Theoretical Computer Science 647 (2016) 43–49

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Minimal consistent DFA revisited

Manuel Vázquez de Parga, Pedro García, Damián López [*]

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

A R T I C L E   I N F O

Article history:
Received 2 June 2015
Received in revised form 13 April 2016
Accepted 20 July 2016
Available online 27 July 2016
Communicated by J. Karhumäki

Keywords:
Minimal consistent DFA
Uniformly-complete sample
DFA identification

A B S T R A C T

We address the classic problem of polynomial computation of a minimal finite-state representation compatible with two finite input sets that contain those strings that have to be accepted and rejected. We extend the previously defined *uniformly-complete* sample and prove that it is a special case of the sufficient condition we propose.

1. Introduction

In this paper we revisit the classic problem of finding, given a finite set of strings to be accepted and another finite set to be rejected, a minimal finite-state description compatible with the input sample.

This problem was addressed by Trakhtenbrot and Barzdin in [1]. In that work the authors prove that, whenever the input sample is *uniformly-complete*, defined as the one that contains exclusively all the strings up to a given length, then there is a polynomial algorithm that outputs a minimal DFA consistent with respect to the sample.

In [2], Gold proves that, given a positive integer n and an input finite sample divided into a *positive* and a *negative* subsets, the problem of determining whether there is a finite automaton of size at most n and compatible with the sample is NP-complete. In this same line of work, in [3], Angluin studies if conditions set on the density of the sample might guarantee the problem to be tractable. In her work, she proves that even a small reduction of an uniformly-complete sample implies the problem to be NP-complete.

A related problem is the computation of the minimal *cover-automaton* for a finite language. In [4], Câmpeanu, Sântean and Yu present an algorithm that obtains the minimal cover-automaton, a concisely representation of any finite language.

Given a finite set of strings D, the problem of obtaining the minimal cover for D is the problem of computing a DFA with the minimum number of states such that it accepts every string in D and rejects the strings in $\Sigma^{\leq l} - D$, where l denotes the length of a longest string in D, and $\Sigma^{\leq l}$ the set of strings over the alphabet of length lower than or equal to l. Note that the automaton, together with l, provide a concise representation of D. Note also that the problem coincides with the one stated by Trakhtenbrot and Barzdin considering an uniformly-complete sample.

Other related works study the separability of regular languages (usually by looking for a language in a given class) [5,6], or the computation of the minimal separating DFA of two input languages [7–9] usually related to compositional verification.

[*] Corresponding author.
E-mail address: dlopez@dsic.upv.es (D. López).

http://dx.doi.org/10.1016/j.tcs.2016.07.023
0304-3975/© 2016 Elsevier B.V. All rights reserved.

In this work we relax the conditions that guarantee the polynomial construction of a finite-state consistent representation of a set of strings. We prove a sufficient condition of the sample that assures that a minimal automaton consistent with the sample can be obtained in polynomial time. Thus, the condition of a sample to be uniformly-complete becomes a special case of our result.

2. Notation and definitions

Let Σ be a (finite) alphabet and let Σ^* be the set of strings over Σ where λ denotes the empty word. A *language* L over Σ is any subset of Σ^*. Given $x \in \Sigma^*$, if $x = uv$ with $u, v \in \Sigma^*$, then u (resp. v) is called a *prefix* (resp. *suffix*) of x. Let us denote the set of prefixes (suffixes) of x by $Pref(x)$ (resp. $Suf(x)$), and the natural extension of the prefixes (suffixes) operation to a set of strings M by $Pref(M)$ (resp. $Suf(M)$). Given any string u and a language L over Σ, let $u^{-1}L$ denote the *right quotient* of the language with respect to u (i.e., $u^{-1}L = \{v \in \Sigma^* \ : \ uv \in L\}$). We also recall here the definition of the *lexicographical 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 *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 $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is the transition function, which can also be seen as $\delta \subseteq (Q \times \Sigma \times Q)$. The transition function can be extended in a natural way to Σ^*. Given an automaton A, we say that it is accessible if, for each $q \in Q$, there exists a string x such that $q \in \delta(I, x)$. An automaton is called *deterministic* (DFA) if, for every state q and every symbol a, the number of transitions $\delta(q, a)$ is at most one, and it has only one initial state q_0. A DFA is said to be complete whenever, given any state q and any symbol a, the number of transitions $\delta(q, a)$ is always one. We will denote the language accepted by an automaton A with $L(A) = \{x \in \Sigma^* \ : \ \delta(I, x) \cap F \neq \emptyset\}$.

In a similar way an automaton is defined, a Moore machine is a 6-tuple $M = (Q, \Sigma, \Delta, \delta, q_0, \Phi)$, where Q is the set of states, $q_0 \in Q$ is the initial state, 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.

Given two disjoint finite sets of words D_+ and D_-, we define the (D_+, D_-)-*prefix tree Moore machine* ($PTMM(D_+, D_-)$) as the Moore machine having $\Delta = \{0, 1, ?\}$, $Q = Pref(D_+ \cup D_-)$, $q_0 = \lambda$, and $\delta(u, a) = ua$ if $u, ua \in Q$ and $a \in \Sigma$. For every state u, the value of the output function associated to u is 1, 0 or ? (undefined) depending whether u belongs to D_+, to D_-, or to $Q - (D_+ \cup D_-)$. We also say that a Moore machine $M = (Q, \Sigma, \{0, 1, ?\}, \delta, q_0, \Phi)$ is *consistent* with respect to (D_+, D_-) if $\forall x \in D_+$ we have $\Phi(x) = 1$ and $\forall x \in D_-$ we have $\Phi(x) \neq 1$.

In the following it will be useful to simulate any given DFA using a Moore machine. Thus, given a DFA $A = (Q, \Sigma, \delta, q_0, F)$, it is possible to simulate it with 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 $L(M) = \{x \in \Sigma^* : \Phi(\delta(q_0, x)) = 1\}$.

3. Minimal consistent DFA and automata identification from data

In the 1970s several works addressed the identification problem. The first remarkable result is due to Trakhtenbrot and Barzdin [1] who proposed a DFA synthesis algorithm that takes into account all the strings of the language up to a given length. In [1], the authors prove that the algorithm obtains a minimal consistent DFA in polynomial time whenever the input sample is *uniformly-complete*, where a uniformly-complete sample is such that exclusively contains all the strings over the alphabet up to a given length. It is to be noted that, if the input does not hold the uniformly-complete condition, then the algorithm may output an inconsistent automaton with respect to the sample.

Another interesting result is due to Gold, who proved in [2] that, given some input positive (D_+) and negative (D_-) data and a positive integer n, the problem of determining the existence of a DFA consistent with the data and with at most n states and is NP-complete. In [3], Angluin proves that, if a small amount of strings are removed from a uniformly-complete sample, and given a positive integer s, the problem to decide if there is a DFA of size at most s and consistent with respect to the sample is also NP-complete.

In [2], Gold proposes a grammatical inference algorithm that outputs the minimal deterministic automaton for the language in polynomial time when a representative enough set of data is provided. This algorithm has polynomial time complexity, and is of great interest because several subsequent algorithms can be viewed as modifications of it. Even though the works by Gold and Trakhtenbrot and Barzdin are not presented in the same context, in essence both present the same algorithm [10], that, in the following, will be referred to as TBG algorithm. Algorithm 3.1 describes the method in a unified manner using the $PTMM$ of the data, which is a presentation that is close to the one used by Trakhtenbrot and Barzdin.

We note that the original TBG algorithm is non-deterministic (line 20), but it is usually implemented taking into account an ordering over the set of strings. We note that, given any input set of strings, the result may be different depending on the order implemented. Nevertheless, if the input set is uniformly-complete, and regardless the chosen order to implement line 20, the output automaton has always the same number of states and it is consistent with the input.

The TBG algorithm takes into account two subsets of states of the $PTMM(D_+, D_-)$. The *Red* set contains the *obviously distinguishable* states already identified, where two states u and v are considered (obviously) distinguishable if there exist a string $w \in \Sigma^*$ such that $\Phi(uw)$ and $\Phi(vw)$ are both defined and different. Given $M_0 = PTMM(D_+, D_-)$, the obviously distinguishable notion (used in line 8 in Algorithm 3.1) can be described more formally as follows:

Algorithm 3.1 Trakhtenbrot–Barzdin and Gold algorithm.

```
1: Input: Two disjoint finite sets $(D_+, D_-)$
2: Output: A consistent Moore Machine
3: Method
4: $M_0 = PTMM(D_+, D_-) = (Q_0, \Sigma, \{0, 1, ?\}, \delta, q_0, \Phi_0)$;
5: $S = Pref(D_+ \cup D_-)$;
6: $Red = \{\lambda\}$;
7: $Blue = ((Red \cdot \Sigma) \cap S) - Red$;
8: while $\exists u \in Blue\ :\ \forall s \in Red,\ od(s, u, M_0) = True$ do
9:     choose first of such $u$ according the lexicographic order;
10:    $Red = Red \cup \{u\}$;
11:    $Blue = ((Red \cdot \Sigma) \cap S) - Red$;
12: end while
13: $Q = Red$; $q_0 = \lambda$;
14: for $s \in Red$ do
15:    $\Phi(s) = \Phi_0(s)$;
16:    for $a \in \Sigma$ do
17:        if $sa \in Red$ then
18:            $\delta(s, a) = sa$
19:        else
20:            $\delta(s, a) =$ any $s' \in Red$ such that $od(sa, s', M_0) = False$
21:        end if
22:    end for
23: end for
24: $M = (Q, \Sigma, \{0, 1, ?\}, \delta, q_0, \Phi)$;
25: if $M$ is consistent with $(D_+, D_-)$ then
26:    Return($M$)
27: else
28:    Return($M_0$);
29: end if
30: End Method.
```

The Red set contains the obviously distinguishablestates already identified,
The Blue set contains the set of non-red successors of a red-state.
od: obviously distinguishable.
lamda is epsilon.

od: obviously distinguishable

$$od(s, s', M_0) = True \Leftrightarrow \exists x \in \Sigma^*\ :\ \begin{cases} \Phi(\delta(s,x)), \Phi(\delta(s',x)) \in \{0,1\} \\ \Phi(\delta(s,x)) \neq \Phi(\delta(s',x)) \end{cases}$$

The *Blue* set contains the set of non-red successors of a red-state. Initially, $Red = \{\lambda\}$ and $Blue = \{a \in \Sigma\ :\ a \in Pref(D_+ \cup D_-)\}$.

Each iteration of the algorithm's first loop (line 8) chooses a blue-state u which is distinguishable from every state in the *Red* set (deterministic selection according to a lexicographic traversal of the blue-states). The *Red* and *Blue* sets are subsequently updated (adding u to the *Red* set and considering the successors of the updated *Red* set). The process ends when the *Red* set cannot be updated, that is, when for every blue-state there exists a red-state (at least one) non-obviously-distinguishable with it.

The second loop of the algorithm (line 14) traverses the set of states of the automaton (the *Red* set) in order to construct the set of transitions (s, a, sa), for any state s and any symbol a in the alphabet. Note that it is possible that sa were not in the set of states. In that case, there exists a red-state (at least one) which is not distinguishable from sa. The algorithm selects one of the possible candidates and creates the transition accordingly. This (originally non-deterministic) selection is implemented following an a-priori fixed order (usually lexicographic). Example 1 illustrates the process.

Example 1. As mentioned above, usually, the TBG algorithm is implemented taking into account an ordering of the states of $PTMM(D_+, D_-)$. In this example we will consider the lexicographic order. Let us consider finite sample $D_+ = \{a, bb, aab, aba, abb\}$ and $D_- = \{\lambda, b, aa, ab, ba, aaa, baa, bab, bba, bbb\}$. The Algorithm constructs the Moore machine $PTMM(D_+, D_-)$ that is shown in Fig. 1.

In the figure, positive and negative states are represented with double circles and single circles respectively (no undefined states are present). Note that the numbering of the states is consistent with the lexicographical order of the strings that reach each state. We will follow this lexicographical order to select the states to be processed.

Initially $Red = \{\lambda\}$ and $Blue = \{a, b\}$. In this moment the state a is distinguishable from every red-state and thus it is included into the *Red* set. Therefore, $Red = \{\lambda, a\}$ and $Blue = \{b, aa, ab\}$. The analysis of the state b also detects that it is distinguishable with respect every red-state and the update of the sets produce $Red = \{\lambda, a, b\}$ and $Blue = \{aa, ab, ba, bb\}$.

The analysis of the state aa returns that it is undistinguishable with respect to b, therefore, it is not added to the *Red* set. Next state processed is state ab which is distinguishable with respect to every red-state, therefore, $Red = \{\lambda, a, b, ab\}$ and $Blue = \{aa, ba, bb, aba, abb\}$. The analysis of the state ba returns that it has to be included in the *Red* set. Once the states are updated $Red = \{\lambda, a, b, ab, ba\}$ and $Blue = \{aa, bb, aba, abb, baa, bab\}$.

In this moment, for each state in the *Blue* set, there is a (at least one) red-state that cannot be distinguished from it (for instance, aba is not distinguishable from a, and baa is not distinguishable from λ). The algorithm uses the lexicographic order established a-priori to select the first non-distinguishable red-state to each blue-state. The automaton output is shown in Fig. 2.

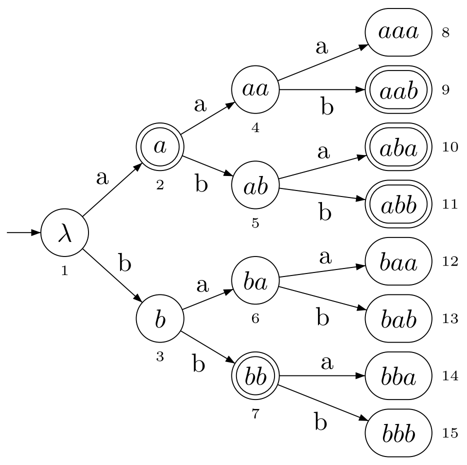

Fig. 1. PTMM example. Single-circled states have output 0 and double-circled ones have output 1.

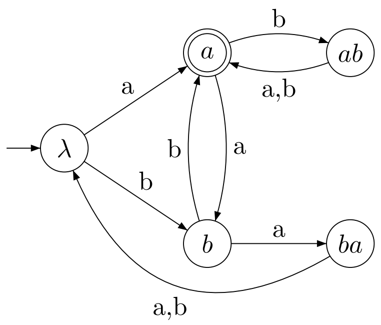

Fig. 2. Output of the TBG algorithm.

As mentioned above, in [1], the authors prove that the TBG algorithm obtains the minimal consistent DFA in polynomial time whenever the input sample is *uniformly-complete*. Otherwise, the output can be inconsistent with respect to the sample. In order to explain the reasons that may lead to this inconsistent output, note that whenever a *Blue* state y is found to be undistinguishable from a *Red* state x, for every string $w \in y^{-1}D$, either $\Phi(xw)$ or $\Phi(yw)$ are undefined, or the algorithm detects that $\Phi(xw) = \Phi(yw)$.

The problem arises when, given a red-state x and two blue-states y and z, both y and z are not obviously-distinguishable from x, but, however, y and z are obviously-distinguishable. In this case, it is possible that, for a given string w, $\Phi(xw) = ?$ and $\Phi(yw) \neq \Phi(zw) \neq ?$. The algorithm promotes neither y nor z to the *Red* set, thus assuming two different values for $\Phi(xw)$, and, therefore, obtaining an inconsistent automaton.

Despite the eventual inconsistency of this algorithm, it has been proved that the method outputs the minimal DFA for any regular language in the limit [11,2].

4. A sufficient condition for polynomial computation of a minimal consistent DFA

In this section, given a finite set of data M, we prove a sufficient condition for M that permits to obtain a minimal DFA consistent with respect to the provided data. The condition is founded on the following definition:

Definition 2. Given a finite set $M \subset \Sigma^*$, we say that M is *reasonably-complete*, if and only if, given any $u, v \in \Sigma^*$, either $u^{-1}M \subseteq v^{-1}M$ or $v^{-1}M \subseteq u^{-1}M$.

Lemma 3, and Corollary 4 as well, follow directly from this definition. We distinguish them for the sake of clarity.

Lemma 3. *Given a reasonably-complete finite set M, it fulfills that, for any $u \in \Sigma^*$ and any $a \in \Sigma$, $(ua)^{-1}M \subseteq u^{-1}M$.*

Proof. The set M is reasonably-complete, therefore, either $(ua)^{-1}M \subseteq u^{-1}M$ or $u^{-1}M \subseteq (ua)^{-1}M$. According the properties of the quotient of languages, $(ua)^{-1}M = a^{-1}(u^{-1}M)$, and therefore, because M is finite, the size of the set $a^{-1}(u^{-1}M)$ is equal to or lower than the size of $u^{-1}M$, and, therefore, $(ua)^{-1}M \subseteq u^{-1}M$. □

Example 1 $A = \{a, aab, baa\}, a^{-1}A = D_aA = \frac{dA}{da} = \{\varepsilon, ab, \emptyset\} = \{\varepsilon, ab\}$

Example 2 $L = \{ba, baa, baab, ca\}, w = \{ba\},$

$w^{-1}L = \{\varepsilon, a, ab, \emptyset\} = \{\varepsilon, a, ab\}$

$(wa)^{-1}L = (baa)^{-1}L = \{\emptyset, \varepsilon, b, \emptyset\} = \{\varepsilon, b\}$

$a^{-1}(w^{-1}L) = a^{-1}\{\varepsilon, a, ab\} = \{\emptyset, \varepsilon, b\} = \{\varepsilon, b\}$

$w \in L \equiv \varepsilon \in w^{-1}L, and (wa)^{-1}L = a^{-1}(w^{-1}L)$

$L_0 = L, L_1 = w^{-1}L, L_2 = a^{-1}(w^{-1}L) = (aw)^{-1}L$

$$\begin{tikzcd} \text{start} \arrow[r] & L_0 \arrow[r, "w"] & L_1 \arrow[r, "a"] & L_2 \end{tikzcd}$$

$w^{-1}A = \{x \in V^* : wx \in A\}$

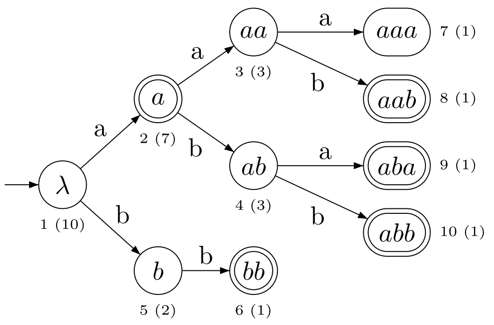

Fig. 3. PTMM example for a reasonably-complete sample. The small figures show the numbering of the states according the $\preccurlyeq$-based order (the weights of each state is shown in parentheses).

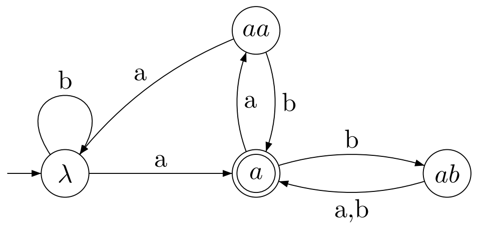

Fig. 4. Output of the TBG algorithm when implemented according the $\preccurlyeq$-based order.

Corollary 4. *Any finite reasonably-complete set M is prefix-closed (any $u \in Pref(M)$ is also in M).*

We are interested in the computation of minimal finite-state representations, thus, we say that a sample D_+ and D_- is reasonably-complete if the union of these sets is reasonably-complete. We note that, according to the definition, any uniformly-complete sample is reasonably-complete.

Given any reasonably-complete sample $M = (D_+, D_-)$, we note that it is possible to define an order over $Pref(D_+ \cup D_-)$ in such a way that, for any strings u, v, $u \preccurlyeq v$ if either $u^{-1}M \supsetneq v^{-1}M$, or $u^{-1}M = v^{-1}M$ and u precedes v in lexicographical order. We remark that, in a similar way it occurs in the TBG algorithm, it is possible to use any other criterion to number the states that have the same information available (those states represented by strings u and v such that $u^{-1}M = v^{-1}M$).

To compute this order, given any finite reasonably-complete sample $M = (D_+, D_-)$, we consider the $PTMM(D_+, D_-)$ and relate each word in $Pref(D_+ \cup D_-)$ to the corresponding state in the tree automaton. Note that it is possible to number the states of the $PTMM$ of the sample by recursively assigning to each state a weight of 1 plus the sum of the weights assigned to its successors. This way the leaves of the tree automaton (states with no successors) have weight of 1. The preorder of the states in $PTMM(M)$ is obtained when the set of weights are downward traversed.

Once the ordering of the states has been carried out, we propose an algorithm with the same structure of the TBG algorithm described in Section 3 but where the *Blue* set is traversed using the above defined order. Thus, the selection of the blue-states to be promoted to the *Red* set (line 9) is undertook according the $\preccurlyeq$ order.

We illustrate the process in Example 5 and prove in Proposition 8 that, given a reasonably-complete sample, the proposed implementation of TBG algorithm obtains a minimum DFA consistent with the input.

Example 5. Let us consider the sample $D_+ = \{a, bb, aab, aba, abb\}$ and $D_- = \{\lambda, b, aa, ab, aaa\}$, note that the set $D_+ \cup D_-$ is reasonably-complete, and, therefore, the sample is also reasonably-complete. The Algorithm constructs the Moore machine $PTMM(D_+, D_-)$ shown in Fig. 3.

We note that the numbering of the states guarantees that those states with a greater amount of information are analyzed first. Thus, initially, $Red = \{\lambda\}$ and $Blue = \{a, b\}$. The first blue-state considered is a, which is distinguishable from λ, therefore, it is included into the *Red* set. Thus, once the sets are updated, $Red = \{\lambda, a\}$ and $Blue = \{aa, ab, b\}$. Note that the enumeration of the states in the *Red* and *Blue* sets represent the $\preccurlyeq$-order that guide the processing.

The analysis of state aa returns that it is also a red-state, and the subsequent update of the sets produce $Red = \{\lambda, a, aa\}$ and $Blue = \{ab, b, aaa, aab\}$. The analysis of state ab returns that it is also distinguishable from the states in the *Red* set, and now $Red = \{\lambda, a, aa, ab\}$ and $Blue = \{b, aaa, aab, aba, abb\}$.

The processing of the remaining blue-states determines that all of them can be related to a red-state, and, therefore, the first loop ends. The algorithm constructs an automaton where any reference to a blue-state is substituted by one of its undistinguishable red states. Fig. 4 shows the output when this selection is carried out following the $\preccurlyeq$-based order.

As mentioned above, TBG process can obtain non-consistent automata when the input sample is not uniformly-complete. As an example of this, note that when the algorithm is implemented to take into account the lexicographical order, then the output obtained with the input in Example 5 is not consistent.

In order to prove in Proposition 8 the correctness of the described algorithm, some previous results are proved.

Lemma 6. *Let consider a reasonably-complete input sample $D = (D_+, D_-)$ and an implementation of TBG algorithm according the $\preccurlyeq$ preorder. Then, once k states have been processed, the $(k+1)$-th state is the successor of a state in the Red set.*

Proof. Follows from Lemma 3. □

Lemma 7. *Given an input reasonably-complete sample $D = (D_+, D_-)$, the consideration of the $\preccurlyeq$ ordering in the implementation of the TBG algorithm guarantees that, at any time during the method's execution, for every state u not yet analyzed, $u^{-1}D \subseteq v^{-1}D$ for every red-state v.*

Proof. If follows from the definition of the $\preccurlyeq$ preorder. □

Proposition 8. *Let consider a reasonably-complete input sample $D = (D_+, D_-)$. The TBG algorithm implemented according the $\preccurlyeq$ preorder always returns a minimal consistent automaton with respect to D.*

Proof. The algorithm first obtains the $PTMM$ for the input (D_+, D_-), that will be denoted by A_0 in the following.

First, we prove that the proposed algorithm obtains a machine with the minimum number of states. Let us consider that a minimum compatible automaton has n states. The algorithm does not detect more than n states (it would imply that the algorithm distinguish at least two undistinguishable states), therefore we only prove that our proposal does not output an automaton with fewer states.

In order to output an automaton with less than n states, the algorithm should include in the Red set a state x such that, there exist two non-red-states y and z, where both are undistinguishable with respect x but distinguishable one each other. Note that this is impossible, because, on the one hand, if y and z are distinguishable then there exists $w \in \Sigma^*$ such that $\Phi(yw)$ and $\Phi(zw)$ are defined and distinct, but, on the other hand, if both y and z are undistinguishable with respect to x implies that $\Phi(xw)$ is undefined, which contradicts the fact that the algorithm traverses the states according the $\preccurlyeq$ preorder.

Second, we prove that the output machine is consistent with respect the input. Note that, by Lemma 6, whenever a blue-state v is analyzed, there is a transition (u, a, v) in A_0 where $u \in Red$ and $a \in \Sigma$. Note also that, by Lemma 7, at any time during the execution of the method, the quotient of D with respect any red-state includes the quotient of D with respect v. Therefore, the state v can be either detected as a new Red state, or identified as equivalent to, at least one, red-state. When v is found to be undistinguishable with respect to a red-state u, the information available for v is a subset of the information for u, and, therefore, no inconsistency may arise. □

We recall that any implementation of the TBG algorithm must take into account the order in which the blue-states are processed and/or selected. Originally, the selection of the state to add to the Red set (line 9) is carried out according the lexicographic order. We note that any uniformly-complete sample is reasonably complete. Thus, because Lemma 3 and the definition of the $\preccurlyeq$-order, if uniformly-complete sample is provided, then both the original TBG and our proposal analyze the states in the same order, and, therefore, obtain the same set of states of the minimal DFA. Furthermore, in practice, the selection of the states to promoted to the Red set (line 9 in Algorithm 3.1) can be carried out according any order that considers first the length of the strings (the shorter strings are first in the order).

We note that both algorithms are non-deterministic in the construction of the transition function (line 9 in Algorithm 3.1). We note that, obviously, different implementations of the non-determinism may lead to different automata. Nevertheless, when the implementation of such non-determinism is the same (for instance, according a lexicographic ordering of the states) and the input is uniformly-complete, then our algorithm and the original TBG algorithm return the same automaton.

5. Conclusions

It has been proved in the literature that the problem of deciding the existence of a DFA of a given size consistent with the some input data is NP-complete [3,2]. Furthermore, Angluin proves in [3] that, even if a small amount of strings are removed from a uniformly-complete sample, the problem has the same complexity.

In this work we relax the previous condition that guaranteed the polynomial computation of a finite automaton consistent with an input sample. We define reasonably-complete sample as a generalization of the uniformly-complete condition, and we prove that, for any sample holding the definition, it is possible to obtain the minimal consistent DFA with polynomial time complexity.

Our results may seem contradicting with respect to the results by Angluin mentioned above. Nevertheless, we note that, regardless the number of strings in the input set, we demand the sample to hold certain conditions. We stress that the previous condition stated in the literature (uniform-completeness) that guarantees a finite state representation can be obtained with polynomial time complexity is a special case of the condition we propose in this work.

The question of whether it is possible to further relax the conditions that guarantee polynomial time complexity of the problem is interesting and open. We propose its study as future work.

References

[1] B.A. Trakhtenbrot, Y.M. Barzdin, Finite Automata. Behavior and Synthesis, North-Holland Pub. Co., 1973.

[2] E.M. Gold, Complexity of automaton identification from given data, Inf. Control 37 (1978) 302–320.

[3] D. Angluin, On the complexity of minimum inference of regular sets, Inf. Control 39 (1978) 337–350.

[4] C. Câmpeanu, N. Sântean, S. Yu, Minimal cover-automata for finite languages, Theoret. Comput. Sci. 267 (2001) 3–16.

[5] T. Place, L. van Rooijen, M. Zeitoun, On separation by locally testable and locally threshold testable languages, Log. Methods Comput. Sci. 10 (3) (2014) 1–28.

[6] T. Place, M. Zeitoun, Separating regular languages with first-order logic, Log. Methods Comput. Sci. 12 (1) (2016) 1–30.

[7] Y-F. Chen, A. Farzan, E.M. Clarke, Y-K. Tsay, B-Y. Wang, Learning minimal separating DFA's for compositional verification, in: Tools and Algorithms for the Construction and Verification of Systems, TACAS'09, in: LNCS, vol. 5505, 2009, pp. 31–45.

[8] S. Gören, F.J. Ferguson, On state reduction of incompletely specified finite state machines, Comput. Electr. Eng. 33 (2006) 58–69.

[9] D. Neider, Computing minimal separating DFAs and regular invariants using SAT and SMT solvers, in: Automated Technology for Verification and Analysis, ATVA'12, in: LNCS, vol. 7561, 2012, pp. 354–369.

[10] P. García, A. Cano, J. Ruiz, A comparative study of two algorithms for automata identification, in: 5th International Colloquium, ICGI 2000, in: LNAI, vol. 1891, 2000, pp. 115–126.

[11] E.M. Gold, Language identification in the limit, Inf. Control 10 (1967) 447–474.