



Learning k -Testable and k -Piecewise Testable Languages from Positive Data

PEDRO GARCÍA, JOSÉ RUIZ
Depto. de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia, Spain
E-mail: {pgarcia,jruiz}@dsic.upv.es

Abstract

The families of locally testable (LT) and piecewise testable (PWT) languages have been deeply studied in formal language theory. They have in common that the role played by the segments of length k of their words in the first family is played in the second by their subwords (sequences of non necessarily consecutive symbols), also of length k . We propose algorithms that, given $k > 0$, identify both families of languages (k - PWT and k - LT) from positive data in the limit. The first one identifies the family of k - PWT languages making use of a combinatorial property discovered by Simon and improves the complexity of a previous algorithm for that family [25]. The second algorithm identifies the family of k - LT languages using a result about the cascade product of automata. In this product, for each k , one of the factors is a fixed transducer and the second is the automaton obtained by the first algorithm for $k = 1$ using the transduction of the sample as input.

1 Introduction

The discipline known as grammatical inference, that is, the way of obtaining a representation of a formal language from examples starts with a result that can be considered negative. This result [15] states that the simplest family of languages in Chomsky's hierarchy, the class of regular languages, is not identifiable from positive data in the limit. As the problem of identifying a language from only positive examples has such strong limitations, inference from positive data seemed to be reduced to study ways of obtaining generalizations of the training sample by means of heuristics, without having the possibility of characterizing the obtained results [5], [11], [17]. However, there are non trivial families of languages identifiable from positive data in the limit, and a characterization of those families has been proposed in [2]. Afterwards, algorithms that identify in the limit certain interesting subclasses of regular languages have been proposed in [3], [14], [24] and even a general setting named function distinguishable has been proposed in [10].

Locally Testable Languages were introduced by McNaughton [22] and algebraically characterized by Brzozowsky and Simon [7] and by Zalcstein [27]. Given an integer $k > 0$, we say that L is k -Testable (k - T) if, given a word $x \in L$, any other word having the same prefix and suffix of length $k - 1$ and containing exactly the same segments of length k than x also belongs to L . The order of appearance of the segments in the words of L is immaterial. A language is *Locally Testable* if it is k -Testable for a value of k .

The family of *Piecewise Testable Languages* (PWT) has been previously studied by Simon [26] and Lothaire [21]. Given an integer $k > 0$, we say that L is k -Piecewise Testable (k - PWT) if, given a word $x \in L$, any other word having the same set of subwords of length less than or equal to k than x also belongs to L . By means of *subword of length k* of a word we understand in this context, a sequence of non necessarily consecutive symbols taken from a word. A language is called PWT if for some integer k , L is k - PWT .

The behavior of both segments and subwords of length k of a word x is somehow similar. If we consider the order of appearance of the segments in the prefixes (resp. suffixes) of x we obtain the families of Right and (resp. Left) Locally Testable languages [12]. If we do so with subwords of x we obtain the families of Right and Left Piecewise Testable languages [6], [20]. Both families of locally testable and piecewise testable languages are the intersection of the respective families in which we consider the order.

The Join of the Right and Left families in both cases have a similar behavior also [1], [13].

We propose in this work two algorithms which are very distinct in nature. The first one takes advantage of a combinatorial property of subwords of length k [26] of a word x and infers the family of k - PWT languages from positive data in the limit. It will be called k -Piecewise Testable Inference Algorithm and will be denoted as k - $PWTI$.

It improves the time complexity of the algorithm proposed in [25] for the same family. If we call n to the sum of the lengths of the input words and m to the size of the alphabet, the cost of the new algorithm is bounded above by $k(\log m)n$, while the former one was bounded above by $k^3m^k(\log m)n$. This means that if we consider k as a parameter, while the later algorithm is polynomial, the former is *strongly uniform fixed-parameter tractable* [8].

As we consider the order of appearance of the subwords in the prefixes of the words of the sample, at every step of the process, the language recognized by the output automaton is the smallest k -Right Piecewise Testable that contains the words of the sample read so far.

The second algorithm identifies the family of k - T languages from positive data in the limit. It will be called k -Testable Inference Algorithm and denoted as k - TI .

The method uses a fixed transducer (for each k) that is first used to obtain the segments of length k of each word of the sample and is afterwards used as one of the factors of the product that obtains the output automaton. The second factor of the product is the automaton obtained by the k - $PWTI$ algorithm for $k = 1$ using the transduction of the sample as input. We again see the connections between subwords

and segments.

This second algorithm can be applied to the inference of the family of k -Testable Languages in the Strict Sense also. As the known algorithm for this family is efficient [14] we give an example just for better understanding of the new algorithm.

2 Definitions and notation

2.1 General definitions concerning formal languages

In this section we will describe some facts about formal languages in order to make the notation understandable to the reader. For further details about the definitions, the reader is referred to [9], [19] and [23].

Let Σ be a finite alphabet and Σ^* the free monoid generated by Σ with concatenation as the binary operation and ε as neutral element. Any subset $L \subseteq \Sigma^*$ is called *language*, we will refer to its elements as *words* and the length of a word x will be denoted as $|x|$. Let Σ^k (resp. $\Sigma^{\leq k}$) be the set of words of length k (resp. less than or equal to k) over Σ . The set of symbols that a word x contains is denoted as $\alpha(x)$.

Given $x \in \Sigma^*$, if $x = uvw$ with $u, v, w \in \Sigma^*$, then u (resp. w) is called *prefix* (resp. *suffix*) of x , whereas v (also u and w) is called a *segment* of x . The set of prefixes (resp. suffixes) of a word x will be denoted as $\text{Pr}(x)$ (resp. $\text{Suf}(x)$).

A deterministic finite automaton (*DFA*) is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is a finite alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and δ is a partial function that maps $Q \times \Sigma$ into Q , which can be extended to words by establishing $\delta(q, \varepsilon) = q$ and $\delta(q, xa) = \delta(\delta(q, x), a), \forall q \in Q, \forall x \in \Sigma^*, \forall a \in \Sigma$.

A word x is accepted by an automaton A if $\delta(q_0, x) \in F$. The set of words accepted by A is denoted by $L(A)$.

A sequential machine is a sextuple $A = (Q, \Sigma, \Delta, \delta, \lambda, F)$ where Q, Σ and δ are defined in the same way as in a *DFA*, Δ is the output alphabet and the output function λ is a function that maps $Q \times \Sigma$ into Δ^* , which can be extended to $Q \times \Sigma^*$ by establishing $\lambda(q, \varepsilon) = \varepsilon$, and $\lambda(q, xa) = \lambda(q, x)\lambda(\delta(q, x), a)$.

Let $L \subseteq \Sigma^*$ and let \equiv be an equivalence relation defined in Σ^* . We say that the relation \equiv saturates L , if L is the union of equivalence classes modulo \equiv . An equivalence relation is called a congruence if it is both-sides compatible with the operation of the monoid.

We use the model of learning called identification in the limit [15]. An algorithm A identifies a class of *DFAs* H in the limit iff for any $M \in H$, on input of any presentation of $L(M)$, the infinite sequence of *DFAs* output by A converges to a *DFA* M' such that $L(M) = L(M')$.

2.2 Definitions concerning k -piecewise testable languages

Given $x, y \in \Sigma^*$, we say that $x = a_1a_2\dots a_n$, with $a_i \in \Sigma, i = 1, 2, \dots, n$ is a *subword* of y , and we denote this relationship by $x \mid y$ if $y = z_0a_1z_1a_2\dots a_nz_n$ with $z_i \in$

Σ^* , $i = 0, 1, 2, \dots, n$. This relationship is compatible with the concatenation, that is, $x \mid y \wedge x' \mid y' \Rightarrow xx' \mid yy'$.

Given a word $y \in \Sigma^*$, the set of subwords of y and their multiplicity can be evaluated in several ways [21].

Definition 1 For a word x , let

$$S(x, k) = \begin{cases} \{x\} & \text{if } |x| < k \\ \{z \in \Sigma^k : z \mid x\} & \text{if } |x| \geq k \end{cases}$$

and $NS(x, k) = \Sigma^{\leq k} - S(x, k)$. We define in Σ^* the equivalence relation \equiv_k as follows:

$$\forall x, y \in \Sigma^* \quad (x \equiv_k y \Leftrightarrow S(x, k) = S(y, k))$$

The above relationship is a congruence of finite index. We denote as $[x]_k$ the equivalence class of the word x , that is,

$$[x]_k = \bigcap_{a_1 \dots a_k \in S(x, k)} \Sigma^* a_1 \Sigma^* \dots \Sigma^* a_k \Sigma^* - \bigcup_{a_1 \dots a_k \in NS(x, k)} \Sigma^* a_1 \Sigma^* \dots \Sigma^* a_k \Sigma^*$$

Properties [21]:

1. \equiv_{k+1} is a refinement of \equiv_k .
2. Any equivalence class modulo \equiv_k is either a singleton or has an infinite number of words.
3. If $x = uv$ is such that there exists $a \in \Sigma$ with $S(ua, k) = S(u, k)$ then for every $n > 0$, we have that $ua^n v \in [x]_k$.
4. Let $k > 0$. $u \equiv_k uv$ if and only if there exist $u_1, u_2, \dots, u_k \in \Sigma^k$ such that $u = u_1 u_2 \dots u_k$ and $\alpha(u_1) \supseteq \alpha(u_2) \supseteq \dots \supseteq \alpha(u_k) \supseteq \alpha(v)$.

Definition 2 A language L is called *k-Piecewise Testable* if and only if it is the union of equivalence classes modulo \equiv_k . A language L is *piecewise testable* if there exists a value of k such that L is *k-piecewise testable*.

Definition 3 For every $x, y \in \Sigma^*$ we say $x \equiv_{k,R} y$ if and only if :

1. For every $u \in \text{Pr}(x)$ there exists $v \in \text{Pr}(y)$ such that $u \equiv_k v$.
2. For every $u \in \text{Pr}(y)$ there exists $v \in \text{Pr}(x)$ such that $u \equiv_k v$.

Informally, two words $x, y \in \Sigma^k \Sigma^*$ are $\equiv_{k,R}$ -equivalent if the order of appearance of new subwords in both words when they are explored from left to right is the same. For example, $aaba \equiv_2 aba$, but these two words are not related using $\equiv_{2,R}$. The relation $\equiv_{k,L}$ is defined in the same way by replacing the prefixes by the suffixes in the above definition.

The four properties of \equiv_k hold for $\equiv_{k,R}$ and for $\equiv_{k,L}$ also (prop. 4 is left-right dual for $\equiv_{k,L}$). One important fact about $\equiv_{k,R}$ and $\equiv_{k,L}$ which does not hold in \equiv_k is that every equivalence class has only one shortest element.

Definition 4 A language L is k -Right (resp. k -Left) Piecewise Testable iff it is saturated by $\equiv_{k,R}$ (resp. $\equiv_{k,L}$). L is Right (resp. Left) Piecewise Testable iff it is k -Right (resp. Left) Piecewise Testable for any value of k . The family of Piecewise Testable languages is the intersection of the families of Right and Left Piecewise Testable languages [6], [26].

Definition 5 Given a DFA, $A = (Q, \Sigma, \delta, q_0, F)$, we say that $q \in Q$ is at level i if and only if $i = \min\{|x| : x \in \Sigma^*, \delta(q_0, x) = q\}$

The following properties come out directly from the above definitions:

1. If L is a k -piecewise testable language, then L is j -piecewise testable, $\forall j \geq k$.
2. If A is an automaton accepting a k -piecewise testable language L and $j < i$, there are no transitions in A from a state at level i to states at level j .
3. The class of k -piecewise testable languages is finite, upper bounded by $|\mathcal{P}(\mathcal{P}(\Sigma^k))| = 2^{2^{|\Sigma|^k}}$.

2.3 Definitions concerning k -testable languages

Given $k > 0$, the prefix (resp. suffix) of length $k - 1$ of a word $x \in \Sigma^*$ will be denoted as $i_{k-1}(x)$ (resp. $f_{k-1}(x)$), whereas the set of segments of length k of x will be denoted as $t_k(x)$.

Definition 6 $\forall x, y \in \Sigma^*$ the relation \sim_k is defined as follows:

1. If $|x| < k$, $x \sim_k y$ if and only if $x = y$.
2. If $|x| \geq k$, $x \sim_k y$ if and only if
 - (a) $i_{k-1}(x) = i_{k-1}(y)$.
 - (b) $f_{k-1}(x) = f_{k-1}(y)$.
 - (c) $t_k(x) = t_k(y)$.

Definition 7 A language L is called k -testable if and only if it is saturated by the congruence \sim_k . A language L is locally testable if there exists a value of k such that L is k -testable.

Definition 8 For every $x, y \in \Sigma^*$ we say $x \sim_{k,R} y$ if and only if $x \sim_k y$ and:

1. For every $u \in \text{Pr}(x)$ there exists $v \in \text{Pr}(y)$ such that $t_k(u) = t_k(v)$.
2. For every $u \in \text{Pr}(y)$ there exists $v \in \text{Pr}(x)$ such that $t_k(u) = t_k(v)$.

Two words $x, y \in \Sigma^k \Sigma^*$ are $\sim_{k,R}$ -equivalent if they are \sim_k -equivalent and the order of appearance of new segments in both words when they are explored left to right is the same. The relation $\sim_{k,L}$ is defined in a similar way, by replacing the prefixes with the suffixes of x and y in the above definition.

It is easily seen that $\sim_{k,R}$ and $\sim_{k,L}$ are congruences of finite index that refine the congruence \sim_k .

Definition 9 We say that a language L is *k-Right Testable (k-RT)* if it is saturated by the relation $\sim_{k,R}$. L is *Right Locally testable (RLT)* if it is *k-RT* for any value of $k \geq 1$. The families of *k-Left Testable (k-LT)* and *Left Locally Testable (LLT)* are defined in a similar way.

Proposition 10 [12] $LT = RLT \cap LLT$.

3 k -PWTI learning algorithm

Given $k > 0$ we can associate to a positive sample $S = \{x_1, x_2, \dots, x_n\}$ a k -piecewise testable language $L_k(S)$ as follows:

$$x \in L_k(S) \Leftrightarrow \text{there exists } i, 1 \leq i \leq n \text{ with } S(x, k) = S(x_i, k)$$

Properties:

1. $S \subseteq L_k(S)$.
2. $L_k(S)$ is the smallest k -piecewise testable language that contains S .
3. $S' \subseteq S$ implies that $L_k(S') \subseteq L_k(S)$.
4. $L_{k+1}(S) \subseteq L_k(S)$.
5. If $k > \max_{x_i \in S} |x_i|$ then $L_k(S) = S$.

Based in the above definitions and in properties 3 and 4 of 2.2, we propose the k -Piecewise Testable Inference algorithm (*k-PWTI*) that, with input of a positive sample S , outputs a *DFA* consistent with S that accepts the smallest k -Right Piecewise Testable language that contains S , which is also a subset of the smallest k -PWT language that contains S .

The input to the algorithm is a natural number and a set of words S . It incrementally constructs an automaton by means of adding states and transitions when the suffix of the word which is being read can't be analyzed by the existing automaton. Generalizations (i.e. loops) are added to the automaton when the length of a list (which is controlled by the procedure *Update*) reaches length k .

For better understanding of this procedure, let us suppose that $k = 3$ and $List = \{\{a, b\}, \{a\}\}$. When the algorithm calls the procedure *Update*, it has to place the new symbol in *List*, so we may have to rearrange the list to keep up with the relation $List_i \supseteq List_{i+1}$. The following different situations may happen depending on the next symbol a_i , on the length of the previous list and on the list itself:

Algorithm k -Piecewise Testable Inference**Input:** $k \in \mathbb{N}$, S set of words over Σ^* **Output** DFA $A = (Q, \Sigma, \delta, q_1, F)$, consistent with S : $L(A) \subseteq L_k(S)$ **Method:** $Q = \emptyset$; $\delta = \emptyset$; $F = \emptyset$; $\forall x = a_1a_2\dots a_n \in S$ **do** $i = 1$; $q = q_1$; $List = \emptyset$; **While** ($i \leq n$) **do** /*Try to analyze the word x in A^* */ **While** ($\exists q' \in \delta(q, a_i)$) **do** $q = q'$; $i = i + 1$; **If** $i = n$ **Then** $F = F \cup \{q\}$ **End If**; **Update** [$List, q, a_i$] **End While**; /*The suffix $a_i a_{i+1} \dots a_n$ can not be analyzed in A^* */ **If** $i = n$ **Then** $F = F \cup \{q\}$ **Else** $Q = Q \cup \{new\}$; $\delta = \delta \cup (q, a_i, new)$; $q = new$; $i = i + 1$; **End If** **Update** [$List, q, a_i$] **End While**;**End**

1. $a_i = a$, then $List = \{\{a, b\}, \{a\}, \{a\}\}$ and, as $|List| = k$, $\delta = \delta \cup \{q, a, q\}$.
2. $a_i = b$, then $List = \{\{a, b\}, \{a, b\}\}$.
3. $a_i = c$, then $List = \{\{a, b, c\}\}$.

3.1 Convergence and time complexity of k -PWTI algorithm

3.1.1 Example of run

Let $k = 2$ and $S = \{aba, aaba, aa\}$.

The output automaton for the first word of the sample, aba , is represented in Figure 1. Generalizations are only made when the ordered list L that characterizes the current state of the automaton reaches length k . The evolution of that list after reading each new symbol can also be observed in Figure 1.

Note that we only have to keep the list L associated to the current state of the automaton and also that L is ordered in the way that every set of L contains the next one. Every new symbol is placed in the first set that makes L to continue ordered. L stretches in or out depending of the new incoming symbol. We have used an alphabet of two symbols to keep the examples of a manageable size.

```

Procedure Update [ $List, q, a_i$ ]
/* $List = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$  with  $\alpha_1 \subseteq \Sigma$  and  $\alpha_j \supseteq \alpha_{j+1}$ ,  $n \leq k^*$ */
If  $|List| = \emptyset$  Then  $Append(List, \{a_i\})$ 
Else
   $j = |List|$ ;
  While  $a_i \notin \alpha_j$  and  $j > 1$  do
    If  $a_i \in \alpha_{j-1}$ 
      Then  $Append(List, \{a_i\}); j = 0$ ;
    Else  $\alpha_j = \emptyset$ ;
    End If
   $j = j - 1$ ;
  End While
  If  $a_i \in \alpha_j$  and  $j < k$  Then  $Append(List, \{a_i\})$  End If
  If  $j = 1$  Then  $\alpha_1 = \alpha_1 \cup \{a_i\}$  End If
End If
/*Generalizations are made using properties 3 and 4 of 2.2*/
If  $|List| = k$  Then  $\forall a \in \alpha_k, \delta = \delta \cup \{q, a, q\}$  End If
End

```

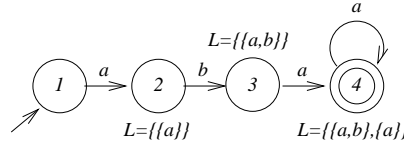


Figure 1: Output automaton of the algorithm k-PWTI after analyzing the word aba .

With the word $aaba$, the transition $(1, a, 2)$ is used for the prefix a . As the transition $\delta(2, a)$ does not exist, state 5 and transition $(2, a, 5)$ are created. The evolution of the list and the output automaton are shown in Figure 2. The word aa , only establishes state 5 as final state.

3.1.2 Convergence

The same reasons for convergence that were given in [25] can be applied here. The algorithm respects the order of appearance of the subwords in the words of the sample so, at every step of the process (when a new word has been analyzed) the language recognized by the output automaton belongs to the family of Right Piecewise Testable languages, that strictly contains the family of PWT languages. We will only obtain the smallest PWT language that contains the sample when that sample contains at least a word for every possible order of appearance of the subwords.

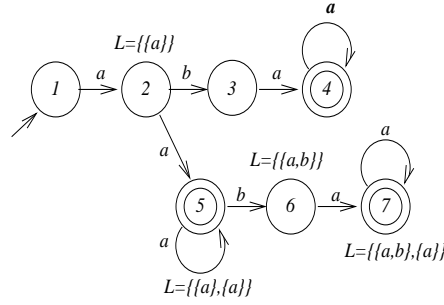


Figure 2: Automaton produced on further input of $\{aaba, aa\}$ to k -PWTI algorithm. The evolution of the list L is only marked at the states of the new part of the automaton.

3.1.3 Time complexity

At every step of the algorithm, the state of the automaton which is being analyzed is represented by a list $L = \{\alpha_1, \dots, \alpha_i\}$ where $i \leq k$ and $\alpha_j \subseteq \Sigma$, $1 \leq j \leq k$. Every new symbol of the data needs to update L , which means that it has to place the symbol in the correct α_j . The list is set to $\{\alpha_1, \dots, \alpha_j\}$ and that operation costs $k \log |\Sigma|$ in the worst case. This operation has to be done a number of times less than or equal to $n = \sum_{x \in S} |x|$, so the total cost of the algorithm is at most $k \log |\Sigma| n$, which improves the cost of $k^3 |\Sigma|^k \log |\Sigma| n$ of the algorithm proposed in [25].

4 k -TI Algorithm for k -Testable Languages.

Definition 11 For $k > 0$, let $\tau_k = (Q, A, B, \delta, \lambda, p_0, F)$ be a sequential machine defined as $Q = \cup_{i=0}^{k-1} A^i$, $p_0 = \varepsilon$, $F = Q$ and for every $p \in Q$ and $a \in A$, the transition and output functions are respectively defined as:

$$\delta(p, a) = \begin{cases} pa & \text{if } |p| < k - 1 \\ f_{k-1}(pa) & \text{if } |p| = k - 1 \end{cases}$$

$$\lambda(p, a) = \begin{cases} \varepsilon & \text{if } |p| < k - 1 \\ pa & \text{if } |p| = k - 1 \end{cases}$$

The sequential machine τ_k , for a given $k > 0$ and a word $x \in A^*$, outputs a word $\tau_k(x)$ whose symbols are the segments of length k of x , in order. Examples of τ_k for the values $k = 2$ and $k = 3$ and $A = \{a, b\}$ can be seen in Figure 3.

Definition 12 Given a sample S , let $\mathcal{A}(S)$ be the automaton defined as $\mathcal{A}(S) = (Q, B, \delta', q_0, F')$, where the alphabet is $B = \alpha(\tau_k(S)) = t_k(S)$, $Q = \{l(u) : u \in \text{Pr}(S)\}$, that is, every state is defined by an ordered list that contains the alphabet of the prefix of a word of the sample, $q_0 = l(\varepsilon)$, $F' = \{l(y) : y \in t_k(S)\}$ and the transition function

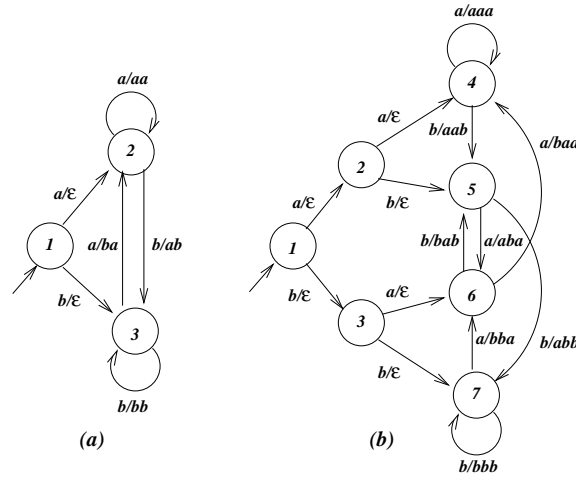


Figure 3: Transducers used for inference: (a) for $k = 2$, (b) for $k = 3$.

defined as $\delta'(l(u), b) = l(ub)$ for every $l(u) \in Q$ and every $b \in B$, where the list l can be recursively defined as follows:

$$l(\epsilon) = \{\} \text{ and}$$

$$l(ub) = \begin{cases} l(u) & \text{if } b \in l(u) \\ l(u) \wedge b & \text{otherwise} \end{cases}$$

(\wedge means appending an element to a list)

Proposition 13 *The automaton $\mathcal{A}(S)$ recognizes the smallest 1-Right Testable Language that contains the sample $\tau_k(S)$.*

Proof. If we consider that 1-Right Piecewise Testable and 1-Right Testable languages coincide, applying algorithm k -PWTI for $k = 1$ we obtain automaton $\mathcal{A}(S)$. Note that if we have a word for every order of appearance of the symbols of B during the inference process, the language recognized by the automaton $\mathcal{A}(S)$ is 1-Testable. ■

Theorem 14 (Ginzburg and Rose, see [4]) *Let $\tau_k : A^* \rightarrow B^*$, be the sequential function realized by the transducer $\tau_k = (P, A, B, \delta_1, \lambda, p_0, F)$, let $\mathcal{A} = (Q, B, \delta_2, q_0, F')$, and let $L = L(\mathcal{A})$. The language $\tau_k^{-1}(L) \subseteq A^*$ is recognized by the product¹ $\mathcal{A} \circ \tau_k = (Q \times P, A, \delta, [q_0, p_0], F' \times F)$, with the transition function defined as $\delta([q, p], a) = (\delta_2(q, \lambda(p, a)), \delta_1(p, a))$.*

¹This product is mentioned in the literature about formal languages as *cascade product*.

Theorem 15 *Let $|x| \geq k$ and let $\mathcal{A}(x)$ be the automaton of definition 12 for $S = \{x\}$ and let τ_k be the sequential machine of definition 11. We have that $L(\mathcal{A}(x) \circ \tau_k) = [x]_{k,R}$, where $[x]_{k,R}$ is the equivalence class of the word x for the congruence that defines the k -Right Testable Languages.*

Proof. Let $y = \tau_k(x) \in L(\mathcal{A}(x))$, then $x \in \tau_k^{-1}(y)$ and thus $x \in L(\mathcal{A}(x) \circ \tau_k)$.

We will see first that $[x]_{k,R} \subseteq L(\mathcal{A}(x) \circ \tau_k)$. Let A' be the automaton $\mathcal{A}(x) \circ \tau_k$, that is $A' = (Q \times P, A, \delta, (\{\}, \varepsilon), F \times \{f_{k-1}(x)\})$ and let $w = zu$ with $|z| = k - 1$, then by definitions of $\mathcal{A}(x)$, τ_k and δ , as in this case $\tau_k(z) = z$ we have $\delta((\{\}, \varepsilon), z) = (\{\}, z)$. We will see using induction in $|v|$ that for every v such that $l(zv) \subseteq l(w)$ we have that $\delta((\{\}, z), v) = (l(zv), f_{k-1}(zv))$. The base case follows from the definition of δ , if $a \in A$ then $\delta((\{\}, z), a) = (l(za), f_{k-1}(za))$.

Let us suppose that $\delta((\{\}, z), z') = (l(zz'), f_{k-1}(zz'))$, then $\delta((\{\}, z), z'a) = \delta((l(zz'), f_{k-1}(zz')), a) = (l(zz'a), f_{k-1}(zz'a))$.

Then we have $\delta((\{\}, \varepsilon), w) = (l(w), f_{k-1}(w)) = (t_k(x), f_{k-1}(x)) \in F \times \{f_{k-1}(x)\}$ and then $w \in L(\mathcal{A}(x) \circ \tau_k)$.

The reciprocal, $L(\mathcal{A}(x) \circ \tau_k) \subseteq [x]_{k,R}$, directly comes from the construction of $\mathcal{A}(x)$ and τ_k , as any word $w \in L(\mathcal{A}(x) \circ \tau_k)$ has the same prefix and suffix of length $k - 1$, and also the same segments of length k than x . The order of appearance of the segments when we scan the word w from left to right is the same as in x .

Note that some changes have to be made in the transducer of definition 11 if we want to work with words of length less than k . We have to set $\lambda(p, a) = \sharp^{k-|p|}pa$ if $|p| < k - 1$ instead of $\lambda(p, a) = \varepsilon$, where \sharp is a symbol not present in B . ■

4.1 k -TI Learning Algorithm

The previous definitions and theorems permit us to propose an algorithm that, for every sample S , and a value of $k > 0$ obtains the smallest k -Right Testable Language that contains S and thus identifies the family of k -Testable languages from positive data in the limit. The scheme of the method is depicted bellow. For each $x \in S$ we obtain $\tau_k(x)$, that is, the word whose symbols are the segments of length k of x , using a fixed transducer τ_k (for each k). Afterwards we infer the automaton $\mathcal{A}(S)$, which recognizes the smallest 1-Right Testable language that contains the transduced word and finally we realize the cascade product $\mathcal{A}(S) \circ \tau_k$.

$$\begin{array}{ccc}
 S & \xrightarrow{\tau_k} & \tau_k(S) \\
 \downarrow k\text{-TI} & & \downarrow 1\text{-TI} \\
 LT_{k,R}(S) & \xleftarrow{\tau_k^{-1}} & LT_{1,R}(S)
 \end{array}$$

The only difficulty that we may have if we apply theorem 15 to the inference problem, as we have done the proof for just one equivalence class, is the way of establishing the set of final states in the output automaton when we have words belonging to several equivalence classes. The easiest solution is to set up $\delta(q_0, S) = \cup_{x \in S} \delta(q_0, x)$ as the set of final states of the automaton. Note that when we have

in the sample S a word for every possible order of appearance of the segments of length k in each of the equivalence classes, the convergence has been achieved and thus $LT_{1,R}(S)$ and $LT_{k,R}(S)$ have to be replaced respectively by $LT_1(S)$ and $LT_k(S)$.

4.2 Examples of run

4.2.1 An example of inference of a k -Testable language

Let $S = \{abba, aba\}$ and let $k = 2$. The transduction of the sample gives us the set of words $\{\langle ab, bb, ba \rangle, \langle ab, ba \rangle\}$ over the alphabet $\{ab, ba, bb\}$. Using this alphabet, applying the k -PWT algorithm for $k = 1$, the base case is obtained, giving us the automaton depicted in Figure 4.

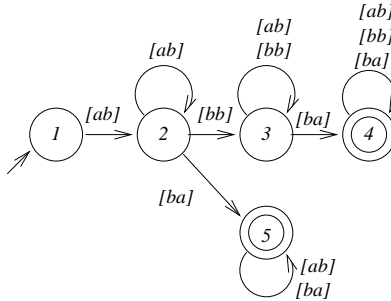


Figure 4: Base case obtained for the sample $S = \{abba, aba\}$ and the value $k = 2$.

We recall that the transition function of the cascade product is $\delta([q, p], a) = (\delta_2(q, \lambda(p, a)), \delta_1(p, a))$, where δ_1 and λ are respectively the transition and the output function of the transducer, whereas δ_2 is the transition function of the base-case automaton. For example $\delta([2, 3], a) = (\delta_2(2, \lambda(3, a)), \delta_1(3, a)) = (\delta_2(2, ba), 2) = (5, 2)$.

After doing the cascade product by the automaton of Figure 3 (a), we obtain the automaton of Figure 5, which recognizes the smallest 2-Testable language that contains the word $abba$.

4.2.2 A particular case: The k -Testable Languages in the Strict Sense

Using the same scheme we can learn the smallest k -Testable Language in the Strict Sense that contains a sample S . We recall that a language L is k -Testable in the Strict Sense (k -TSS) if there exist three sets $I, F \subseteq \Sigma^{k-1}$ and $T \subseteq \Sigma^k$ such that $L \cap \Sigma^{k-1}\Sigma^* = I\Sigma^* \cap \Sigma^*F - \Sigma^*T\Sigma^*$. A language L is Locally Testable in the Strict Sense (LTSS) if it is k -TSS for any value of k . LT languages are the boolean closure of $LTSS$ languages. An algorithm that efficiently infers the family of k -TTSS has been proposed in [14].

You should observe that for $k = 1$ and a given sample S , the smallest locally testable language in the strict sense that contains S is $\alpha(S)^*$, as in the former expression, $I = F = \{\varepsilon\}$ and T is the set of symbols not present in S .

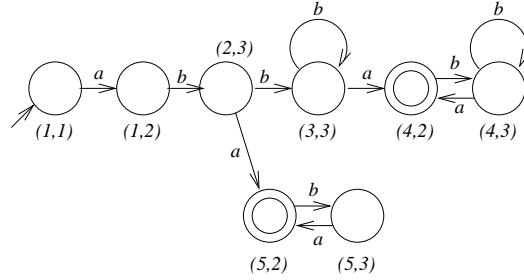


Figure 5: Automaton that recognizes the smallest 2-Right Testable Language that contains the sample $S = \{abba, aba\}$.

We adapt theorem 15 being $\mathcal{A}(S)$ the automaton that recognizes the smallest 1-Testable Language in the Strict Sense that contains $\tau_k(S)$ (i.e. the alphabet is the set of segments of length k contained in S), that is, $\mathcal{A}(S) = (\{q_0\}, t_k(S), \delta_2, q_0, \{q_0\})$, where $\delta_2(q_0, a) = q_0, \forall a \in t_k(S)$. The automata for the cases $k = 2$ and $k = 3$ and the sample $S = \{abbaba, aba\}$ are shown in Figure 6.

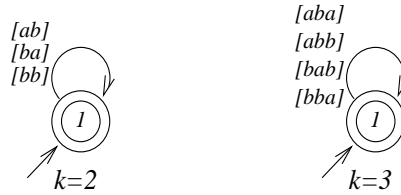


Figure 6: Smallest 1-testable language that contains $\tau_k(S)$ for the values $k = 2$ and $k = 3$, being $S = \{abbaba, aba\}$.

If we realize the cascade product of each of these automata by the sequential transducers of figure 3 we obtain the output automata shown in Figure 7.

4.3 Convergence and time complexity

4.3.1 Convergence

The convergence of the method follows directly from theorem 15, as for every word we obtain its equivalence class, which is the smallest k -Right Testable Language that contains it.

4.3.2 Time complexity

The cost of generating the sequential transducer, which is fixed for every values of k and Σ , is $|\Sigma|^k$. On the other hand, if we call n to the sum of the lengths of the words

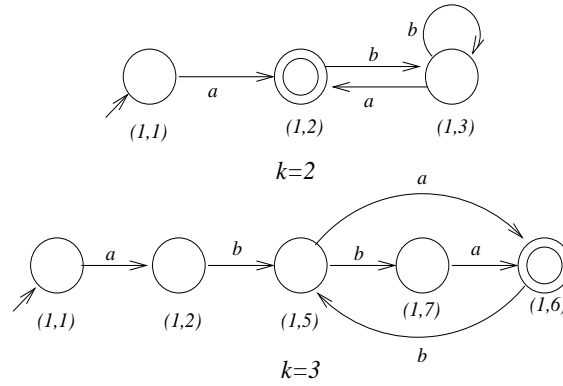


Figure 7: Automaton that recognizes the smallest k -testable languages in the strict sense for the sample $S = \{abbaba, aba\}$ and the values $k = 2$ and $k = 3$.

of the sample, the cost of inferring the base case is $|\Sigma|^{kn}$, as the size alphabet we use is upper bounded by $|\Sigma|^k$. That would give us a cost of $|\Sigma|^{2kn}$, which is clearly linear with the length of the sample. We remark that this complexity is the same as the complexity reported in the existing algorithm for this class [14].

5 Conclusions

We have presented algorithms that infer the families of k -Testable and k -Piecewise Testable languages from positive presentation in the limit. We had shown in two previous works [12], [13] that segments and subwords behave in a similar manner in questions related to order. This work intends to give another step to fill in the gap existing between subwords and segments, on the one hand, and segments and symbols on the other. As a by-product we have shown that the family of k -Testable Languages in the Strict Sense can be inferred using the same cascade product that we use for the k -Testable languages.

References

- [1] J. Almeida, A. Azevedo, “The join of the pseudovarieties of R-Trivial and L-Trivial monoids”, *Journal of Pure and Applied Algebra*, 60, pp. 129-137, 1989.
- [2] D. Angluin, “Inductive Inference of Formal Languages from Positive Data”, *Inform. and Control*, pp. 117-135, 1980.
- [3] D. Angluin, “Inference of Reversible Languages”, *Journal of the ACM*,. 29(3), pp. 741-765, 1982.
- [4] J. Berstel, *Transductions and Context Free Languages*, Teubner, 1979.

- [5] A.W. Biermann, J.A. Feldman, "On the synthesis of finite state machines from samples of their behavior", *IEEE Trans. on Computers*, C-21, pp. 592-597, 1972.
- [6] J.A. Brzozowski, "Languages of R-Trivial Monoids", *Journal of Computer and System Sciences*, 20, pp. 32-49, 1980.
- [7] J.A. Brzozowski, I. Simon, "Characterizations of locally testable events", *Discrete Mathematics*, 4, pp. 243-271, 1973.
- [8] R.G. Downey, M.R. Fellows, *Parametrized Complexity*, Springer, 1999.
- [9] S. Eilenberg, "Automata, Languages and Machines", *Vols. A and B*, Academic Press, 1976.
- [10] H. Fernau, "Identification of function distinguishable languages", *Theoretical Computer Science*, 290, pp. 1679-1711, 2003.
- [11] K.S. Fu, *Syntactic Pattern Recognition*, Prentice Hall, 1982.
- [12] P. García, J. Ruiz, "Right and left Locally testable languages", *Theoretical Computer Science*, 246(1-2), pp. 253-264, 2000.
- [13] P. García P, J. Ruiz, M. Vazquez de Parga, "Bilateral locally testable languages", *Theoretical Computer Science*, 299, pp. 775-783, 2003.
- [14] P. García, E. Vidal, "Inference of k -Testable languages in the Strict Sense and Applications to Syntactic Pattern Recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12/9, pp. 920-925, 1990.
- [15] E.M. Gold, "Language identification in the limit. Information and Control", 10, pp. 447-474, 1967.
- [16] E.M. Gold, "Complexity of Automaton Identification from Given Data", *Information and Control*, 37, 302-320, 1978.
- [17] R.C. González, M.G. Thomason, *Syntactic Pattern Recognition: An Introduction*, Addison Wesley, 1978.
- [18] C. de la Higuera, J. Oncina, E. Vidal, "Data dependent vs. data independent algorithms. In Grammatical Inference: Learning Syntax from Sentences", L. Miclet and C. de la Higuera, eds., *Lecture Notes in Artificial Intelligence*, 1147, Springer-Verlag, pp. 313-325, 1996.
- [19] J. Hopcroft, J. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.
- [20] R. König, "Reduction algorithms for some classes of aperiodic monoids", *R.A.I.R.O. Theoretical Informatics*, 19, 3, pp. 233-260, 1985.
- [21] M. Lothaire, *Combinatorics on words*, Addison-Wesley, 1983.

- [22] R. McNaughton, S. Papert, *Counter-Free Automata*, M.I.T. Press, 1971.
- [23] J. Pin, *Varieties of formal languages*, Plenum, 1986.
- [24] V. Radhakrisnan, G. Nagaraja, “Inference of regular grammars via skeletons”, *IEEE Trans. System, Man, and Cybernetics*, SCM-17, pp. 982-992, 1987.
- [25] J. Ruiz, P. García, “Learning k -piecewise testable languages from positive data. In Grammatical Inference: Learning Syntax from Sentences”, L. Miclet and C. de la Higuera eds., *Lecture Notes in Artificial intelligence*, 1147, Springer-Verlag, pp. 203-210, 1996.
- [26] I. Simon, “Piecewise testable events”, *Lecture Notes in Computer Science*, 33, Springer-Verlag, 1980.
- [27] Y. Zalcstein, “Locally testable languages”, *Journal of Computer and System Sciences*, 6, pp. 151-167. 1972.