Learning Context-Sensitive Languages from Linear Structural Information[⋆]

José M. Sempere

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
jsempere@dsic.upv.es

Abstract. In this work we propose a method to infer context-sensitive languages from positive structural examples produced by linear grammars. Our approach is based on a representation theorem induced by two operations over strings: duplication and reversal. The inference method produces an acceptor device which is an unconventional model of computation based on biomolecules (*DNA computing*). We prove that a subclass of context-sensitive languages can be inferred by using the representation result in combination with reductions from linear languages to k-testable in the strict sense regular languages.

Keywords: context-sensitive languages, Watson-Crick finite automata, linear languages, k-testable languages, identifiability from positive structural data.

1 Introduction

In the recent times, an unconventional theory of computation based on some biomolecules behavior has been proposed as the research area of DNA computing [9]. In a wide point of view, DNA computing deals with the capacities of DNA molecules to make (universal) computations. So, different models have been proposed in the framework of the formal language theory with some ingredients of the DNA features in order to process strings. We can mention Watson-Crick finite automata, sticker systems, splicing systems, Insertion-Deletion systems, among others. A profound study of these new models has pointed out its capacity to characterize language classes from Chomsky's hierarchy and new language classes which are related to the previous ones. In addition, these new models have provided a new look to the formal language theory in the sense that they have provided new operations over strings (splicing, duplication, twin shuffles, etc.) and new representations for the languages (i.e. circular strings or double strings). A comprehensive reference in this field is [9].

In this work, we will work with a DNA based computing model, the Watson-Crick finite automaton (WKFA) [3]. It has been proved that this model is able

[⋆] Work supported by the Spanish Ministerio de Educación y Ciencia under project TIN2007-60769.

to recognize context-sensitive languages. In addition, the languages accepted by WKFA can be obtained as the intersection between linear languages and even linear ones. So, given that these language families have been widely studied in the framework of Grammatical Inference we can take some advantages of previous results in order to learn efficiently some language classes which are characterized by restricted versions of WKFA. Observe that this approach enables the inference of new language classes which contains non-trivial context-sensitive languages.

The structure of this work is as follows: First we will give basic definitions and we will fix the notation related to formal language theory and some aspects of DNA computing used in the sequel. In section 3, we will propose an algorithm to learn language classes characterized by some restricted versions of the WKFA. We will prove the identifiability in the limit of these new classes based on the reducibility of this problem to previously solved ones. We will generalize our results by providing a learning scheme for different language classes. Finally, we will show our conclusions and we will provide some future research guidelines.

2 Basic Concepts and Notation

In this section, we will provide some concepts from formal language theory and DNA computing models. We suggest the books [9] and [10] to the reader.

Formal Languages

An alphabet Σ is a finite non-empty set of elements named symbols. A string defined over Σ is a finite ordered sequence of symbols from Σ. The infinite set of all the strings defined over Σ will be denoted by Σ^*. Given a string $x \in \Sigma^*$ we will denote its length by $|x|$. The set of strings defined over Σ with length equals to (less than) k will be denoted by Σ^k ($\Sigma^{\leq k}$). The empty string will be denoted by λ and Σ^+ will denote $\Sigma^* - \{\lambda\}$. Given a string x we will denote by x^r the reversal string of x. A language L defined over Σ is a set of strings from Σ^*.

A grammar is a construct $G = (N, \Sigma, P, S)$ where N and Σ are the alphabets of auxiliary and terminal symbols with $N \cap \Sigma = \emptyset$, $S \in N$ is the *axiom* of the grammar and P is a finite set of productions in the form $\alpha \to \beta$. We will say that w_1 directly derives to w_2, and we will denote it by $w_1 \underset{G}{\Rightarrow} w_2$ if $w_1 = u\alpha v$, $w_2 = u\beta v$ and $\alpha \to \beta \in P$. We will denote by $\overset{*}{\underset{G}{\Rightarrow}}$ the reflexive and transitive closure of $\underset{G}{\Rightarrow}$. The language of the grammar is denoted by $L(G)$ and it is the set of terminal strings that can be obtained from S by applying symbol substitutions according to P. So, $L(G) = \{w \in \Sigma^* : S \overset{*}{\underset{G}{\Rightarrow}} w\}$.

We will say that a grammar $G = (N, \Sigma, P, S)$ is *right linear* (regular) if every production in P is in the form $A \to uB$ or $A \to w$ with $A, B \in N$ and $u, w \in \Sigma^*$. The class of languages generated by right linear grammars is the class of regular languages and will be denoted by $\mathcal{REG}$. We will say that a grammar $G = (N, \Sigma, P, S)$ is *linear* if every production in P is in the form $A \to uBv$ or $A \to w$ with $A, B \in N$ and $u, v, w \in \Sigma^*$. The class of languages generated by linear grammars will be denoted by $\mathcal{LIN}$. We will say that a grammar $G = (N, \Sigma, P, S)$ is *even linear* if every production in P is in the form $A \to uBv$ or $A \to w$ with $A, B \in N$, $u, v, w \in \Sigma^*$ and $|u| = |v|$. The class of languages generated by even linear grammars will be denoted by $\mathcal{ELIN}$. We will say that a grammar $G = (N, \Sigma, P, S)$ is context-free if every production in P is in the form $A \to w$ with $A \in N$ and $w \in (\Sigma \cup N)^*$. The class of languages generated by context-free grammars will be denoted by $\mathcal{CF}$.

A context-sensitive grammar is a grammar $G = (N, \Sigma, P, S)$ where every production in P is in the form $\alpha A \beta \to \alpha\omega\beta$ with $\alpha, \beta \in (N \cup \Sigma)^*$, $\omega \in (N \cup \Sigma)^+$ and $A \in N$. If $S \to \lambda$ then S does not appear in the right side of any other production in P. The class of the languages generated by context-sensitive grammars will be denoted by $\mathcal{CS}$. A well-known result from formal language theory is the inclusions $\mathcal{REG} \subset \mathcal{ELIN} \subset \mathcal{LIN} \subset \mathcal{CF} \subset \mathcal{CS}$.

A homomorphism h is defined as a mapping $h : \Sigma \to \Gamma^*$ where Σ and Γ are alphabets. We can extend the definition of homomorphisms over strings as $h(\lambda) = \lambda$ and $h(ax) = h(a)h(x)$ with $a \in \Sigma$ and $x \in \Sigma^*$. The homomorphism over a language $L \subseteq \Sigma^*$ is defined as $h(L) = \{h(x) : x \in L\}$.

***Stickers*, Molecules and Watson-Crick Finite Automata**

Given an alphabet $\Sigma = \{a_1, \cdots, a_n\}$, we will use the symmetric (and injective) relation of complementarity $\rho \subseteq \Sigma \times \Sigma$. For any string $x \in \Sigma^*$, we will denote by $\rho(x)$ the string obtained by substituting the symbol a in x by the symbol b such that $(a, b) \in \rho$ (remember that ρ is injective) with $\rho(\lambda) = \lambda$.

Given an alphabet Σ, a *sticker* over Σ will be the pair (x, y) such that $x = x_1vx_2$, $y = y_1wy_2$ with $x, y \in \Sigma^*$ and $\rho(v) = w$. The sticker (x, y) will be denoted by $\begin{pmatrix} x \\ y \end{pmatrix}$. A sticker $\begin{pmatrix} x \\ y \end{pmatrix}$ will be a *molecule* if $|x| = |y|$ and $\rho(x) = y$, and it will be denoted by $\begin{bmatrix} x \\ y \end{bmatrix}$. Obviously, any sticker $\begin{pmatrix} x \\ y \end{pmatrix}$ or molecule $\begin{bmatrix} x \\ y \end{bmatrix}$ can be represented by $x\#y^r$ where $\# \notin \Sigma$.

There have been different computational models and generating devices that use stickers and molecules to define formal languages. We will fix our attention to the acceptor models named *Watson-Crick finite automata*. A Watson-Crick finite automaton (WKFA) [3] is a good example of how DNA biological properties can be adapted to propose computation models in the framework of DNA computing. WK finite automata work with double strings inspired by double-stranded molecules with a complementary relation between elements, that is, the classical complementary relation between DNA nucleotides A-T and C-G. So, a WK finite automaton has an input double tape which is organized into upper and lower cells, it has two tape heads that access to the upper and lower cells and that can move independently and a finite control which holds a state of the machine during its computation. In addition, the automaton has a transition function that guides the movements of the machine. The machine works as follows: initially a sticker or molecule is placed in the double tape (i.e. the lower strand in the lower tape and the upper strand in the upper tape), the tape heads are placed at the beginning of the tape (i.e. pointing out to the first symbol of every tape) and the finite control holds an initial state. Then, the machine starts to apply the transition function which is nondeterministic. Every time that the machine applies a transition then the state in the finite control changes and the tape heads advance one cell to the right or stay at the same cell independently (i.e. maybe a transition only moves the upper head or only the lower head or it can move both heads). The machine stops when no transition can be applied or the input sticker or molecule have been completely processed. Observe that, in this case, the machine can halt into an acceptation state. The criterium which is imposed to accept a sticker is that it has been completely processed, the machine has stopped within an acceptation state and the sticker is a molecule.

Formally, an *arbitrary* WK finite automaton is defined by the tuple $M = (V, \rho, Q, s_0, F, \delta)$, where Q and V are disjoint alphabets (states and symbols), s_0 is the initial state, $F \subseteq Q$ is a set of final states and the finitely defined function $\delta : Q \times \binom{V^*}{V^*} \rightarrow \mathcal{P}(Q)$ (which denotes the power set of Q, that is the set of all possible subsets of Q). Furthermore, we can impose a normal form such that for every transition $q \in \delta(q, \binom{x_1}{x_2})$ then $|x_1x_2| = 1$. This normal form defines the so called 1-*limited* WK finite automata and they were proved to be equivalent to arbitrary ones [3].

An instantaneous description of the WK finite automaton will be denoted by $\binom{x_1}{y_1} q \binom{x_2}{y_2}$, where $\binom{x_1}{y_1}$ is the part of the sticker which has been processed, q is the state of the finite control and $\binom{x_2}{y_2}$ is the rest of the sticker to be processed. We can relate instantaneous descriptions as follows: $\binom{x_1}{y_1} q \binom{x_2}{y_2} \Rightarrow \binom{x_1v_1}{y_1w_1} p \binom{v_2}{w_2}$ if $x_2 = v_1v_2$, $y_2 = w_1w_2$ and $p \in \delta(q, \binom{v_1}{w_1})$. We will denote the reflexive and transitive closure of $\Rightarrow$ by $\overset{*}{\Rightarrow}$.

Given an arbitrary WK finite automaton $M = (V, \rho, Q, s_0, F, \delta)$, the language of molecules accepted by M will be defined by the set $L_m(M) = \{\begin{bmatrix} x \\ y \end{bmatrix} : s_0 \begin{bmatrix} x \\ y \end{bmatrix} \overset{*}{\Rightarrow} \begin{bmatrix} x \\ y \end{bmatrix} p \text{ with } p \in F\}$. The upper strand language accepted by M will be defined by the set $L_u(M) = \{x : s_0 \begin{bmatrix} x \\ y \end{bmatrix} \overset{*}{\Rightarrow} \begin{bmatrix} x \\ y \end{bmatrix} p \text{ with } p \in F\}$. The family of upper languages accepted by arbitrary Watson-Crick finite automata will be denoted by $\mathcal{AWK}_u$, and it has been proved that $\mathcal{AWK}_u \subset \mathcal{CS}$. That is, WKFA accept context-sensitive languages in the upper strand. In addition, it has been proved that context-free languages and $\mathcal{AWK}_u$ are disjoint classes of languages [7].

In a previous work, we proved that the languages accepted by arbitrary WKFA can be represented by operations over linear and even linear languages [13]. The main theorem that supports this statement is the following

Theorem 1. *(Sempere, [13]) Let $M = (V, \rho, Q, s_0, F, \delta)$ be an arbitrary Watson-Crick finite automaton. Then, there exists a linear language L_1 and an even linear language L_2 such that $L_m(M) = L_1 \cap L_2$.*

In [13], we provided an algorithm to construct a linear grammar G_1 such that $L(G_1) = L_1$ and an even linear grammar G_2 such that $L(G_2) = L_2$. It works as follows: First, we can construct the grammar $G_1 = (N, V \cup \{\#\}, P, s_0)$ where $N = Q$, s_0 is the axiom of the grammar and P is defined as follows

- If $q \in F$ then $q \rightarrow \# \in P$
- If $p \in \delta(q, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix})$ then $q \rightarrow x_1\ p\ x_2^r \in P$.

The language L_2 can be defined by the grammar $G_2 = (\{S\}, V \cup \{\#\}, P, S)$ where P is defined as follows

- $S \rightarrow \# \in P$
- For every pair of symbols $a, b \in V$, such that $(a, b) \in \rho$, $S \rightarrow aSb \in P$

In order to characterize the upper strand language we provided the following result

Corollary 1. *(Sempere, [13]) Let $M = (V, \rho, Q, s_0, F, \delta)$ be an arbitrary WK finite automaton. Then $L_u(M)$ can be expressed as $g(h^{-1}(L_1 \cap L_2) \cap R)$ with L_1 being a linear language, L_2 an even linear language, R a regular language and g and h two morphisms.*

Observe that the last result can be obtained by using a morphism $h : V \cup V' \cup \{\#\} \rightarrow V \cup \{\#\}$, defined as $h(a) = h(a') = a$ for every $a \in V$ where $V' = \{a' : a \in V\}$, and $h(\#) = \#$. Then, $R = V^*\#V'^*$ and g is a morphism defined as $g(\#) = \lambda$, $g(a) = a$ and $g(a') = \lambda$ for every $a \in V$ and every $a' \in V'$.

Example 1. Let $M = (V, \rho, Q, q_0, F, \delta)$ be a WKFA where $V = \{a, b, c\}$, $\rho = \{(a, a), (b, b), (c, c)\}$, $F = \{q_f\}$ and δ is defined as follows:

$$\begin{array}{lll}
\delta(q_0, \begin{pmatrix} a \\ \lambda \end{pmatrix}) = \{q_a\} & \delta(q_a, \begin{pmatrix} a \\ \lambda \end{pmatrix}) = \{q_a\} & \delta(q_a, \begin{pmatrix} b \\ a \end{pmatrix}) = \{q_b\} \\
\delta(q_b, \begin{pmatrix} b \\ a \end{pmatrix}) = \{q_b\} & \delta(q_b, \begin{pmatrix} c \\ b \end{pmatrix}) = \{q_c\} & \delta(q_c, \begin{pmatrix} c \\ b \end{pmatrix}) = \{q_c\} \\
\delta(q_c, \begin{pmatrix} \lambda \\ c \end{pmatrix}) = \{q_f\} & \delta(q_f, \begin{pmatrix} \lambda \\ c \end{pmatrix}) = \{q_f\} &
\end{array}$$

It can be easily proved that $L_u(M) = \{a^n b^n c^n : n \geq 1\}$. Then the corresponding linear grammar associated with M is the following one, which we will name G_1:

$$\begin{array}{lll}
q_0 \rightarrow aq_a & q_a \rightarrow aq_a \mid bq_b a & q_b \rightarrow bq_b a \mid cq_c b \\
q_c \rightarrow cq_c b \mid q_f c & q_f \rightarrow q_f c \mid \# &
\end{array}$$

The even linear grammar associated with ρ is trivially defined by the rules $S \rightarrow aSa \mid bSb \mid cSc \mid \#$ which we will name G_2. Observe that $L_m(M) = L(G_1) \cap L(G_2)$ while the language $L_u(M)$ is obtained from the strings in $L(G_1) \cap L(G_2)$ by taking only the complete prefixes up to the # symbol. This last operation can be performed by applying Corollary 1.

From the previous results, it can be proved that upper strand languages accepted by WKFA can be reduced to regular languages. So, we introduced in the WKFA model well-known features such as *k-testability* [14] and *reversibility* [15]. In addition, we establish a way to obtain *regular-like* expressions from WKFA [16].

Local Testability

Here, we will introduce the definition of local testability and local testability in the strict sense. For any string $x \in \Sigma^*$ and any integer value $k > 0$, the testability vector $v_k(x)$ is defined by the tuple $(i_k(x), t_k(x), f_k(x))$ where

$$i_k(x) = \begin{cases} x, & \text{if } |x| < k \\ u : x = uv, |u| = k-1 & \text{if } |x| \geq k \end{cases}$$

$$f_k(x) = \begin{cases} x, & \text{if } |x| < k \\ v : x = uv, |v| = k-1 & \text{if } |x| \geq k \end{cases}$$

$$t_k(x) = \{v : x = uvw, u, w \in \Sigma^* \wedge |v| = k\}$$

We will define the equivalence relation $\equiv_k$ in $\Sigma^* \times \Sigma^*$ as $x \equiv_k y$ iff $v_k(x) = v_k(y)$. It has been proved in [6] that $\equiv_k$ is a finite index relation and that $\equiv_k$ covers $\equiv_{k+1}$.

So, we will say that any language L is k-testable iff it is defined as the union of some equivalence classes of $\equiv_k$. In addition, L is local testable iff it is k-testable for any integer value $k > 0$. The family of k-testable languages will be denoted by $k - \mathcal{LT}$ while $\mathcal{LT}$ will denote the class of testable languages.

A different kind of testability is the so called testability in the strict sense which was again proposed in [6]. Here, for any alphabet Σ we will take the sets $I_k, F_k \subseteq \Sigma^{\leq k-1}$ and $T_k \subseteq \Sigma^k$. Then, a language L is said to be k-testable in the strict sense if the following equation holds

$$L \cap \Sigma^{k-1}\Sigma^* = (I_k\Sigma^*) \cap (\Sigma^* F_k) - (\Sigma^* T_k \Sigma^*).$$

Observe that, according to the last equation, any word in L with length greater than or equals to $k-1$ begins with a segment in I_k, ends with a segment in F_k and has no segment from T_k. Any language L is locally testable in the strict sense iff it is k-testable in the strict sense for any $k > 0$. The family of k-testable languages in the strict sense will be denoted by $k - \mathcal{LTSS}$ while $\mathcal{LTSS}$ will denote the class of locally testable languages in the strict sense.

It has been proved that $k - \mathcal{LT}$ is the boolean closure of $k - \mathcal{LTSS}$ [22]. In addition, both classes $k - \mathcal{LT}$ and $k - \mathcal{LTSS}$ are subclasses of $\mathcal{REG}$.

3 Learning Watson-Crick Finite Automata from Positive Structural Data

In order to learn formal languages accepted by (restricted) WKFA we will use two operations to represent such languages: duplication and reversal. Once we fix this representation for formal languages, we can infer restricted versions of WKFA in order to recognize languages in the upper strand. Our method is a reduction technique based on the linear and even-linear languages used in Theorem 1 and Corollary 1. Observe, that a similar approach was employed in [8] where the authors reduced languages accepted by WKFA to regular languages by using a technique which is different from the one in [13].

The relation of complementarity that we will use is the trivial identity relation. That is, $(\forall a \in \Sigma)\ (a, a) \in \rho$. The duplication of a string is defined as $duplicate(x) = x\#x^r$. The extension over a set S is trivially defined as $duplicate(S) = \{duplicate(x) : x \in S\}$. Observe that this is a linear time operation.

Our main task is to learn unknown linear languages from the strings obtained by duplication. Here, we can adopt different solutions in order to carry out the learning task. First, we can use learning algorithms for subclasses of linear languages as in [1,2,5]. Another option could be the use of structural information as in [5]. Observe that the use of structural information has been widely accepted as an information protocol since Sakakibara's works about learning context-free languages [11,12]. We will use structural information in this work in order to avoid the use of complete data (negative and positive strings). Nevertheless, the use of any algorithm to infer linear languages from different information protocols could be easily introduced in our learning scheme.

So, the information given to the learning algorithm will not be the duplicated strings but the structural information associated with them, according to an unknown WKFA.

Observe that the structural information from linear grammars allows the transformation to even linear ones as was shown in a previous work [17]. We can introduce the reduction in a formal manner as follows: Let us take the linear structured string w defined over the alphabet Σ, then we can obtain an even linear structure from w by applying the function $ell(w)$ with the following rules

1. $ell((\lambda)) = \lambda$
2. $ell((a)) = a$ for every $a \in \Sigma$
3. $ell((a(x))) = a \cdot ell((x)) \cdot *$ for every $a \in \Sigma$ and x a structural string over Σ
4. $ell(((x)a)) = * \cdot ell((x)) \cdot a$ for every $a \in \Sigma$ and x a structural string over Σ

The last transformation can be easily extended over sets of structural strings.

The even linear languages can be reduced to regular ones by using control sets as was proved in [21]. In addition, a different solution was proposed in [18] by using the σ reduction over strings in Σ^* that we will define as follows:

1. $\sigma(axb) = [ab]\sigma(x)$ with $a, b \in \Sigma$ and $x \in \Sigma^*$
2. $\sigma(a) = [a]$ with $a \in \Sigma \cup \{\lambda\}$

It has been proved that if L is an even linear language then $\sigma(L)$ is regular [18]. From the last reduction, we proposed a learning algorithm in [19] that could learn even linear languages with k-testability based on a previous algorithm to learn k-testable languages in the strict sense from only positive data proposed by García *et al.* [4]. We will refer to that algorithm as `KTSS`.

In [14], we defined WKFA with local testability (in the strict sense). The basic concept is that the finite automaton obtained from the WK finite automaton by using the $ell(\cdot)$ and the σ operations defines a regular language. So, if the obtained regular language is locally testable (in the strict sense) then, the molecule language accepted by the WK finite automaton will be locally testable (in the strict sense) in a wide sense. The class of languages accepted by k-testable (in the strict sense) WKFA in the upper strand will be denoted by $\mathcal{AWK}_u^{\mathcal{KLT}(\mathcal{SS})}$.

Finally, you can observe that the σ reduction and the structural transformation $ell(\cdot)$ that we have proposed before can be easily reversed in order to obtain the initial language.

Now, we can mix up all these operations in order to learn a restricted subclass of WKFA which are still able of recognizing non-trivial context-sensitive languages. The proposed algorithm is shown as Algorithm 1.

Algorithm 1. An algorithm to learn $\mathcal{AWK}_u^{\mathcal{KLTSS}}$ languages from structural information

Input: A finite sample of linear structural duplicated strings S defined over Σ
Output: A WKFA A such that $S^+ \subseteq L_u(A)$
Method:

```
1. $S_{ell} = ell(S)$
2. $S_\sigma = \sigma(S_{ell})$
3. $A_r$=KTSS($S_\sigma$)
4. $G_{ell} = \sigma^{-1}(A_r)$
5. $G_{lin} = ell^{-1}(G_{ell})$
6. $A = AFWK(G_{lin})$ where $\rho = \{(a,a) : a \in \Sigma\}$
7. Return($A$)
```

EndMethod.

The proposed algorithm is able to infer WKFA from positive structural information sample only. The restrictions over the learned WKFA are the following:

1. The linear grammar associated with the WK finite automaton generates structured strings according to S
2. The set S^+ is associated to the set S. Here, S^+ is defined by the strings obtained from S by taking only the upper strand (as an application of Corollary 1).
3. The grammar obtained by reducing the corresponding linear grammar of the WK finite automaton to the regular one is k-testable in the strict sense. Alternatively, we can say that the WK finite automaton is k-testable in the strict sense as shown in [14].

4. The operation $A = AFWK(G_{lin})$ takes a linear grammar and obtains a WK finite automaton by applying the Theorem 1.

We can easily prove that there exists context-sensitive languages which are not context-free and that can be accepted in the upper strand by a WK finite automaton with the previous restrictions.

Example 2. Let us take the language $L = \{a^n b^n c^n : n \geq 1\}$ which can be accepted by the WK finite automaton shown in the Example 1. A positive structural information associated with the WK finite automaton could be the following

$$S = \{(a(b(c((\#)c)b)a)), (a(a(b(b(c(c(((\#)c)c)b)b)a)a)))\}$$

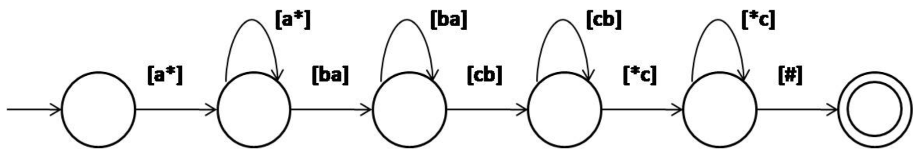
Then, by transforming the linear structures in even linear ones we obtain

$$S_{ell} = \{abc * \#cba*, aabbcc * *\#ccbbaa * *\}$$

Then, by applying the σ transformation to S_{ell} we obtain

$$S_\sigma = \{[a*][ba][cb][*c][\#], [a*][a*][ba][ba][cb][cb][*c][*c][\#]\}$$

A k-testable language in the strict sense inferred from the previous sample, with $k = 2$, by applying the learning algorithm `KTSS`, is the following one



From the last finite automaton we can obtain an even linear grammar by applying the σ^{-1} transformation. The corresponding even linear grammar is the following

$S \rightarrow aA*$ $A \rightarrow aA* \mid bBa$ $B \rightarrow bBa \mid cCb$
$C \rightarrow cCb \mid *Dc$ $D \rightarrow *Dc \mid \#$

From the last even linear grammar we can obtain a linear one, by applying the homomorphism $g(a) = a$, $g(b) = b$ $g(c) = c$, $g(\#) = \#$ and $g(*) = \lambda$. The linear grammar obtained from g is the following one

$S \rightarrow aA$ $A \rightarrow aA \mid bBa$ $B \rightarrow bBa \mid cCb$
$C \rightarrow cCb \mid Dc$ $D \rightarrow Dc \mid \#$

Observe that the WK finite automaton associated with the previous grammar is the one shown in the Example 1. In addition, the complementarity relation is obtained again from the input sample as $\rho = \{(a, a), (b, b), (c, c)\}$. So, $L \in \mathcal{AWK}_u^{\mathcal{KLTSS}}$.

The efficiency of the proposed method is shown in the following result.

Proposition 1. *The proposed Algorithm 1 runs in polynomial time with the size of the input sample S.*

Proof. It can be trivially proved that the Algorithm 1 runs in polynomial time. The structural transformation $ell(\cdot)$ is linear with the size of the string. The application of the σ transformation is linear again. The application of the algorithm KTSS is polynomial time [4]. All the operations used to obtain the WK finite automaton are polynomial time given to the fact that they are the application of the σ^{-1} and the $ell^{-1}(\cdot)$ operations. □

Finally, the identifiability of a non-trivial context-sensitive language class in the limit is shown as follows

Proposition 2. $\mathcal{AWK}_u^{\mathcal{KLTSS}}$ *is identifiable in the limit from only positive structural information.*

Proof. The identification in the limit comes from the convergence result of the algorithm KTSS exposed in [4]. So, if we apply the Algorithm 1, the identifiability in the limit of the class $\mathcal{AWK}_u^{\mathcal{KLTSS}}$ is guaranteed. □

Generalizing the Learning Scheme

In the Algorithm 1, we have used the learning algorithm for k-testable languages in the strict sense, KTSS. Nevertheless, any learning algorithm for a given subclass of regular languages could be fruitful in proposing new learning algorithms for different restrictions of WKFA. So, a generalization of Algorithm 1 is proposed as Algorithm 2 which is a learning scheme to take advantages of the previously proposed reductions from WKFA to regular languages.

In the Algorithm 2, the method LearningRegPos refers to any learning algorithm that works with only positive data and obtains a finite automaton or a representation for a regular language. Observe that the learning algorithms referred

Algorithm 2. An algorithm to learn different families of $\mathcal{AWK}_u$ languages from structural information

```
Input: A finite sample of linear structural duplicated strings $S$ defined over $\Sigma$
Output: A WKFA $A$ such that $S^+ \subseteq L_u(A)$
Method:

  1. $S_{ell} = ell(S)$
  2. $S_\sigma = \sigma(S_{ell})$
  3. $A_r$=LearningRegPos($S_\sigma$)
  4. $G_{ell} = \sigma^{-1}(A_r)$
  5. $G_{lin} = ell^{-1}(G_{ell})$
  6. $A = AFWK(G_{lin})$ where $\rho = \{(a,a) : a \in \Sigma\}$
  7. Return($A$)

EndMethod.
```

as `LearningRegPos`, characterize different subclasses of regular languages (i.e. k-*reversible*, *terminal distinguishable*, *function distinguishable* in general, different (*positive*) *varieties* of regular languages, etc.). Furthermore, if we change the information protocol, and we provide positive structural information together with negative data as input, then we can apply other learning algorithms that work with complete data to infer regular languages.

4 Conclusions and Future Work

We have proposed an efficient method to infer a subclass of context-sensitive languages from linear structured positive strings. The method uses a computational device that has been previously defined in the framework of DNA computing. We think that this emerging area provides models, operations and new looks for the proposal of new learning algorithm that would enrich the map of efficiently learnable languages.

In this work, we have used structural positive information for the inference of restricted WKFA which are able of recognize non-trivial context-sensitive languages. It is an open question whether the use of different learning algorithms for some subclasses of regular languages still holds the inclusion of non-trivial context-sensitive languages in the upper strand or not. Actually, this issue is under study.

Another work that we are carrying out in the present is the application of this method to the processing of *biosequences*. Observe that in this framework the use of duplication strings comes in a natural way (i.e. DNA strings) and the availability of structural information comes from the domain task in an easy way (i.e. location of promoters, genes, motifs, etc.).

References

1. Calera-Rubio, J., Oncina, J.: Identifying Left-Right Deterministic Linear Languages. In: Paliouras, G., Sakakibara, Y. (eds.) ICGI 2004. LNCS (LNAI), vol. 3264, pp. 283–284. Springer, Heidelberg (2004)
2. de la Higuera, C., Oncina, J.: Inferring Deterministic Linear Languages. In: Kivinen, J., Sloan, R.H. (eds.) COLT 2002. LNCS (LNAI), vol. 2375, pp. 185–200. Springer, Heidelberg (2002)
3. Freund, R., Păun, G., Rozenberg, G., Salomaa, A.: Watson-Crick finite automata. In: Proceedings of DNA Based Computers III DIMACS Workshop (June 1997), pp. 297–327. The American Mathematical Society (1999)
4. García, P., Vidal, E., Oncina, J.: Learning locally testable languages in the strict sense. In: Proceedings of the First International Workshop on Algorithmic Learning Theory. Japanese Society for Artificial Intelligence, pp. 325–338 (1990)
5. Laxminarayana, J.A., Sempere, J.M., Nagaraja, G.: Learning Distinguishable Linear Grammars from Positive Data. In: Paliouras, G., Sakakibara, Y. (eds.) ICGI 2004. LNCS (LNAI), vol. 3264, pp. 279–280. Springer, Heidelberg (2004)
6. McNaughton, R., Papert, S.: Counter-free automata. MIT Press, Cambridge (1971)

7. Okawa, S., Hirose, S.: The Relations among Watson-Crick Automata and Their Relations to Context-Free Languages. IEICE Transactions on Information and Systems E89-D(10), 2591–2599 (2006)
8. Onodera, K., Yokomori, T.: Doubler and linearizer: an approach toward unified theory for molecular computing based on DNA complementarity. Natural Computing 7, 125–143 (2008)
9. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing. New computing paradigms. Springer, Heidelberg (1998)
10. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. 1. Springer, Heidelberg (1997)
11. Sakakibara, Y.: Learning context-free grammars from structural data in polynomial time. Theoretical Computer Science 76(2-3), 223–242 (1990)
12. Sakakibara, Y.: Efficient learning of context-free grammars from positive structural examples. Information and Computation 97(1), 23–60 (1992)
13. Sempere, J.M.: A representation theorem for languages accepted by Watson-Crick finite automata. Bulletin of the EATCS 83, 187–191 (2004)
14. Sempere, J.M.: On Local Testability in Watson-Crick Finite Automata. In: Vaszil, G. (ed.) Proceedings of the International Workshop on Automata for Cellular and Molecular Computing, pp. 120–128 (2007)
15. Sempere, J.M.: Exploring regular reversibility in Watson-Crick finite automata. In: Proceedings of the 13th International Symposium on Artificial Life and Robotics, pp. 505–509. AROB (2008)
16. Sempere, J.M.: Sticker expressions. In: Goel, A., Simmel, F.C., Sosik, P. (eds.) Proceedings of the 14th International Meeting on DNA Computing, pp. 200–201 (2008)
17. Sempere, J.M., Fos, A.: Learning Linear Grammars from Structural Information. In: Miclet, L., de la Higuera, C. (eds.) ICGI 1996. LNCS, vol. 1147, pp. 126–133. Springer, Heidelberg (1996)
18. Sempere, J.M., García, P.: A Characterization of Even Linear Languages and its Application to the Learning Problem. In: Carrasco, R.C., Oncina, J. (eds.) ICGI 1994. LNCS, vol. 862, pp. 38–44. Springer, Heidelberg (1994)
19. Sempere, J.M., García, P.: Learning Locally Testable Even Linear Languages from Positive Data. In: Adriaans, P.W., Fernau, H., van Zaanen, M. (eds.) ICGI 2002. LNCS (LNAI), vol. 2484, pp. 225–236. Springer, Heidelberg (2002)
20. Sempere, J.M., Nagaraja, G.: Learning a Subclass of Linear Languages from Positive Structural Information. In: Honavar, V.G., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433, pp. 162–174. Springer, Heidelberg (1998)
21. Takada, Y.: Grammatical inference for even linear languages based on control sets. Information Processing Letters 28(4), 193–199 (1988)
22. Zalcstein, Y.: Locally Testable Languages. Journal of Computer and System Sciences 6, 151–167 (1972)