Left/Right Deterministic Linear Languages Identification*

Jorge Calera-Rubio and Jose Oncina
Universidad de Alicante. Departamento de Lenguajes y Sistemas Informáticos.
Apt.99. E-03080 Alicante, Spain.
{calera, oncina}@dlsi.ua.es

Abstract

Left deterministic linear languages are a subclass of context free languages that includes all regular languages. Recently was proposed an algorithm to identify in the limit with polynomial time and data such class of languages. It was also pointed that a symmetric class, right deterministic linear languages, is also identifiable in the limit from polynomial time and data. In this paper we show that the class of the Left-Right Deterministic Languages formed by the union of both classes is also identifiable. The resulting class is the largest one for which this type of results has been obtained so far.

In this paper we introduce the notion of n-negative characteristic sample, that is a sample that forces an inference algorithm to output a hypothesis of size bigger than n when strings from a non identifiable language are provided.

Keywords: example-based learning, learning context-free languages

1 Introduction

Over the time, diverse paradigms has been proposed to formalize when a learning process is successful. One of those paradigms is the *identification in the limit* [1]. In the identification in the limit paradigm, the learning process is seen as an infinite process in which an algorithm receives items of information about a target model. Each time an item is received the algorithm should propose a hypothesis. In order to be successful the algorithm should assure that, after receiving a finite number of items, the hypothesis model is always equivalent to the target.

Unfortunately, this paradigm does not put any restriction on the resources the inference algorithm can use. Several criteria has been introduced to cover this gap [2]. In this paper we are going to use the criterion of *identification in the limit from polynomial time and data* introduced by de la Higuera [3]. This criterion requires the

*Work partially supported by Spanish CICyT though project TIC2003-08496-C04 and by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

existence of a polynomial size set of items such that, when provided to the algorithm along with other items, the algorithm should produce a hypothesis equivalent to the target.

In our case the models are formal languages and the items strings with labels indicating their belonging or not to the grammar.

The class of the *Linear Languages* is a subclass of the *Context Free* Grammars. This languages are produced by Context Free Grammars such that on the right hand side of the rules there is at most a nonterminal. Although it seems a quite simple class, it has been shown that even the question of saying if two linear languages are equivalent is undecidable. Then, the identification in the limit from polynomial time and data is impossible.

An important subclass of the linear languages are the *Left Deterministic Linear Languages* (LDLL) [4]. Those languages can be generated by *Left Deterministic Linear Grammars* that shares with the Deterministic Regular Grammar the property of knowing which is the next rule to use in the parsing of a string just by observing the leftmost terminal in the unparsed part of the string. $\{a^n b^n | n \geq 0\}$ and $\{a^m b^n c^n | m, n \geq 0\}$ are some examples of languages in the class, while $\{a^n b^n c^m | m, n \geq 0\}$ is not. This class includes the Regular Languages.

Unfortunately this class is not closed over the reversibility, that is not every language formed by the reversals of the strings in an LDLL are in LDLL. The class formed by the reversals of the languages in LDLL is called the *Right Deterministic Linear Languages* (RDLL) and are defined in a symmetrical way in which LDLL are defined. The regular languages, $\{a^n b^n | n \geq 0\}$ and $\{a^n b^n c^m | m, n \geq 0\}$ are examples of languages in the class.

Recently it was showed ([4]) that the class of the LDLL can be identified in the limit from polynomial time and data. Obviously, the class of the RDLL can also be identified just by reversing the strings before to introduce them on the LDLL inference algorithm.

In this paper we propose a new class of languages, the Left-Right Deterministic Languages (LRDLL), formed by the union of the LDLL and RDLL. We show that this class can be identified in the limit from polynomial time and data. The inference algorithm makes use of two inference algorithms, one for LDLL and other for RDLL. When a sample is given the main algorithm runs both inference algorithms and outputs the smaller of the hypothesis produced by the algorithms.

In order to show the identification in the limit from polynomial time and data, the concept of n-negative characteristic sample is introduced. That is a sample that forces an inference algorithm to output a hypothesis of size bigger than n when strings from a non identifiable language are provided.

The paper is organized as follows:

- Section 2 introduces the main notation used trough the paper, defines the classes of grammars object of this paper and defines the identification in the limit form polynomial time and data learning paradigm.
- Section 3 reviews the main properties of the LDLL and describes a learning algorithm for this class of languages.
- Section 4 describes the learning algorithm for the LRDLL, introduces the concept of n-negative characteristic set and it is used to show the identifiability in the limit from polynomial time and data.
- Section 5 concludes and propose new related open problems.

2 Definitions

2.1 Languages and Grammars

An alphabet Σ is a finite nonempty set of symbols. Σ^* denotes the set of all finite strings over Σ, $\Sigma^+ = \Sigma^* - \{\lambda\}$ where λ denotes the empty string. A language L over Σ is a subset of Σ^*. In the following, unless stated otherwise, symbols are indicated by $a, b, c \ldots$ and strings by $u, v, \ldots$. $\mathbb{N}$ is the set of non negative integers. The length of a string u will be denoted $|u|$, so $|\lambda| = 0$. Let I be a finite set of strings, $|I|$ denotes the number of strings in the set and $\|I\|$ denotes the total sum of the lengths of all strings in I.

Let $u, v \in \Sigma^*, u^{-1}v = w$ such that $v = uw$ (undefined if u is not a prefix of v) and $uv^{-1} = w$ such that $u = wv$ (undefined if v is not a suffix of u). Let L be a language and $u \in \Sigma^*$, $u^{-1}L = \{v : uv \in L\}$ and $Lu^{-1} = \{v : vu \in L\}$.

Let L be a language, the *prefix set* is $\Pr(L) = \{x : xy \in L\}$.The symmetrical difference between two languages L_1 and L_2 will be denoted $L_1 \ominus L_2$. The *longest common suffix* ($\mathrm{lcs}(L)$) of L is the longest string u such that $(Lu^{-1})u = L$.

Let u^R denote the reversal of the string u, the reversal of a string can be computed recursively as $(\lambda)^R = \lambda$ and $(ua)^R = au^R$. Let X be a set of strings $X^R = \{x^R : x \in X\}$.

Definition 1 (Context-free grammars). *A* context-free grammar *(CFG) G is a quadruple (Σ, V, P, S) where Σ is a finite alphabet (of terminal symbols), V is a finite alphabet (of variables or non-terminals), $P \subset V \times (\Sigma \cup V)^*$ is a finite set of production rules, and $S(\in V)$ is the axiom. We will denote $uTv \to uwv$ when $(T, w) \in P$.*

If there exists $u_0, \ldots, u_k$ such that $u_0 \to \cdots \to u_k$ we will write $u_0 \overset{k}{\to} u_k$. We denote by $L_G(T)$ the language $\{w \in \Sigma^* : T \overset{*}{\to} w\}$ and by $L(G)$ the language $\{w \in \Sigma^* : S \overset{*}{\to} w\}$. were $\overset{*}{\to}$ denotes the transitive, reflexive clousure of $\to$. Two grammars are equivalent if they generate the same language.

Let $G = (\Sigma, V, P, S)$ a CFG, the CFG $G^R = (\Sigma, V, P', S)$ is the reversal of G iff $(A, \alpha) \in P \iff (A, \alpha^R) \in P'$. Obviously, $x \in L(G) \iff x^R \in L(G^R)$.

Definition 2 (Linear grammars). *A context-free grammar* $G = (\Sigma, V, P, S)$ *is* linear *if* $P \subset V \times (\Sigma^* V \Sigma^* \cup \Sigma^*)$

We will be needing to speak of the *size* of a grammar. Without entering into a lengthy discussion, the size has to be a quantity polynomially linked with the number of bits needed to encode a grammar [2]. We will consider here the size of G denoted by $\|G\| = \sum_{(T,u)\in P}(|u| + 1)$.

2.2 Deterministic Linear Grammars

In [4] was introduced the class of the Left Deterministic Linear Grammars and Right Deterministic Linear Grammars as follows:

Definition 3 (Left Deterministic Linear Grammars). *A* Left Deterministic Linear Grammar *(LDLG)* $G = (\Sigma, V, P, S)$ *is a linear grammar where* $P \subset (V \times \Sigma V \Sigma^*) \cup (V \times \{\lambda\})$ *and*

$$\left.\begin{array}{l} \forall A \in V \\ \forall a \in \Sigma \\ \forall \alpha, \beta \in V\Sigma^* \end{array}\right. \qquad \left.\begin{array}{l} (A, a\alpha) \in P \\ (A, a\beta) \in P \end{array}\right\} \Rightarrow \alpha = \beta$$

Definition 4 (Right Deterministic Linear Grammars). *A* Right Deterministic Linear Grammar *(RDLG)* $G = (\Sigma, V, P, S)$ *is a linear grammar where* $P \subset (V \times \Sigma^* V \Sigma) \cup (V \times \{\lambda\})$ *and*

$$\left.\begin{array}{l} \forall A \in V \\ \forall a \in \Sigma \\ \forall \alpha, \beta \in \Sigma^* V \end{array}\right. \qquad \left.\begin{array}{l} (A, \alpha a) \in P \\ (A, \beta a) \in P \end{array}\right\} \Rightarrow \alpha = \beta$$

The languages generated by LDLG and RDLG are called Left Deterministic Linear Languages (LDLL) and Right Deterministic Linear Languages (RDLL) respectively.

Note that, RDLG = $(\text{LDLG})^R$ and then RDLL = $(\text{LDLL})^R$. The Deterministic Regular Grammars are a special case of the LDLG (RDLL) where the string that

appear on the rightmost (leftmost) part of the rules is λ. Then the LDLL and the RDLL include the regular languages. Both classes of languages include languages such as $\{a^n b^n | n \geq 0\}$ however, $\{a^m b^n c^n | m, n \geq 0\} \in$ LDLL but $\notin$ RDLL and, obviously, its reverse $\{c^n b^n a^m | m, n \geq 0\} \in$ RDLL but $\notin$ LDLL.

Finally, we are interested in deterministic linear grammars which can be LDLG or RDLG:

Definition 5 (Left-Right Deterministic Linear Grammars). *A CFG G is* Left-Right Deterministic Linear Grammar (LRDLG) *iff $G \in LDLG \cup RDLG$.*

The languages generated by LRDLG are called Left-Right Deterministic Linear Languages (LRDLL).

This class is closed over the reversal operation.

2.3 Learning and Identifying

In this paper we are concerned with the identification in the limit from polynomial time and data using positive and negative information. In this setting the learner is asked to learn from a learning *sample*, *i.e.* a finite set of strings, each string labelled by '+' if the string is a positive instance of the language (an element of L), or by '$-$' if it is a negative instance of the language (an element of $\Sigma^* - L$). Alternatively we denote $I = (I_+, I_-)$ where I_+ is the sub-sample of positive instances and I_- the sub-sample of negative ones.

Definition 6 (Identification in the limit from polynomial time and data). *A class $\mathcal{L}$ of languages is identifiable in the limit from polynomial time and data in terms of a grammar class $\mathcal{G}$* iff *there exist two polynomials $p()$ and $q()$ and an inference algorithm $\phi(\cdot)$ such that:*

1. *Given any sample (I_+, I_-), $\phi(I)$ returns a grammar $G \in \mathcal{G}$ such that $I_+ \subseteq L(G)$ and $I_- \cap L(G) = \emptyset$ in $O(p(\|I\|))$ time;*
2. *$\forall L \in \mathcal{L}$ and $\forall G \in \mathcal{G} : L(G) = L$, there exists a sample $C = (C_+, C_-)$(called characteristic) such that $\|C\| < q(\|G\|)$ for which, if $C_+ \subseteq I_+$, $C_- \subseteq I_-$, $\phi(I)$ returns a grammar G' such that $L(G') = L$.*

To simplify, we are going to say that a class of grammars $\mathcal{G}$ is identifiable in the limit from polynomial time and data if the class of languages $\mathcal{L} = L(\mathcal{G})$ is identifiable in the limit from polynomial time and data in terms of $\mathcal{G}$.

With this definition it is known that deterministic finite automata [5] and even linear grammars [6] are identifiable in the limit from polynomial time and data whereas non-deterministic finite automata and linear (and hence context-free) grammars are not [3].

3 Left Deterministic Linear Languages

As was pointed in section 2.2, the definition of LDLL is somewhat similar to the definition of Deterministic Regular Grammars. This similarity is going to allow us to define a normal form and a canonical automaton for such type of languages.

3.1 Canonical form

Let us first define the common suffix free languages that are going to play the role of the set of tails in a regular language.

LDLL use common suffix properties; in the sequel we are going to denote the *longest common suffix reduction* of a language L by $L\downarrow = L(\mathrm{lcs}(L))^{-1}$.

Definition 7 (Common suffix-free language equivalence). *Given a language L we define recursively the* common suffix-free *languages $\mathrm{CSF}_L(\cdot)$, and the associated equivalence relation as follows:*

$$\begin{array}{l} \mathrm{CSF}_L(\lambda) = L \\ \mathrm{CSF}_L(xa) = (a^{-1}\,\mathrm{CSF}_L(x))\downarrow \end{array} \quad \Bigg| \quad x \equiv_L y \iff \mathrm{CSF}_L(x) = \mathrm{CSF}_L(y)$$

It was shown in [4] that, a $L \in \mathrm{LDLL}$ iff $\{\mathrm{CSF}_L(x) : x \in \Sigma^*\}$ is finite.

A consequence of this is the following corolary:

Corollary 1. *Let $L \notin LDLL$ then $|\{\mathrm{CSF}_L(x) : x \in \Sigma^*\}| = \infty$*

Now, following the parallelism with the Deterministic Regular Grammars, the canonical grammar for a LDLL can be defined as follows:

Definition 8 (canonical grammar for LDLL). *Given any linear deterministic language L, the associated canonical grammar is $G_L = (\Sigma, V, P, S_{\mathrm{CSF}_L(\lambda)})$ where:*

$$\begin{aligned} V =& \{S_{\mathrm{CSF}_L(x)} : \mathrm{CSF}_L(x) \neq \emptyset\} \\ P =& \{S_{\mathrm{CSF}_L(x)} \to a S_{\mathrm{CSF}_L(xa)}\,\mathrm{lcs}(a^{-1}\,\mathrm{CSF}_L(x)) : \mathrm{CSF}_L(xa) \neq \emptyset\} \\ & \cup \{S_{\mathrm{CSF}_L(x)} \to \lambda : \lambda \in \mathrm{CSF}_L(x)\} \end{aligned}$$

We are going to define now a cannonical form to write this grammar:

Definition 9 (Advanced form for LDLL). *A linear grammar $G = (\Sigma, V, P, S)$ is deterministic in* advanced *form if:*

1. *all rules are in the form $(T, aT'w)$ or (T, λ);*

2. $\forall (T, aT'w) \in P, w = \text{lcs}(a^{-1}L_G(T))$;
3. *all non-terminal symbols are accessible:* $\forall T \in V\ \exists u, v \in \Sigma^* : S \xrightarrow{*} uTv$ *and useful:* $\forall T \in V,\ L_G(T) \neq \emptyset$;
4. $\forall T, T' \in V,\ L_G(T) = L_G(T') \Rightarrow T = T'$.

Now it was proved in [4] that:

Theorem 1. *Let $L \in LDLL$, then G_L is the smallest LDLL advanced grammar such that $L(G_L) = L$. Moreover, it is unique up to isomorphisms.*

3.2 Learning LDLL

As LDLL admit a small canonical form it is sufficient to have an algorithm that can learn this type of canonical form at least when a characteristic set is provided. In doing so we are following the type of proof used to prove learnability of *dfa* [7, 5].

The idea of the algorithm is to provide a systematic way to build the canonical grammar provided we can make some type of queries to an unlimited oracle. In a second step, the queries to the oracle are changed by functions that extract equivalent information from the learning set.

Let first introduce the concept of *short prefix*:

Definition 10. *Let L be a LDLL, and $\leqslant$ a length lexicographic order relation over Σ^*, the* shortest prefix *set of L is defined as* $\text{Sp}_L = \{x \in \text{Pr}(L) : \text{CSF}_L(x) \neq \emptyset \wedge y \equiv_L x \Rightarrow x \leqslant y\}$

Note that, in a canonical grammar, we have a one to one relation between strings in Sp and non terminals of the grammar. We shall thus use the strings in Sp as identifiers for the non terminal symbols.

Imagine we have an unlimited oracle that knows language L and to which we can address the following queries:

$$\begin{array}{rl} \text{next}(x) = \{xa : \exists xay \in L \wedge \text{CSF}_L(xa) \neq \emptyset\} & \quad \text{equiv}(x, y) \iff x \equiv_L y \\ \text{right}(xa) = \text{lcs}(a^{-1}\,\text{CSF}_L(x)) & \quad \text{isfinal}(x) \iff \lambda \in \text{CSF}_L(x) \end{array}$$

An algorithm (alg. 1) can be built to construct the canonical grammar. Algorithm 1 visits the prefixes of the language L in length lexicographic order, and constructs the canonical grammar responding to definition 8. If a prefix xa is visited and no previous equivalent non terminal has been found (and placed in Sp), this prefix is added to Sp as a new non terminal and the corresponding rule is added to the grammar. If there exists an equivalent non terminal y in Sp then the corresponding rule is added but the strings for which x is a prefix will not be visited (they will not

be added to W). When the algorithm finishes, Sp contains all the short prefixes of the language.

In order to simplify notations we introduce:

Definition 11.

$$\forall x : \mathrm{CSF}_L(x) \neq \emptyset,\ \mathrm{tail}_L(x) = \begin{cases} \mathrm{lcs}(x^{-1}L) & \text{if } x \neq \lambda \\ \lambda & \text{if } x = \lambda \end{cases}$$

Lemma 1. *Let $G_L = (\Sigma, V, P, S)$ be the canonical grammar of a LDLL L, $\forall x$: $CSF(x) \neq \emptyset$,*

1. $\mathrm{lcs}(a^{-1}\,\mathrm{CSF}_L(x)) = (\mathrm{tail}_L(xa))(\mathrm{tail}_L(x))^{-1}$
2. $xv\,\mathrm{tail}_L(x) \in L \iff v \in L_{G_L}([x])$.

In order to use algorithm 1 with a sample $I = (I_+, I_-)$ instead of an oracle with access to the whole language L the 4 functions next, right, equiv and isfinal have to be implemented as functions of $I = (I_+, I_-)$ rather than of L:

$$\begin{aligned} \mathrm{next}(x) &= \{xa : \exists xay \in I_+\} \\ \mathrm{right}(xa) &= \mathrm{tail}_{I_+}(xa)\,\mathrm{tail}_{I_+}(x)^{-1} \\ \mathrm{equiv}(x,y) &\iff xv\,\mathrm{tail}_{I_+}(x) \in I_+ \Rightarrow yv\,\mathrm{tail}_{I_+}(y) \notin I_- \\ &\quad \wedge yv\,\mathrm{tail}_{I_+}(y) \in I_+ \Rightarrow xv\,\mathrm{tail}_{I_+}(x) \notin I_- \\ \mathrm{isfinal}(x) &\iff x\,\mathrm{tail}_{I_+}(x) \in I_+ \end{aligned}$$

Algorithm 1 Computing G using functions next, right, equiv and isfinal

Require: functions next, right, equiv and isfinal, language L
Ensure: $L(G) = L$ with $G = (\Sigma, V, P, S_\lambda)$
  $\mathrm{Sp} = \{\lambda\}$; $V = \{S_\lambda\}$
  $W = \mathrm{next}(\lambda)$
  while $W \neq \emptyset$ **do**
    $xa = \min_{\leq} W$
    $W = W - \{xa\}$
    if $\exists y \in \mathrm{Sp} : \mathrm{equiv}(xa, y)$ **then**
      add $S_x \to aS_y\,\mathrm{right}(xa)$ to P
    else
      $\mathrm{Sp} = \mathrm{Sp} \cup \{xa\}$; $V = V \cup \{S_{xa}\}$
      $W = W \cup \mathrm{next}(xa)$
      add $S_x \to aS_{xa}\,\mathrm{right}(xa)$ to P
    end if
  end while
  for all $x \in \mathrm{Sp} : \mathrm{isfinal}(x)$ **do**
    add $S_x \to \lambda$ to P
  end for

It is easy to see that, if a set fulfils the following conditions, then the algorithm will be force to output the canonical grammar (see [4] for detail).

Definition 12 (characteristic sample). *Let $I = (I_+, I_-)$ be a sample of the LDLL L. I is a* characteristic sample *(CS) of L if:*

1. $\forall x \in \mathrm{Sp}_L \, \forall a \in \Sigma : xa \in \mathrm{Pr}(L) \Rightarrow \exists xaw \in I_+$
2. $\forall x \in \mathrm{Sp}_L \, \forall a \in \Sigma : \mathrm{CSF}_L(xa) \neq \emptyset \Rightarrow \mathrm{tail}_{I_+}(xa) = \mathrm{tail}_L(xa)$
3. $\forall x, y \in \mathrm{Sp}_L \, \forall a \in \Sigma : \mathrm{CSF}_L(xa) \neq \emptyset \wedge xa \not\equiv_L y \Rightarrow$
   $\exists v : xav \, \mathrm{tail}_L(xa) \in I_+ \wedge yv \, \mathrm{tail}_L(y) \in I_- \; \vee$
   $\exists v : yv \, \mathrm{tail}_L(y) \in I_+ \wedge xav \, \mathrm{tail}_L(xa) \in I_-$
4. $\forall x \in \mathrm{Sp}_L : x \, \mathrm{tail}_L(x) \in L \Rightarrow x \, \mathrm{tail}_L(x) \in I_+$

Condition 1 assures that all the non terminals will be represented on the output grammar. Condition 2 assures that the right hand part of the rules will be well constructed. Condition 3 assures that every non equivalent non terminals tested on the algorithm will be detected as non equivalent. And condition 4 assures that all the rules with shape $A \to \lambda$ will be included in the grammar.

In [4] was proved that a polynomial set that fulfils all the conditions can be build.

As a corollary of that we have:

Corollary 2. *The LDLG can be identified in the limit from polynomial time and data using positive and negative sample.*

4 Learning LRDLG

On the previous section we have defined an algorithm LDLGA$(\cdot)$ that identifies the LDLG. Reminding that RDLL = LDLL^R, then it is easy to build an algorithm RDLGA$(\cdot)$ for RDLG such that RDLGA$(I) = (\mathrm{LDLGA}(I^R))^R$.

Now, for the LRDLG let us define an algorithm (LRDLGA) (see alg. 2) that given a sample, uses it with LDLGA and RDLGA, and returns the hypothesis grammar with a lower number of non terminals.

If the target language is in LDLL$-$RDLL and the sample is enough big, LDLGA will provide the canonical LDLG for the language, but the RDLGA, by corollary 1, is going to produce bigger and bigger grammars as the sample grows. Then it has to exist a point when the correct hypothesis will be outputted.

The case when the language is in RDLL$-$LDLL is similar. And the case when the target is in both classes, LRDLGA will output the smaller of both representations.

Now, in order to show the identification in the limit from polynomial time and data, we have to show the existence of a polynomial characteristic set. The idea is to find a sample such that, if the language is not in the class it will force the algorithm to output a hypothesis with size bigger that a given parameter. Let us formalize this idea:

Definition 13 (n-negative characteristic sample). *Let $\phi(\cdot)$ an inference algorithm that identifies in the limit the class of languages $\mathcal{L}$ in terms of the class of grammars $\mathcal{G}$, let $L \notin \mathcal{L}$, $C = (C_+, C_-)$ is a n-*negative characteristic sample *$(n$-NCS$)$ for ϕ if for all sample $I = (I_+, I_-)$ of $L : C_+ \subseteq I_+, C_- \subseteq I_-$, then $\|\phi(I)\| \geq n$.*

In our case, we are going to use the number of non terminals as the size of a grammar. Then, if we can show that for every language in $\mathrm{LDLL} - \mathrm{RDLL}$ (or $\mathrm{RDLL} - \mathrm{LDLL}$) with n non terminals we can find a polynomial size $(n+1)$-NCS for RDLGA (LDLGA), the union of the characteristic sample for LDLGA (RDLGA) of the language with the $(n+1)$-NCS will be a polynomial size characteristic sample for the LRDLGA.

Let us show that this polynomial size n-NCS exists.

Proposition 1. *Let $L \notin$ LDLL and let $n \in \mathbb{N}$. As $L \notin$ LDLL we know that $|\operatorname{Sp}_L|$ is infinite, let Sp_{L_n} be the set of n smallest elements $x \in \operatorname{Sp}_L$ in the lenght lexicographic order. Let $I = (I_+, I_-)$ be a sample of L, I is an n-negative characteristic sample $(n$-NCS$)$ for LRDLGA if:*

1. $\forall x \in \operatorname{Sp}_{L_n} \forall a \in \Sigma : xa \in \Pr(L) \Rightarrow \exists xaw \in I_+$
2. $\forall x \in \operatorname{Sp}_{L_n} \forall a \in \Sigma : \mathrm{CSF}_L(xa) \neq \emptyset \Rightarrow \operatorname{tail}_{I_+}(xa) = \operatorname{tail}_L(xa)$
3. $\forall x, y \in \operatorname{Sp}_{L_n} \forall a \in \Sigma : \mathrm{CSF}_L(xa) \neq \emptyset \wedge xa \not\equiv_L y \Rightarrow$
   $\exists v : xav \operatorname{tail}_L(xa) \in I_+ \wedge yv \operatorname{tail}_L(y) \in I_- \ \vee$
   $\exists v : yv \operatorname{tail}_L(y) \in I_+ \wedge xav \operatorname{tail}_L(xa) \in I_-$
4. $\forall x \in \operatorname{Sp}_{L_n} : x \operatorname{tail}_L(x) \in L \Rightarrow x \operatorname{tail}_L(x) \in I_+$

Algorithm 2 Computing the grammar G for a language $L \in$ LRDLL

Require: Algorithm 1, language L
Ensure: $L(G) = L$ with $G = (\Sigma, V, P, S_\lambda)$ and $|V|$ smaller.
  Let $G_L = (\Sigma, V_L, P_L, S_{\lambda,L})$ the grammar computed by algorithm 1 with L as input.
  Let $G_R = (\Sigma, V_R, P_R, S_{\lambda,R})$ the reversed grammar computed by algorithm 1 with L^R as input.
  if $|V_L| \leq |V_R|$ **then**
    $G = G_L$
  else
    $G = G_R$
  end if

Proof. As $L \notin$ LDLL we know that $\{CSF_L(x)\}$ is infinite and then, if we try to build a canonical grammar, we are going to obtain an infinite number of non terminals. Observe that the inference algorithm constructs the grammar iteratively from the non terminal nearest to the start symbol to the farthest. The conditions of the n-NCS provide enough information to the inference algorithm to construct correctly the productions related to the first n non terminal of the infinite grammar.

It is easy to see that condition 1 assures that all the first n non terminals will be represented on the output grammar. Condition 2 assures that the right part of the rules related with the first n non terminals will be well constructed. Condition 3 assures that every non equivalent non terminals (of the first n) tested on the algorithm will be detected as non equivalent. And condition 4 assures that all the rules with shape $A \to \lambda$, for the first n non terminals will be included in the grammar. □

Now we have to show that there is a polynomial sample that fulfils the previous conditions. In order to show this, the following lemma proved in [4] is needed.

Lemma 2. *Let $G_L = (\Sigma, V, P, S)$ be the canonical grammar of a LDLL L, and let x, y be such that $\mathrm{CSF}_L(x) \neq \mathrm{CSF}_L(y)$, then $\exists z \in L_{G_L}([x]) \ominus L_{G_L}([y])$ such that $|z| \leq ||G_L||^2$.*

Theorem 2. *For any $L \notin$ LDLL there is an n-negative characteristic sample of polynomial size.*

Proof. Obviously, the number of strings involved in the conditions is polynomial, then we have to show that their lengths are also polynomial.

Note that each time a production rule of a LDLG is applied in the parsing of a string, a non terminal is removed from the prefix of a string, then $\forall x \in \mathrm{Sp}_{L_n} : |x| \leq n$. Those strings needs to reach the non terminal represented by the short prefix x use the rule whose right hand part beginning with terminal a and then a string to reach a final non terminal (a terminal A such that $A \to \lambda \in P$). So, the length of the strings xaw in condition 1 of proposition 1 are bounded by $(2n+1)(|w_l|+1)$, where w_l is the longest suffix in the right hand side of the production rules of G.

In a similar way, we can see that the length of strings related with condition 2 and 4 can also be bounded by $(2n+1)(|w_l|+1)$.

Finally, lemma 2 shows that the length of the strings necessary for third condition can be quadratically bounded. □

Example 1. Consider the language $L = \{a^n b^n c^m : n \geq 0, m \geq 0\}$. This language is in RDLL but is not in LDLL. The right canonical grammar G_R for it is:

$$\begin{aligned} S &\longrightarrow Sc \\ S &\longrightarrow aAb \\ S &\longrightarrow \lambda \\ A &\longrightarrow aAb \\ A &\longrightarrow \lambda \end{aligned}$$

We are going to show that the left canonical grammar has an infinite number of non terminals.

The characteristic sample (I_+, I_-) with $I_+ = \{\lambda, c, ab, abc, aabb, abcc\}$ and $I_- = \{acb, aaccbb\}$ let us identify[1] G_R with $\mathrm{Sp}_R = \{\lambda, a\}$, ($S_\lambda \equiv S$, $S_a \equiv A$). On the other hand, we can compute $CSF_L(x)$ for every $x \in \mathrm{Pr}(L)$:

x	$CSF(x)$
λ	$a^n b^n c^m$
a	$a^n b^{n+1} c^m$
c	$a^n b^n c^m$
aa	$a^n b^{n+2} c^m$
ab	c^m
aaa	$a^n b^{n+3} c^m$
aab	bc^m
abc	c^m
$\cdots$	$\cdots$

One can see that in this case Sp_L remains unbounded and, in order to identify correctly the grammar as RDLG with algorithm LRDLGA, is sufficient supply a 3-NCS. If we add the string $aabbc$ to I_+ and the strings b and bb to I_-, the sample provided above becomes 3-NCS for languages in LDLL, giving the left grammar:

$$\begin{aligned} S_\lambda &\longrightarrow aS_a \\ S_\lambda &\longrightarrow cS_\lambda \\ S_\lambda &\longrightarrow \lambda \\ S_a &\longrightarrow aS_{aa} \\ S_a &\longrightarrow bS_\lambda \\ S_{aa} &\longrightarrow bS_a \end{aligned}$$

[1] Recall that the input to algorithm 1 must be the reversal of the sample, and the obtained grammar must be also reversed.

5 Summary and Future Work

Left Deterministic Linear Languages (LDLL) and Right Deterministic Linear Languages (RDLL) are subclasses of Linear Languages that, in turn, includes the Regular Languages. We define the Left-Right Deterministic Languages (LRDLL) as the union of the LDLL and RDLL.

In this paper we have proved that the class of the LRDLL is identifiable in the limit from polynomial time and data. This class of languages is the largest one for which this type of results has been obtained so far. To do so we have introduced the notion of n-negative characteristic sample as a sample that forces an inference algorithm to produce a hypothesis of size n when strings from a non identifiable grammar are provided.

Note that in the parsing of a string by a LDLG if we have reached a non terminal, the next rule to apply can be determined by looking the leftmost terminal of the non parsed string. In RDLG the nonterminal to look is the rightmost. Let we call the non terminals in LDLG left deterministic while the non terminals in RDLG right deterministic.

Now, a new class of linear languages can be defined as a grammars such that each non terminal is left deterministic or right deterministic, but not both at the same time. It is easy to see that, on such grammars, the parsing can be done in a deterministic way provided we know if the reached non terminal is left or right deterministic. This class includes properly the LRDLL.

Can the n-negative characteristic sample technique be expanded in order to elucidate if a non terminal is left or right deterministic? Can this class of grammars be identified from polynomial time and data?

6 Acknowledgement

The authors thank Colin de la Higuera for fruitful discussions on the subject.

References

[1] E.M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[2] L. Pitt. Inductive inference, DFA's, and computational complexity. In *Analogical and Inductive Inference*, number 397 in Lecture Notes in Artificial Intelligence, pages 18–44. Springer-Verlag, Berlin, 1989.

[3] C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27:125–138, 1997.

[4] C. de la Higuera and J. Oncina. Learning deterministic linear languages. In *Computational Learning Theory, COLT 02*, number 2375 in Lecture Notes in Artificial Intelligence, pages 185–200. Springer Verlag, 2002.

[5] J. Oncina and P. García. Identifying regular languages in polynomial time. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition*, volume 5 of *Series in Machine Perception and Artificial Intelligence*, pages 99–108. World Scientific, 1992.

[6] J.M. Sempere and P. García. A characterisation of even linear languages and its application to the learning problem. In *Grammatical Inference and Applications, ICGI'94*, number 862 in Lecture Notes in Artificial Intelligence, pages 38–44. Springer Verlag, 1994.

[7] E.M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.