*$ is a string such that both xz and yz are in $L_\cup$, then $xz \in L_+$ if and only if $yz \in L_+$ $(x{\sim}y)$.

According to the definition, $x \sim_M y$ if and only if $\delta((p_0, q_0), x) \equiv_M \delta((p_0, q_0), y)$, or in other words, if and only if, for every $z \in \Sigma^*$, when both $\Phi(\delta((p_0, q_0), xz))$ and $\Phi(\delta((p_0, q_0), yz))$ are defined (either 0 or 1), they are equal.

Let us suppose that both xz and yz are in $L_\cup$. Then both $\delta((p_0, q_0), x)$ and $\delta((p_0, q_0), y)$ are in $P \times F_\cup$. Note also that both are either in $F_+ \times F_\cup$ or in $(P - F_+) \times F_\cup$ because $x \sim_M y$. Therefore $xz \in L_+$ if and only if $yz \in L_+$, and thus $x{\sim}y$.

Second, we prove that if $x{\sim}y$, then $x \sim_M y$. In this case, for every $z \in \Sigma^*$ such that both xz and yz are in $L_\cup$, then $xz \in L_+$ if and only if $yz \in L_+$. Let us consider that $\delta((p_0, q_0), xz) = (p, q)$ and $\delta((p_0, q_0), yz) = (p', q')$.

First, if both xz and yz are in $L_\cup$, then q and q' are both in $F_\cup$. Second, $x{\sim}y$, and therefore p and p' are either both or none in F_+. In other words, $\Phi((p, q))$ and $\Phi((p', q'))$ are in {0, 1} and are equal. This implies that $\delta((p_0, q_0), x) \equiv_M \delta((p_0, q_0), y)$ and that $x \sim_M y$. □

In Example 16, we consider the automata in Fig. 1 and the corresponding Moore machine shown in Example 14 (Fig. 2). We also depict a preliminary method to establish the similarity between strings, which can be seen as a variation of the Huffmann-Moore minimization method described in [16].

Example 16. Consider the Moore machine in Fig. 2. In order to state if two states p and q are not related according to $\equiv_M$, it is sufficient to find a word z such that $\Phi(\delta(p, z))$ and $\Phi(\delta(q, z))$ are defined but distinct. This process can be carried out by traversing the Moore machine starting from the states to be checked, looking for states with this defined and distinct output. Fig. 3 summarizes this process for the pair of states (1, 3) and (5, 1).

As shown in Fig. 3, the process detects that $(3, 1) \not\equiv_M (5, 1)$, and that (3, 4) $\equiv_M$(4, 2) as well. Also, since (3, 4) $\equiv_M$(4, 2), every pair of strings x and y such that $\delta((1, 3), x) = (3, 4)$ and $\delta((1, 3), y) = (4, 2)$ fulfills that $x \sim_M y$, and, therefore, $x{\sim}y$.

We have proved the relationship between $\equiv_M$ and $\sim$in Proposition 15 and depicted it in Example 16. This relationship allows us to extend some definitions on strings over the alphabet in order to consider states of the Moore machine.

Table 1
The relationship of states related according to the relation $\equiv_M$.

	2	3	4	5	6	7
1	×	×	×	×	×	✓
2		×	✓	×	×	✓
3			×	✓	×	✓
4				×	×	✓
5					×	✓
6						✓

Thus, the notions of canonical-covering of Σ^* by $\sim$and the canonical-partition induced by $\sim$can be extended in order to take into account the relation $\equiv_M$ and the set of states of the Moore machine M. Thus, given a Moore machine M and the relation $\equiv_M$, we say that C (resp. π) is a *canonical-covering* (resp. *canonical-partition*) of the set of states of M if the union of any pair of blocks of the covering (resp. partition) implies that the result contains at least two non-equivalent states according to $\sim_M$.

We also extend the $\leq$ preorder to consider states of the defined Moore machine M. Thus, given any pair of states p and q, we say that $p \leq q$ if, given any string x over the alphabet, whenever $\Phi(\delta_M(q, x))$ is defined, then $\Phi(\delta_M(p, x))$ is also defined. This allows us to define the *first state* of a set $B = \{q_1, q_2, \dots, q_m\}$ as the state q_i such that $q_i \leq q_j$ for any $i \neq j$.

Example 17 shows the process to obtain a canonical-covering of the set of states as well as any canonical partition of the set of states.

Example 17. In this example, we again consider the automata in Fig. 1 and the Moore machine in Fig. 2. For the sake of brevity, we rename the states of M taking into account the order induced by $L(A_\cup)$ in Σ^*

Table 1 shows the $\equiv_M$-relations among all the states. Taking into account the information in the table, a canonical-covering of the set of states can be obtained by traversing the set of states according to the preorder $\leq$. This assures that the states with a greater amount of information are considered first. Thus, considering an initially empty set, whenever the state being analyzed is not already in the set, its set of relationships is included in the set. According to this procedure, Table 1 can be summarized with the following set:

$$\{\{1, 7\}, \{2, 4, 7\}, \{3, 5, 7\}, \{6, 7\}\}.$$

This set summarizes the relationships of every state. The set is a covering of the set of states of the Moore machine. Obviously, it is a canonical-covering of the set of states of the Moore machine, and, therefore, it has the minimum number of similarity sets. This canonical-covering has the advantage of gathering all of the relationships that have the minimum number of similarity sets (this is the *coarsest* canonical-covering).

This canonical-covering allows us to obtain every canonical-partition of the set of states taking into account the different ways of distributing the states that are present in more than one element. In this example, the possible canonical-partitions are:

$$\{\{1, 7\}, \{2, 4\}, \{3, 5\}, \{6\}\}$$
$$\{\{1\}, \{2, 4, 7\}, \{3, 5\}, \{6\}\}$$
$$\{\{1\}, \{2, 4\}, \{3, 5, 7\}, \{6\}\}$$
$$\{\{1\}, \{2, 4\}, \{3, 5\}, \{6, 7\}\}$$

We now propose a method to construct a *DFA* taking into account a partition induced by a similarity relation. We consider Proposition 15, which proves that the similarity relation $\sim$among strings equals the indistinguishability relation $\sim_M$ over the states of the Moore machine that we define.

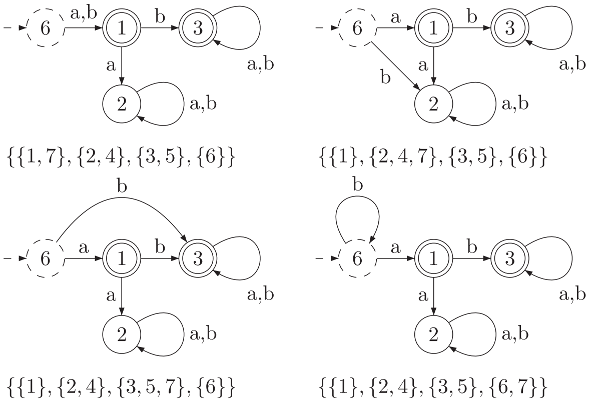
The process takes into account a partition π of the set of states of the Moore machine induced by the indistinguishability relation. In order to construct a consistent *DFA*, it is essential to select the representant of each block of the partition. For each block $B_i^\pi = \{q_{i1}, q_{i2}, \dots, q_{ik}\}$, let the *first element in the block* q_{i1} be such that $q_{i1} \leq q_{ik}$ for any k. Proposition 18 describes the construction of a consistent *DFA* and proves that the construction is consistent with respect to L_+ and $L_\cup - L_+$.

Proposition 18. *Let $L_\cup$ be a well-structured regular language over Σ and let L_+ be a regular language such that $L_+ \subseteq L_\cup$. Let $\sim_M$ be the indistinguishability relation defined on the Moore machine $M = (Q^M, \Sigma, \{0, 1, ?\}, \delta^M, q_0^M, \Phi)$ that is obtained using DFA s for the languages L_+ and $L_\cup$ according to Definition 13.*

Also let π be a partition of the states of the Moore machine induced by $\sim_M$.

The DFA $A_\pi = (Q, \Sigma, \delta, q_0, F)$ where: Q is the set of the first elements of π; the initial state of the automaton $q_0 = q_{j1}$ is the first element of the partition block π that q_0^M belongs to; the set of final states is $F = \{q \in Q \ : \ \Phi(q) = 1\}$; and, for any $q \in Q$ and $a \in \Sigma$, $\delta(q, a)$ is the first element of $\delta^M(q, a)$ according to π.

This DFA is consistent with respect to L_+ and $L_\cup - L_+$.



$\{\{1,7\},\{2,4\},\{3,5\},\{6\}\}$ $\{\{1\},\{2,4,7\},\{3,5\},\{6\}\}$

$\{\{1\},\{2,4\},\{3,5,7\},\{6\}\}$ $\{\{1\},\{2,4\},\{3,5\},\{6,7\}\}$

Fig. 5. Minimal consistent *DFA* s for the input languages. The dashed state can be either final or non-final.

Proof. The definition of the *DFA* is similar to the one obtained from an equivalence relation. We first prove that the transition function is consistent.

Let q_1 and q_i be the first element of a block of the partition and any other state in the block respectively. Since $q_1 \sim_M q_i$, for any $z \in \Sigma^*$ such that $\Phi(\delta(q_1, z))$ is defined, $\Phi(\delta(q_i, z))$ either equals $\Phi(\delta(q_1, z))$ or is undefined, and therefore the transition function is consistent.

According to the constructions, it is possible to obtain states with undefined output. If such states are present, they can either be included or not included in the set of final states. In either case, the automaton obtained is consistent. □

These results allow us to prove that any canonical-partition induced by the similarity relation that we propose defines a minimal consistent *DFA* .

Proposition 19. *Let $L_\cup$ be a well-structured regular language over Σ and let L_+ be a regular language such that $L_+ \subseteq L_\cup$. Let $\sim$be the similarity relation with respect to L_+ and $L_\cup$.*

A canonical-partition induced by $\sim$defines a minimal consistent DFA with respect to L_+ and $L_\cup - L_+$.

Proof. Note that Proposition 18 proves the consistency of an automaton obtained from any partition (canonical or not) induced by $\sim$. Corollary 10 proves that the equivalence induced by a *DFA* that is consistent with respect to L_+ and $L_\cup - L_+$ refines a canonical-partition induced by $\sim$. The proof follows from these results and the fact that any canonical-partition induced by $\sim$has the minimum number of classes (i.e., no blocks can be united in order to obtain a similarity set). □

As we proved in Proposition 18, an automaton can be constructed from any partition induced by a similarity relation. Example 20 depicts this construction taking into account the partitions that have the minimum number of blocks.

Example 20. Let us consider the canonical-partitions that are obtained at the end of Example 17. The automata obtained by considering the partitions shown are depicted in Fig. 5. Note that we have not taken into account the undefined output of state 6. As noted in Proposition 18, this state can be considered either final or non-final without affecting the consistency of the automaton. Therefore, there exist eight minimal automata that are consistent with respect to $L(A_+)$ and $L(A_\cup) - L(A_+)$.

6. An efficient algorithm for computing a minimal consistent *DFA*

Taking into account any regular language L_+ that is included in a well-structured regular language $L_\cup$, Section 5 proves that it is possible to reduce the computation of $\sim$to a relation over the states of a machine that is consistent with respect to L_+ and $L_\cup - L_+$. In this section, we propose an efficient algorithm to carry out this computation. At the end of the section, we prove the correctness and the complexity of the algorithm that we propose.

Our method takes into account the Moore machine that is obtained from the automata that accept the input languages. The states of the Moore machine are renumbered in order to consider the preorder described in Definition 3. This ordering is used to traverse the states and select those states for which no equivalent state has already been detected. Once all of the states have been traversed, the algorithm considers the selected states for computing the transition function, selecting the initial state, and distinguishing the accepting states. The algorithm is summarized in Algorithm 6.1 and its behavior is depicted in Example 21.

Example 21. We consider the automata in Fig. 1 and, therefore, the Moore machine shown in Fig. 2.

The states in $A_\cup$ are numbered according to the order described in Definition 3 and the states in A_+ are numbered taking into account the first string in canonical order that reaches each state. Note that the renaming of the states of the Moore machine carried out in Fig. 4 is consistent with the ordering because it considers the second component in each state first. Any other numbering that satisfies this is also valid.

Algorithm 6.1 Computation of a minimal consistent *DFA* .

```
Require: A DFA $A_\cup$ that accepts a well-structured regular language
Require: A complete DFA $A_+$ that accepts a sublanguage of $L(A_\cup)$
Ensure: A minimal consistent DFA with respect to $L(A_+)$ and $L(A_\cup) - L(A_+)$
 1: Method
 2: Obtain $M = (Q, \Sigma, \{0, 1, ?\}, \delta_M, q_0, \Phi)$according to Definition 13
 3: Rename the states in $M$ according to the preorder $\leq$ described in Definition 3
 4: $Eqs = QSet = \emptyset$
 5: for $p \in Q$ do
 6:   $EqFound = False$
 7:   for $p' \in QSet$ do
 8:     if $p' \equiv_M p$ then
 9:       Append $(p, p')$ to $Eqs$
10:       $EqFound = True$
11:       BreakFor
12:     end if
13:   end for
14:   if $EqFound == False$ then Append $p$ to $QSet$ end if
15: end for
16: for $p \in QSet$; $a \in \Sigma$ do
17:   if $\delta_M(p, a) \in QSet$ then Set $\delta(p, a) = \delta_M(p, a)$
      else Set $\delta(p, a) = p'$, where $(p, p') \in Eqs$ end if
18: end for
19: if $\exists(q_0, p')$ in $Eqs$ then $p_0 = p'$
    else $p_0 = q_0$  end if
20: $F = \{q \in QSet : \Phi(q) \neq 0\}$
21: Return $A = (QSet, \Sigma, \delta, p_0, F)$
22: End Method.
```

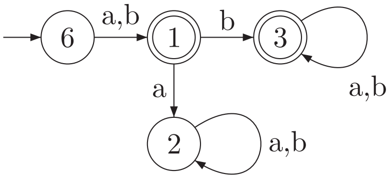


Fig. 6. Output of the proposed algorithm.

The loop in line 5 traverses the states taking into account an (initially) empty *QSet* and therefore state 1 is added to *QSet*. The analysis of state 2 returns that $1 \not\equiv_M 2$ (i.e., they have different outputs and therefore process the λ string in a different way). Hence, state 2 is also added to *QSet*.

The analysis of state 3 returns that $3 \not\equiv_M 1$ (e.g., because of the processing of the string a) and also that $3 \not\equiv_M 2$ (because of they have different output). Therefore, state 3 is also added to *QSet*. Since State 4 is found to be similar to state 2, once the updating has been carried out, $Qset = \{1, 2, 3\}$ and $Eqs = \{(4, 2)\}$.

The processing of state 5 returns that it is similar to state 3. Therefore, $Qset = \{1, 2, 3\}$ and $Eqs = \{(4, 2), (5, 3)\}$. Despite its undefined output, the analysis of state 6 shows that it is distinguishable from every state in *Qset*. Therefore, after updating the variables, $Qset = \{1, 2, 3, 6\}$ and $Eqs = \{(4, 2), (5, 3)\}$.

State 7 is similar to any of the states that are already in *Qset*. Nevertheless, in order to obtain an automaton, it suffices to detect one similarity. The algorithm detects that state 7 is similar to 1; therefore, $Eqs = \{(4, 2), (5, 3), (7, 1)\}$.

The initial state is in *QSet*, the set of accepting states is set to {1, 3}, and the transition function is obtained taking into account only the states in *QSet*. The final output is shown in Fig. 6.

Propositions 22 and 23 prove the correctness and complexity of our the method.

Proposition 22. *Given any pair of DFA s $A_\cup$ and A_+ that accept a well-structured regular language and a sublanguage of $L(A_\cup)$, respectively, the Algorithm 6.1 obtains the minimum consistent DFA with respect to $L(A_+)$ and $L(A_\cup) - L(A_+)$.*

Proof. In order to prove the correctness of the algorithm, the method traverses the set according to the preorder $\leq$ that guarantees that the states with the most information available are considered first. The main loop in line 5 essentially constructs one of the possible canonical-partitions of the set of states according to the relation $\equiv_M$. Once this partition is obtained, the algorithm constructs the output automaton in a straightforward way. Proposition 15 assures that the output is a minimal consistent *DFA*. □

Proposition 23. *Algorithm 6.1 runs with polynomial time complexity.*

Proof. Let n and m be the number of states of A_+ and $A_\cup$, respectively.

The complexity of the algorithm can be reduced to the complexity of the main loop in line 5. This loop traverses the set of states of M whose size is bounded by $\mathcal{O}(nm)$. Each analysis of similarity between states can be carried out with $\mathcal{O}(nm)$ time complexity. The number of similarity analyses carried out depends on the number of states of the minimal consistent *DFA*, but it is bounded by $\mathcal{O}(n^2m^2)$. Therefore, the algorithm runs with polynomial time complexity. □

7. Well-structured languages

As we have proven in the previous sections, it is possible to polynomially compute the MSA of two regular languages L_+ and L_- when the union of both languages is well-structured. In this section, we present some results that describe the class of well-structured languages. First, we prove that there exists a method to decide (with polynomial time complexity) whether a *DFA* identifies a well-structured language. Second, we characterize the automata that identify the class of well-structured languages, and we provide a method to generate any automaton in that class. Third, we prove that the class of well-structured languages is closed under some of the usual operations on languages. Finally we show the relationship between the class of well-structured languages and other well-known subclasses of regular languages.

Some of the results that we present here are based on the properties of the *Universal Automaton* for a language as defined by Lombardy and Sakarovitch in [19]. In order to describe the universal automaton for a language L, the set of quotients of L plays an important role. Therefore, we define the set $D^L = \{u^{-1}L \ : \ u \in \Sigma^*\}$ and also the set $D^L_\cap = \{q_1 \cap \ldots \cap q_k \ : \ k \geq 0, \ q_1, \ldots, q_k \in D^L\}$ (the intersection closure of the set D^L). Whenever the language L is regular, both the sets D^L and $D^L_\cap$ are finite.

In their paper, Lombardy and Sakarovitch define the universal automaton for a language L as the automaton $\mathscr{U}^L = (D^L_\cap, \Sigma, \delta, I, F)$, where $I = \{p \in D^L_\cap \ : \ p \subseteq L\}$, $F = \{p \in D^L_\cap \ : \ \lambda \in p\}$, and the set of transitions is defined as $\delta(p, a) = \{p' \in D^L_\cap \ : \ p' \subseteq a^{-1}p\}$. The method proposed by Lombardy and Sakarovitch for computing the universal automaton considers the minimal *DFA* $A = (Q, \Sigma, \delta, q_0, F)$ for the language, and first computes the automaton $DR(A) = (Q^{DR}, \Sigma, \delta^{DR}, F, F^{DR})$ (according to the subset-construction and the reverse operations explained in Section 2). Using this intermediate result, the universal automaton $\mathscr{U}_L = (\mathcal{U}, \Sigma, \delta_\mathscr{U}, I_\mathscr{U}, F_\mathscr{U})$ is obtained as follows:

- $\mathcal{U} = Q^{DR}_\cap$
- $I_\mathscr{U} = \{X \in \mathcal{U} \ : \ q_0 \in X\}$
- $F_\mathscr{U} = \{X \in \mathcal{U} \ : \ X \subseteq F\}$
- $\delta_\mathscr{U}(X, a) = \{Y \in \mathcal{U} \ : \ \delta(X, a) \subseteq Y \ \wedge \ \forall q \in X, \ \delta(q, a)_\downarrow\}$, where $\delta(q, a)_\downarrow$ means that the transition $\delta(q, a)$ is defined.

In order to propose a decision algorithm to decide if a given language is well-structured we prove the condition on which the algorithm is based in Proposition 24, and we prove that the decision process can be carried out with polynomial time complexity in Proposition 25.

Proposition 24. *Given a language L, if L is well-structured, then the minimal DFA and the universal automaton for L have the same number of states.*

Proof. To prove this proposition, we take into account that any automaton that accepts L has a morphic image that is a subautomaton of the universal language for L [19]. We also recall that the set of states of the minimal *DFA* for L is D^L (the set of different quotients of L).

By definition, a language L is a well-structured language if there exists an inclusion relationship between any pair of quotients. In this case, D^L equals $D^L_\cap$, and, therefore, both automata have the same number of states. □

Proposition 25. *It is possible to decide whether a minimal DFA A accepts a well-structured language with polynomial time complexity with respect to n, the number of states of A.*

Proof. As shown in Proposition 24, the decision procedure is based on the computation of the $DR(A)$ automaton. Note that the computation of $R(A)$ can be carried out with linear time complexity with respect to n. Note also that the name of each state in the $DR(A)$ automaton denotes an intersection of right languages in A.

In order to decide if the input automaton identifies a well-structured language, it is not necessary to compute the $DR(A)$ automaton (whose size is potentially exponential) completely. To carry out the process, it suffices to modify the well-known subset construction procedure.

The modification is twofold: first, every time a new state is found, the procedure checks if the inclusion relationship holds (this can be done with polynomial time with respect to n by analyzing the names of the states that the subset construction method obtains); second, we note that, if the process does not end after n iterations, then the universal automaton for the language identified by the input *DFA* would have more states than n, and, by Proposition 24, the input automaton does not identify a well-structured language. Therefore, we can conclude that the decision process can be carried out with polynomial time complexity with respect to n. □

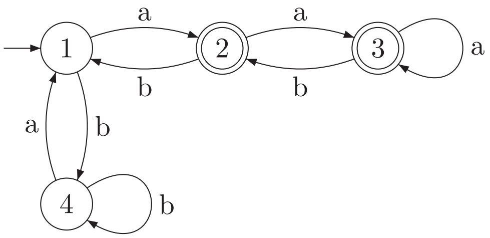


Fig. 7. A *DFA* example.

As mentioned in Proposition 24, when a language is well-structured, the minimal *DFA* and the universal automaton for the language have the same number of states and there exists a one-to-one correspondence between the states of the two automata. Therefore, as a byproduct of the decision process and whenever the input *DFA* is well-structured, it is possible to order the set of states of the *DFA* with respect to the inclusion relation of their right languages. The method described below for ordering the states of the input *DFA* takes into account the computation of the *DR*(*A*) automaton, especially the names of the states that are output by the subset construction procedure. We illustrate the decision procedure and the ordering process in Example 26.

Example 26. Let *A* be the *DFA* in Fig. 7.

The set of states of the automaton *DR*(*A*) is {{3}, {2, 3}, {1, 2, 3}, {1, 2, 3, 4}}. For example, state {2, 3} is related to the intersection of the right languages of states 2 and 3. As can be observed, there exists an inclusion relationship in the elements of the set, and the intersection closure of the set does not produce new states. Therefore, by Proposition 24, the *DFA* is well-structured.

Note that every state of the universal automaton is related to the right language of state 3, and, therefore, state 3 is the first in the order. Taking into account the frequency of the right languages of the input *DFA* in the set of states of the universal automaton, the second state is state 2, the third one is state 1 and the fourth state state 4.

We now present a characterization of the automata that identify any language in the class of well-structured languages. We first extend Definition 4 to automata. This natural extension takes into account the relationship between quotients of the language and the states of the minimal *DFA* for a language. We then prove a characterization of the well-structured automata class in Proposition 28.

Proposition 27. *The minimal DFA $A = (Q, \Sigma, \delta, q_0, F)$ of a well-structured language L satisfies that, for every pair of states p and q in Q, either $L_p^A \subset L_q^A$ or $L_q^A \subset L_p^A$.*

Proof. The proof is straightforward from Definition 4 □

Proposition 28. *A language L is well-structured if and only if its minimal DFA $A = (Q, \Sigma, \delta, q_0, F)$ is such that there is a total order $\leq$ over the states in Q that satisfies the following conditions:*

- *For any pair of states p and q, if $p \in F$ and $q \in Q - F$, then $p \leq q$.*
- *Given any two states such that $p \leq q$, then, for any symbol $a \in \Sigma$, it hold that $\delta(p, a) \leq \delta(q, a)$.*

Proof. First, we prove that if it is possible to establish the order described above on the states of the automaton, then it is well-structured. Let p and q be two states such that $p \leq q$, and let us suppose that $L_q^A \not\subset L_p^A$. Then there exists a string $x \in L_q^A - L_p^A$, and therefore $\delta(q, x) \in F$ but $\delta(p, x) \in Q - F$. This implies that $\delta(p, x) \not\leq \delta(q, x)$, and thus $p \not\leq q$, which contradicts the hypothesis.

Second, let us consider the minimal *DFA* for any given well-structured language and let the order $\leq$ be defined as $p \leq q$ if and only if $L_q^A \subseteq L_p^A$. Let us suppose that $p \in F$ and $q \in Q - F$. The automaton is well structured (there exists an inclusion relation with respect to the right languages of the states) and $\lambda \notin L_q^A$. Therefore, $L_q^A \subset L_p^A$ and $p \leq q$. Let us now suppose that p and q are two states such that $L_q^A \subset L_p^A$. Therefore, for any symbol a in Σ, $L_{\delta(q,a)}^A \subset L_{\delta(p,a)}^A$ and $\delta(p, a) \leq \delta(q, a)$. □

Proposition 28 states the conditions that a *DFA* must fulfill to identify a well-structured language. In the following, any *DFA* that fulfills these conditions is referred to as well-structured. Furthermore, the described conditions can be used to implement a method that is able to generate well-structured automata. The method that we propose to do this is summarized in Algorithm 7.1.

The method described in Algorithm 7.1 does not take into account accessibility conditions and therefore can output a *DFA* whose minimal version would have less than n states. Example 29 illustrates the behavior of the algorithm.

Example 29. Let $n = 4$ and $k = 2$ be the input values of the method. Therefore, $Q = \{1, 2, 3, 4\}$ and $\Sigma = \{1, 2\}$. Also let the set of final states be $F = \{1, 2\}$. Note that the name of each state is related to its order in the *DFA* (e.g., $1 \leq 3$).

Algorithm 7.1 A method to generate well-structured automata.

```
Require: Two integers $n$ (number of states) and $k$ number of symbols of the DFA
Ensure: A DFA $A = (Q, \Sigma, \delta, q_0, F)$ that accepts awell-structured language
Method
$Q = \{1, \dots, n\}$
$\Sigma = \{1, \dots, k\}$
Let $f$ be a random integer in the interval $[1, n-1]$
$F = \{1, \dots, f\}$
for all $s \in \Sigma$ from $s = 1$ to $k$ do
  $limit = 1$
  for all $q \in Q$ from 1 to $n$ do
    Choose a random number $m$ in the interval $[limit, n]$
    $limit = m$
    Add the transition $(q, s, m)$ to $\delta$
  end for
end for
Randomly choose the initial state $q_0$ in the interval $[1, n]$
Return $A = (Q, \Sigma, \delta, q_0, F)$
End Method.
```

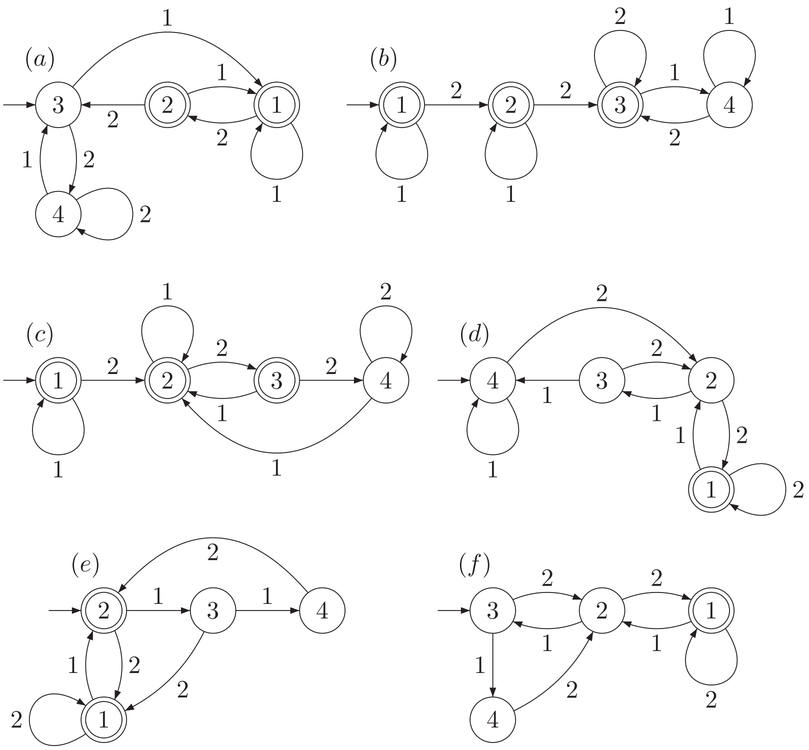


Fig. 8. Some examples of well-structured automata with 4 states and 2 symbols obtained by Algorithm 7.1. The names of the states show their ordering according to the inclusion of their right languages.

The first loop traverses the set of symbols. For each symbol, the second loop traverses the set of states and chooses the destination state of the transition. The *limit* variable establishes the first state that the transitions can be addressed to and it is properly updated.

Some examples of well-structured automata are shown in Fig. 8. The set of final states as well as the initial state can be modified to obtain other automata in the class. Nevertheless, this modification may imply that the automata are not minimal. The automata at the bottom of Fig. 8 are examples of incomplete *DFA* s that identify well-structured languages.

In order to get an idea of how strict the condition of being well-structured is, it should to be noted that the class contains the only two situations where the polynomial computation of the minimum separating *DFA* has been proved (when $L_\cup$ is either Σ^* or a uniformly-complete set). We now present some operations on languages under which the class of well-structured languages is closed. For each one of the closure properties, we provide the essential base to prove it. Later, we show the relationship between the class of well-structured languages and aperiodic, ordered, locally testable, or definite languages among others. This set of results provides an idea of the depth and breadth of the class of well-structured languages.

Property 30. *The class of well-structured languages is closed under the complement operation.*

The proof of Property 30 can be based on the fact that, given any language L, its complement $\overline{L}$, and any two strings x and y over the alphabet, $x^{-1}L \subseteq y^{-1}L$ is satisfied if and only if $y^{-1}\overline{L} \subseteq x^{-1}\overline{L}$.

Property 31. *The class of well-structured languages is closed under the inverse homomorphism operation.*

Property 31 can be proved taking into account that, given any well-structured language L and a homomorphism h, for any two strings x and y over the alphabet, if $h(x)^{-1}L \subseteq h(y)^{-1}L$, then $x^{-1}h^{-1}(L) \subseteq y^{-1}h^{-1}(L)$.

Property 32. *The class of well-structured languages is closed under the quotient operation.*

The proof of Property 32 can be based on the properties of the quotient operation on languages. Given any language L and any three strings x, y, and z over the alphabet, $x^{-1}(z^{-1}L) \subseteq y^{-1}(z^{-1}L)$ is satisfied if and only if $(zx)^{-1}L \subseteq (zy)^{-1}L$.

Property 33. *The class of well-structured languages is closed under the reverse operation.*

Let L be any language in the class of well-structured languages and its set of quotients $D^L = \{x_1^{-1}L, \ldots, x_n^{-1}L\}$, where $x_i^{-1}L \supseteq x_{i+1}^{-1}L$ for $1 \le i < n$. Thus, $D^{R(L)} = \{(\bigcup_{j=1}^{i}[x_j]_{\equiv_L})^r, 1 \le i \le n\}$, where $[x_j]_{\equiv_L}$ denotes Nerode's class of strings that are equivalent to x_j.

Property 33 can be proved taking into account that $(\bigcup_{j=1}^{i}[x_j]_{\equiv_L})^r \subseteq (\bigcup_{j=1}^{i+1}[x_j]_{\equiv_L})^r$.

Property 34. *The class of well-structured languages is not closed under the union or intersection operations.*

Both $L_1 = \{\lambda, a, aa\}$ and $L_2 = \{\lambda, b, bb\}$ are finite well-structured languages, but $L_1 \cup L_2$ is not. The intersection is not a closure operation for the class of well-structured languages because this class is closed under complementation, and therefore, it would lead to a contradiction.

We finally prove that the class of well-structured languages is closed under positive closure. We provide full proof because the reasoning in this case is more complex. In the proof of the property, we take into consideration a known result on non-deterministic automata that is used in several studies (e.g., [10,19]). This result states that, given two states p and p' of a non-deterministic automaton A such that $L_p^A \subseteq L_{p'}^A$, and, for some other state q and a symbol a, there exist transitions (q, a, p) and (q, a, p') in A, then, the transition (q, a, p) in A can be deleted without modifying the language $L(A)$. We illustrate the proof in Example 36.

Property 35. *Let L be any well-structured language. Then L^+ is also a well-structured language.*

Proof. Let L be a well-structured language and let $A = (Q, \Sigma, \delta, q_0, F)$ be the minimal DFA for L. Let us recall that because A is well-structured, there is a relation inclusion between the right languages of any pair of states of A.

We define the (non-deterministic) automaton $A' = (Q, \Sigma, \delta', q_0, F)$, where δ' contains every transition in A plus a transition (p, a, q), *for every transition* (q_0, a, q) *in* δ, *and every* $p \in F$ (a transition from every final state to every successor of the initial state). Note that $L(A') = L^+$.

Let us now consider the set $F' = \{q \in F - \{q_0\} : q_0 \le q\}$. We take into account the automaton A and the set F' to define the (non-necessarily minimal) DFA $A^+ = (Q - F', \Sigma, \delta^+, q_0, F - F')$, where for every state q in $Q - F'$ and any symbol a:

$$\delta^+(q, a) = \begin{cases} \delta(q, a) & \text{if } \delta(q, a) \notin F' \\ q_0 & \text{if } \delta(q, a) \in F' \end{cases}$$

The definition of A^+ satisfies the conditions stated in Proposition 28; therefore, $L(A^+)$ is well-structured. We now prove that $L(A^+) = L(A') = L^+$ and, therefore, that L^+ is well-structured.

First, for those states $p' \in F'$, the addition of the above mentioned transitions in A' implies that $L_{p'}^{A'} = L_{q_0}^{A'}$; therefore, the initial state q_0 and the states in F' can be merged in A' without modifying the language $L(A')$. The merging of these states in A' implies the modification of the transitions (p, a, q), where p is in $(Q - F) \cup \{q_0\}$ and q is in F', by transitions (p, a, q_0). The language $L(A')$ does not change because $L_q^{A'} = L_{q_0}^{A'}$.

Second, in order to construct A', it is not necessary to add transitions from the states $p \in F - F'$ to the set of successors of q_0 because $L_p^A \supseteq L_{q_0}^A$. Therefore, for any state in $F - F'$, it is not necessary to modify the transition function.

If these considerations are implemented on A', it outputs an equivalent automaton and the only (possible) difference with respect to A^+ is related to the transition function of the initial state because it is possible for the automaton to be non-deterministic. If this is the case, it is because in A' there is a loop on q_0 and a transition from q_0 to a state q, where q either is in $F - F'$ or in $Q - F$. In both cases, the non-determinism can be eliminated (without modifying the language $L(A')$) by selecting the transition to the first state (either q or q_0) according to the inclusion relation of the right languages [19].

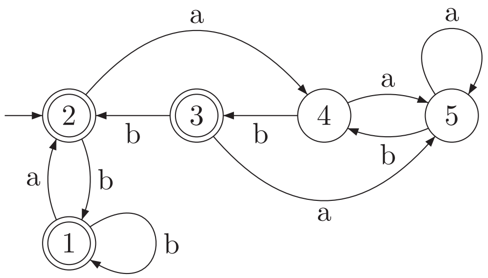


Fig. 9. A well-structured *DFA*.

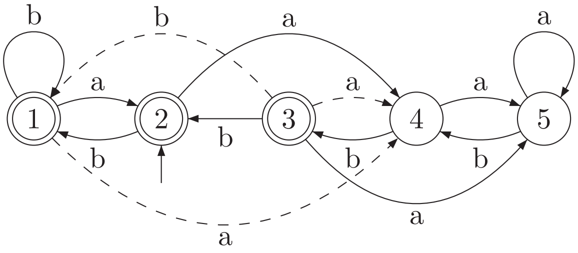


Fig. 10. A (non-deterministic) automaton that accepts the positive closure of the language identified by the *DFA* in Fig. 9.

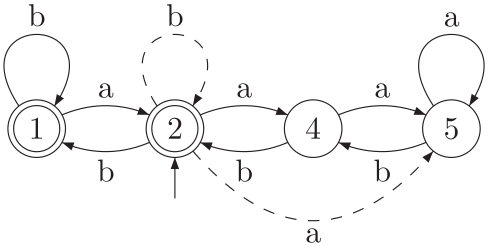


Fig. 11. An automaton that is equivalent to the *DFA* in Fig. 9 and to the automaton in Fig. 10.

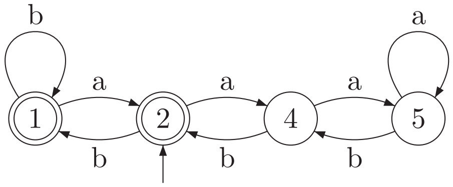


Fig. 12. The *DFA* A^+ obtained from the *DFA* in Fig. 9.

Therefore, the automaton A' can be modified to obtain A^+ without affecting the language $L(A')$. Thus, A^+ identifies the language L^+, *and, as we have stated above,* A^+ satisfies the conditions in Proposition 28. Therefore, L^+ is a well-structured language. □

Example 36. Let A be the *DFA* in Fig. 9. It can be observed that A is well-structured.

Fig. 10 shows the automata A' as the result of adding transitions from every final state to any successor of the initial state. Dashed transitions denote the ones that have been added.

Note that, in A', the addition of transitions does not affect the order between states present in A. Therefore, as shown in Proposition 35, the transition (1, a, 4) is redundant because $L_1^A \supseteq L_2^A$. Besides, the addition of the transitions (3, a, 4) and (3, b, 1) implies that $L_2^{A'} = L_3^{A'}$. Therefore, the automaton can be modified (without affecting the language) by merging states 2 and 3. The result is shown in Fig. 11.

As stated in Proposition 35, the only (possible) non-determinism is due to the initial state. In this example, we describe the two different cases that may appear. First, note that $L_1 \supseteq L_2$, therefore, the loop (2, b, 2) can be deleted without modify the language. Second, since $L_4 \supseteq L_5$, the transition (2, a, 5) can be deleted without modifying the language. The result gives the *DFA* A^+ shown in Fig. 12.

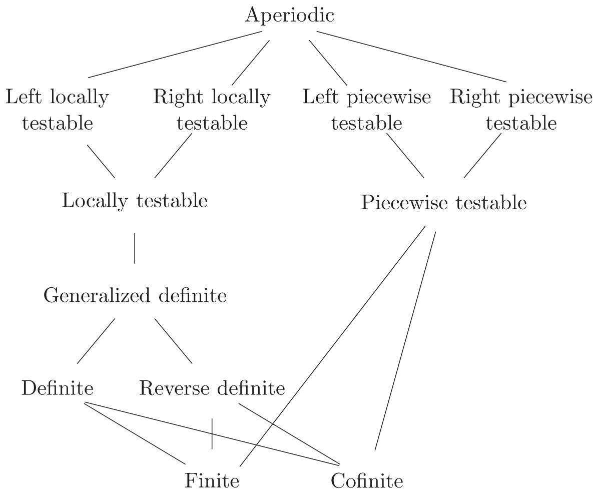


Fig. 13. Inclusion relationship between some subclasses of aperiodic languages. A connection between two classes implies that the class below is included into the upper one.

We now relate the class of well-structured languages to other well-known subclasses of regular languages. Let us first consider the class of *aperiodic* languages (also known as *star-free*, *counter-free*, or *H-trivial* languages) [20] defined as the set of languages that can be obtained by boolean and concatenation closure over the class of finite languages. Aperiodic languages are accepted by *DFA* such that, for any state q, there is no symbol $a \in \Sigma$ and $i > 1$ such that $\delta(q, a) \neq q$ and $\delta(q, a^i) = q$. According to this property, every well-structured language is also aperiodic. In order to show the distribution of well-structured languages within the class of aperiodic languages, we consider the work by Brzozowski and Knast, who prove in [4] that the class of aperiodic languages contains an infinite hierarchy with respect to the number of concatenation operations (thus obtaining an infinite number of *dot-depth* classes). In their proof of the result, we note that Brzozowski and Knast provide, for each class in the hierarchy, an automaton that accepts a language in the class but not into any other of the lower classes of the hierarchy. We note that the automata used by Brzozowski and Knast in the proof are well-structured automata, and, therefore, that the class of well-structured languages intersects with every subclass of the dot-depth hierarchy.

Second, we show the relation between well-structured languages and *ordered languages*, a subclass of aperiodic languages defined in [28]. In that paper, Shyr and Thierrin define ordered automata as the *DFA* s where the set of states can be ordered by a relation that is preserved by the transition function (i.e. given two states such that $p \preceq q$, for any symbol a of the alphabet, $\delta(p, a) \preceq \delta(q, a)$), and an ordered language is any accepted by an ordered automaton. According to this definition and Proposition 28, it is possible to see that the class of well-structured languages is properly included in the class of ordered languages.

We now recall briefly some definitions of relevant well-known subclasses of aperiodic languages. For the sake of brevity, we focus the attention on the definition. We refer the interested reader to the cited bibliography for further information. Given an alphabet Σ, the *definite* [24], *reverse definite* [5,13], and *generalized definite* [13] languages are defined as the languages of the form $A \cup \Sigma^* B$, $A \cup B\Sigma^*$ and $A \cup B\Sigma^* C$ respectively, where A, B, and C are three finite sets of strings over Σ. In [20], McNaughton and Papert define the class of locally testable languages as the set of languages that belong to the Boolean algebra generated by languages of the form $u\Sigma^*$, $\Sigma^* u$ and $\Sigma^* u \Sigma^*$, where $u \in \Sigma^*$. In [11], García and Ruiz define the classes of right and left locally testable languages as a generalization of the locally testable class. Fig. 13 shows the inclusion relationship between these classes and the more restricted classes of *finite* and *cofinite* languages.

In [29], Simon define the *Piecewise testable* languages as the set of languages that belong to the Boolean algebra generated by languages of the form $\Sigma^* a_1 \Sigma^* a_2 \ldots a_n \Sigma^*$, where $a_i \in \Sigma$ for $1 \leq i \leq n$. In [3], Brzozowski and Fich define the right piecewise testable and left piecewise testable classes taking into account the order in which the symbols a_i appear.

All these classes of languages have been algebraically characterized taking into account the syntactic semigroup properties of the languages in the class [17,26]. For instance, the classes of piecewise, left piecewise and right piecewise testable languages have been characterized using Green's equivalence relations and are also known as *J*-trivial, *L*-trivial and *R*-trivial respectively. These results permit to decide the membership of any language to any of the mentioned classes. For instance, when these methods are applied to the automata in Fig. 8 it is possible to prove that: the language accepted by automaton (a) is left locally testable but it does not belong to any other class below this one; the language accepted by automaton (c) is locally testable and left piecewise testable but it does not belong to any other class; and the language accepted by automata (e) and (f) are aperiodic but they do not belong to any other class. Note that, on the one hand, every automaton in Fig. 8 is well-structured, and, on the other hand, the finite language $\{aa\}$ is not well-structured. Therefore, it is possible to conclude that, leaving aside the proper inclusion relationship with aperiodic and ordered languages, the class of well-structured languages is incomparable to any other mentioned class.

8. Conclusions

The problem of computing a minimal separating automaton (MSA) for regular languages has been studied from many (unrelated) different points of view. The study of the complexity of the problem has been dealt with by the following authors: Trakhtenbrot and Barzdin [30], who studied the problem within a framework for synthesis of automata; Gold [14] and Angluin [1], who studied the problem in a Grammatical Inference context; and Pfleeger [25], who took into account the minimization of incompletely specified finite state machines. All of these authors reached the same conclusion: in the general case, the problem is NP-complete. Despite this exponential worst case complexity, this problem is important in many fields, from the refinement of integrated circuits or GI to Model Checking.

In this paper, we show that the problem can be reduced to the computation of a similarity relation and that this relation is equivalent to an indistinguishability relation over a Moore machine obtained from the input languages. We prove that this relation can be polynomially computed whenever the language that contains all the strings involved in the problem is well-structured. This is the most general condition proved to date that guarantees polynomial computation of the MSA.

We also prove that it is possible to decide whether or not a given language is well-structured with polynomial time complexity. We also characterize the class of well-structured automata and provide an algorithm to generate any automaton in the class. We prove that the class of well-structured languages is closed under some of the usual operations on languages. Finally, we show that the class of well-structured languages is a subclass of ordered languages and it is not comparable to several widely-studied subclasses of aperiodic languages.

References

[1] D. Angluin, On the complexity of minimum inference of regular sets, Inform. Control 39 (1978) 337–350.
[2] D. Angluin, Learning regular sets from queries and counterexamples, Inform. Comput. 75 (1987) 87–106.
[3] J.A. Brzozowski, F.E. Fich, Languages of $\mathscr{R}$-trivial monoids, J. Comput. Syst. Sci. 20 (1980) 32–49.
[4] J.A. Brzozowski, R. Knast, The dot-depth hierarchy of star-free languages is infinite, J. Comput. Syst. Sci. 16 (1978) 37–55.
[5] J.A. Brzozowski, Canonical regular expressions and minimal state graphs for definite events, in: Mathematical Theory of Automata, in: MRI Symposia Series, Polytecnic Press, Polytecnic Institute of Brooklyn, 1962, pp. pages. 529–561.
[6] C. Câmpeanu, N. Sântean, S. Yu, Minimal cover-automata for finite languages, Theor. Comput. Sci. 267 (2001) 3–16.
[7] J.-M. Champarnaud, F. Guingne, G. Hansel, Similarity relations and cover automata, Inform Theor. Appl. 39 (2005) 115–123, doi:10.1051/ita:2005006.
[8] Y.-F. Chen, A. Farzan, E.M. Clarke, Y.-K. Tsay, B.-Y. Wang, Learning minimal separating DFA's for compositional verification, in: LNCS, Tools and Algorithms for the Construction and Verification of Systems (TACAS'09), 5505, 2009, pp. 31–45.
[9] E.M. Clarke, O. Grumberg, D.A. Peled, Model Checking, MIT Press, 1999.
[10] F. Denis, A. Lemay, A. Terlutte, Learning regular languages using RFSA, Theor. Comput. Sci. 313 (2) (2004) 267–294.
[11] P. García, J. Ruiz, Right and left locally testable languages, Theor. Comput. Sci. 246 (2000) 253–264.
[12] P. García, M.V. de Parga, D. López, J. Ruiz, Learning automata teams, in: LNAI, in: 10th International Colloquium, ICGI-10, 6339, 2010, pp. 52–65.
[13] A. Ginzburg, About some properties of definite, reverse-definite and related automata, IEEE Trans. Electron. Comput. 15 (1966) 806–810.
[14] E.M. Gold, Complexity of automaton identification from given data, Inform. Control 37 (1978) 302–320.
[15] S. Gören, F.J. Ferguson, On state reduction of incompletely specified finite state machines, Comput. Elect. Eng. 33 (2006) 58–69.
[16] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley Publishing Company, 1979.
[17] J.M. Howie, Automata and Languages, Oxford Science Publications, 1991.
[18] J. Kaneps, R. Freivalds, Minimal nontrivial space complexity of probabilistic one-way turing machines, in: LNCS, Mathematical Foundations of Computer Science (MFCS'90), 452, 1990, pp. 355–361.
[19] S. Lombardy, J. Sakarovitch, The universal automaton, in: J. Flum, E. Grädel, T. Wilke (Eds.), Logic and Automata, Texts in Logic and Games, 2, Amsterdam University Press, 2008, pp. 457–504.
[20] R. McNaughton, S. Papert, Counter-free automata, The MIT Press, 1971.
[21] D. Neider, Computing minimal separating DFAs and regular invariants using SAT and SMT solvers, in: LNCS, Automated Technology for Verification and Analysis (ATVA'12), 7561, 2012, pp. 354–369.
[22] M.C. Paull, S.H. Unger, Minimizing the number of states in incompletely specified sequential switching functions, IRE Trans. Electron. Comput. (1959) 356–367.
[23] J.M. Pena, A. Oliveira, A new algorithm for exact reduction of incompletely specified finite state machines, IEEE Trans. Comput. Aided Des. Integrated Circuits Syst. 18 (11) (1999) 1619–1632.
[24] M. Perles, M.O. Rabin, E. Shamir, The theory of definite automata, IEEE Trans. Electron. Comput. 12 (1963) 233–243.
[25] C.P. Pfleeger, State reduction in incompletely specified finite-state machines, IEEE Trans. Comput. 22 (1099–1102) (1973).
[26] J.E. Pin, Varieties of formal languages, North Oxford Academic Publishers Limited, 1986.
[27] J.-K. Rho, G.D. Hatchel, F. Somenzi, R.M. Jacobi, Exact and heuristic algorithms for the minimization of incompletely specified state machines, IEEE Trans. Comput. Aided Des. Integrated Circuits Syst. 13 (2) (1994) 167–177.
[28] H. Shyr, G. Thierrin, Ordered automata and associated languages, Tamkang J. Math. 5 (1974) 9–20.
[29] I. Simon, Piecewise testable events, in: LNCS, Automata Theory and Formal Languages 2nd GI Conference, 33, 1975, pp. 214–222.
[30] B.A. Trakhtenbrot, Y.M. Barzdin, Finite Automata. Behavior and Synthesis, North-Holland Publishing Company, 1973.
[31] M.V. de Parga, P. García, D. López, Minimal consistent DFA revisited, 2015, (submitted).