Inferring Deterministic Linear Languages

Colin de la Higuera[1†] and Jose Oncina[2‡]

[1] EURISE, Université de Saint-Etienne, 23 rue du Docteur Paul Michelon,
42023 Saint-Etienne, France
cdlh@univ-st-etienne.fr,
WWW home page: http://eurise.univ-st-etienne.fr/~cdlh
[2] Departamento de Lenguajes y Sistemas Informáticos,
Universidad de Alicante, Ap.99. E-03080 Alicante, Spain
oncina@dlsi.ua.es,
WWW home page: http://www.dlsi.es/~oncina

Abstract. Linearity and determinism seem to be two essential conditions for polynomial language learning to be possible. We compare several definitions of deterministic linear grammars, and for a reasonable definition prove the existence of a canonical normal form. This enables us to obtain positive learning results in case of polynomial learning from a given set of both positive and negative examples. The resulting class is the largest one for which this type of results has been obtained so far.

1 Introduction

The general problem of learning formal grammars, also known as grammatical inference has concentrated essentially on the sub-case of deterministic finite automata (*dfa*) or regular inference for which have been obtained positive results in different learning settings [1], but also negative ones, such as the impossibility (in polynomial time) to find a small *dfa* consistent with the learning data [2], or the dependence of being able to approximatively learn *dfa* depending on well accepted cryptographic assumptions [3].

The harder question of learning context-free grammars has given rise to different approaches over the past few years. A survey on grammatical inference (with special interest on context-free grammar learning) can be found in [4] and a specific survey covering a wide range of context-free grammar learning results can be found in [5]. Main results include:

- Negative results have been provided: the general class of context-free grammars is not learnable [6]. Moreover even though this also holds for the class of linear languages, one major problem for the general case is that of expansiveness, *i.e.* the capacity for exponentially long words to be obtained through parsing trees of polynomial hight.

† This work was done when the first author visited the Departamento de Lenguajes y Sistemas Informáticos of the University of Alicante, Spain. The visit was sponsored by the Spanish Ministry of Education.

‡ The second author thanks the Spanish CICyT for partial support of this work through project TIC2000-1703-C03-02.

- Positive results have been obtained in the context of sub-classes of linear grammars. For instance *even linear grammars* have been studied [7] and identification in different contexts established. A generalization of these results (but still to special classes of linear grammars) has been given [8].
- Typical artificial intelligence techniques have been used to search for a small consistent context-free grammar. For instance Sakakibara and Kondo [9] use genetic algorithms to search for a context-free grammar in Chomsky normal form, consistent with the data.
- With the help of additional information (parenthesis, distribution, queries) alternative algorithms have been obtained: structured examples (the strings are given with the unlabelled parse tree) allow to adapt *dfa* learning algorithms to infer context-free grammars [10]. Probabilistic context-free grammars are learned from structured positive strings [11]. A subclass of context-free languages for which the expansiveness problem is an issue is dealt with in [12], but in the context of learning from queries [13].

A close analysis of the positive results for polynomial learning grammars or automata and the negative ones for larger classes shows us the importance of two factors:

- Determinism: the fact that there is just one way of parsing any string, even if it is not in the language, is a much used feature in all greedy algorithms. A rule can be refuted at any one point because it's presence would imply the presence of some undesired string.
- Linearity: this allows a polynomial relationship (in fact linear) between the size of the grammar and that of small, representative strings in the language. Such strings are needed in *characteristic sets* [6].

It is thus reasonable to consider deterministic sub-classes of the linear languages. Moreover it would be interesting that the class could generalize that of the regular languages. A variety of definitions concerning such grammars exist, which we shall discuss in the next section. In section 3 we will give the characteristics of this class and provide a *Nerode* type of theorem, allowing us to consider a canonical form of deterministic linear grammar. In section 4 we make use of this normal form in an algorithm that can identify in the limit deterministic linear grammars from polynomial time and data. We conclude in section 5.

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

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 $\mathrm{Pr}(L) = \{x : xy \in L\}$.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 $(Lu^{-1})u = L$.

A *context-free grammar* G is a quadruple $< \Sigma, V, P, S >$ 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 $uTv \rightarrow uwv$ when $(T, w) \in P$. $\xrightarrow{*}$ is the reflexive and transitive closure of $\rightarrow$. If there exists $u_0, \ldots, u_k$ such that $u_0 \rightarrow \cdots \rightarrow u_k$ we will write $u_0 \xrightarrow{k} u_k$. We denote by $L_G(T)$ the language $\{w \in \Sigma^* : T \xrightarrow{*} w\}$. Two grammars are equivalent if they generate the same language. 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 [14]. We will consider here the size of G denoted $\|G\| = \sum_{(T,u) \in P} (|u| + 1)$.

Another way of defining context-free languages is through *pushdown automata* (*pda*). We will not formally define these here. A specific study on these machines can be found in [15].

Informally a *pda* is a one-way finite state machine with a stack. Criteria for recognition can be by empty stack or by accepting states, but in both cases the class of languages is that of the context-free ones. If the machine is deterministic (in a given configuration of the stack, with a certain symbol to be read, only one rule is possible) the class of languages is that of the *deterministic languages*. Given a computation of a *pda*, a *turn* in the computations is a move that decreases the height of the stack and is preceded by a move that did not decrease it. A *pda* is said to be *one-turn* if in any computation there is at most one turn. A language is linear if and only if it is recognized by a one-turn *pda* [15].

2.2 Deterministic Linear Grammars

A variety of definitions have been given capturing the ideas of determinism and linearity. We first give our own formalism:

Definition 1 (Deterministic linear grammars). *A* deterministic linear (DL) grammar *$G =< \Sigma, V, P, S >$ is a linear grammar where all rules are of the form $(T, aT'u)$ or (T, λ) and $(T, au), (T, av) \in P \Rightarrow u = v$ with $u, v \in V\Sigma^*$.*

This definition induces the determinism on the first symbol of the right hand of rules (as in [12]). This extends by an easy induction to the derivations: let $T \xrightarrow{*} uT'v$ and $T \xrightarrow{*} uT''w$ then $T' = T''$ and $v = w$.

A more general form of determinism, allowing for less restrictive right hand of rules, is given by the following definition:

Definition 2 (General deterministic grammars). *A general deterministic linear (*Gen-DL*) grammar is a linear grammar* $G =< \Sigma, V, P, S >$ *where* $\forall u_3 \in \Sigma^+$*:* $\forall (T, u_1T_1v_1), (T, u_2T_2v_2), (T, u_3) \in P$*,* $u_i \notin \mathrm{Pr}(\{u_j\})\ \forall i \neq j$*.*

Nasu and Honda [16] defined another class of deterministic linear grammars:

Definition 3 (NH**-deterministic grammars).** *A* NH*-deterministic linear grammar* $G =< \Sigma, V, P, S >$ *is a linear grammar where all rules in* P *are of the form* $(T, aT'u)$ *and* (T, a) *and* $\forall u, v \in (\Sigma \cup V)^*$*:* $(T, au), (T, av) \in P \Rightarrow u = v$*.*

Ibarra, Jiang and Ravikumar [17] propose another type of grammars:

Definition 4 (IJR**-deterministic grammars).** *An* IJR*-deterministic linear grammar is a linear grammar* $G =< \Sigma, V, P, S >$ *where* $\forall (T, u_1T_1v_1)$*,* $(T, u_2T_2v_2)$*,* $(T, u_3) \in P$*,* $u_i \notin \mathrm{Pr}(\{u_j\})\ \forall i \neq j$*.*

NH-deterministic linear grammars cannot generate λ, whereas an IJR-deterministic linear grammar can only generate λ when the generated language is exactly $\{\lambda\}$ [18]. The possibility of having λ-rules is what differenciates these models with the one we propose. Another type of grammar, well known in compiler literature is that of the LL-grammars, that we present here adapted to the linear setting [18]:

Definition 5 (Linear LL(1) **grammars).** *A linear* LL(1) *grammar is a linear grammar* $G =< \Sigma, V, P, S >$ *such that if* $S \xrightarrow{*} uTv \rightarrow uzv \xrightarrow{*} uw$*,* $S \xrightarrow{*} uTv \rightarrow uz'v \xrightarrow{*} uw'$ *and* $\mathrm{FIRST}_G(w) = \mathrm{FIRST}_G(w')$*, where* $\mathrm{FIRST}_G(w)$ *is the first symbol of* w *(if* $w = \lambda$*,* $\mathrm{FIRST}_G(w) = \lambda$*), then* $z = z'$*.*

Let Det, Lin, DetLin, DL, Gen-DL, NH-DL, IJR-DL, and LinLL(1) be the classes of languages generated respectively by deterministic *pda*, linear grammars or one-turn *pda*, deterministic one-turn *pda*, DL grammars, Gen-DL grammars, NH-deterministic linear grammars, IJR-deterministic linear grammars and linear LL(1) grammars.

2.3 About Deterministic Linear Languages

As described in the previous section there is a variety of definitions that may capture the idea of linearity and determinism in context-free languages. They are not all equivalent, but the following holds:

Theorem 1. DL = Gen-DL*.*

Proof. Clearly DL $\subseteq$ Gen-DL. Conversily, algorithm 1 transforms any Gen-DL grammar into a DL grammar. Note that the algorithm terminates when a DL grammar is reached (each step decreases the length of some right hand of rule). After each step the grammar remains Gen-DL. Finally the size of the DL grammar we obtain from a Gen-DL grammar $G =< \Sigma, V, P, S >$ is at most $2\|G\|$, so linear in that of G.

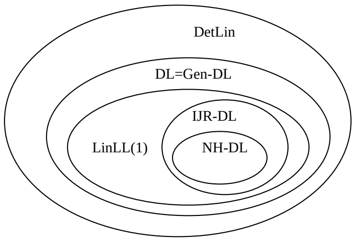
Algorithm 1 Transforming a general grammar

Require: G
Ensure: G is DL
  while G not a DL-grammar **do**
    Choose $T \in V, a \in \Sigma$ such that $\exists(T, auT'v) \in P$, with $u \in \Sigma^+, v \in \Sigma^*$
    $U \leftarrow \{v : (T, auT'v) \in P\}$.
    Replace each rule $(T, auT'v)$ by rules $(T, aT''(\mathrm{lcs}(U))$ and $(T'', uT'v(\mathrm{lcs}(U))^{-1})$
    where T'' is a new non-terminal associated to the pair (a, T)
  end while

We note that LinLL(1) contains only languages with the *prefix*[1] property (so do NH-DL and IJR-DL). Other differences are:

Theorem 2 (Some strict inclusions).

1. DetLin $\subset$ Det $\cap$ Lin
2. DL $\subset$ DetLin
3. LinLL(1) $\subset$ DL
4. IJR-DL $\subset$ LinLL(1)
5. NH-DL $\subset$ IJR-DL



Proof.

1. $\{a^n b^m a^p b^q : n = m \text{ or } p = q, n, m, p, q > 0\} \in$ Det $\cap$ Lin $\setminus$ DetLin [15]. A deterministic *pda* and a one-turn *pda* can both be designed to generate this language, but no one-turn deterministic *pda* can exist: you have to decide at the beginning of the parse what you want to count.
2. $\{a^n b^n : n > 0\} \cup \{a^n c^n : n > 0\} \in$ DetLin $\setminus$ DL based on [15]. Yet the mirror language is in DL.
3. (LinLL(1) $\subset$ DL). Let $G =< \Sigma, V, P, S >$ be a linear LL(1) grammar. Then G has no left recursive variable (*i.e.* T such that $T \xrightarrow{*} Tu$).
   By standard techniques any rule in P of the form $T \rightarrow T'u$ can be replaced by rules $T \rightarrow zu$ ($z \in \Sigma^* V \Sigma^* \cup \Sigma^*$) for each rule $(T' \rightarrow z)$ in P, resulting in a LL(1) linear grammar with no $T \rightarrow T'u$ rules.
   We prove that all rules are either $T \rightarrow auT'v$ or $T \rightarrow u$, and respect conditions of definition 2: there are 4 cases to consider; we sketch the proof for the 2 main ones:
   – $T \rightarrow auT'v \in P$ and $T \rightarrow au'T''v' \in P \Rightarrow u = u'$, $T' = T''$, $v = v'$ since G is linear LL(1).
   – $T \rightarrow auT'v \in P$ and $T \rightarrow u' \in P \Rightarrow \mathrm{FIRST}(u') \neq a$.
   The inclusion is strict as LinLL(1) only contains languages with the prefix property. A language like $\{a^n b^p, 0 \leqslant n \leqslant p\} \in$ DL $\setminus$ LinLL(1).

[1] no string in the language is a proper prefix of another string in the language.

4. The stronger NH-DL $=$ λ-free LinLL(1) is proved in [18];
5. One can actually prove the stronger NH-DL $=$ IJR-DL $\setminus\{\lambda\}$ [18].

As we will in the sequel concentrate on DL-languages, here are some examples of such languages: $\{a^nb^n : 0 \leqslant n\}$, $\{a^mb^n : 0 \leqslant m \leqslant n\}$, $\{a^mb^nc^n : 0 \leqslant m, 0 \leqslant n\}$ (a variant of the main example in [9]), the set of all palindromes over some alphabet Σ, any linear LL(1) language. All regular languages are also DL as any regular grammar obtained from a *dfa* is DL.

It should also be noticed that some even linear languages [7] are not DL-languages: $\{a^nb^n, n > 0\} \cup \{a^nc^n, n > 0\}$ for instance.

2.4 Learning and Identifying

A variety of learning paradigms have been examined and used in the literature of machine learning, but in the case of grammatical inference some of these:

– are too hard (no reasonable class can be learned): *PAC*-learning[2] for instance [19];
– are too easy (nearly anything can be learned): identification in the limit [20].

Based on identification in the limit, we will use identification in the limit from polynomial time and data [6] as our learning model. Learning from queries [13] or simple-*PAC* learning [21] are alternative frameworks. It should be noticed that if a class of grammars is identifiable in the limit from polynomial time and data, the simple grammars in this class are also simple-*PAC* learnable [22]. In this setting the learner is asked to learn from a learning *sample*, *i.e.* a finite set of strings, each string labeled 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 $\Sigma^*\setminus L$). Alternatively we will denote $S = (S_+, S_-)$ where S_+ is the sub-sample of positive instances and S_- the sub-sample of negative ones.

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

1. *Given any sample (S_+, S_-), $\mathcal{A}$ returns a grammar G in $\mathcal{G}$ such that $S_+ \subseteq L_G$ and $S_- \cap L_G = \emptyset$ in $O(p(\|S_+\| + \|S_-\|))$ time;*
2. *for each grammar G in $\mathcal{G}$ of size n, there exists a characteristic sample (CS_+, CS_-) such that $\|CS_+\| + \|CS_-\| < q(n)$ for which, if $CS_+ \subseteq S_+$, $CS_- \subseteq S_-$, $\mathcal{A}$ returns a grammar G' equivalent with G.*

With this definition it is known that deterministic finite automata [1] and even linear grammars [7] are identifiable in the limit from polynomial time and data whereas non-deterministic finite automata and linear (and hence context-free) grammars are not [6].

[2] Probably Approximatively Correct

3 Deterministic Linear Languages have Finite Index and Canonical Form

For a class of languages to be identifiable in the limit from polynomial time and data a reasonable assumption is that there exists some small canonical form for any language representable in the class. We prove in this section that such is indeed the case for deterministic linear grammars.

Deterministic linear languages 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:*

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

We define now another equivalence relation over Σ^*, when given a deterministic linear grammar:

Definition 8. *Let $G =< \Sigma, V, P, S >$ be a* DL *grammar. With every string x we associate the unique non terminal $[x]_G = T$ such that $S \xrightarrow{*} xTu$; we extend L_G to be a total function by setting $L_G([x]_G) = \emptyset$ when $[x]_G$ is undefined.*

We use this definition to give another equivalence relation over Σ^*, when given a deterministic linear grammar:

Definition 9. *Let $G =< \Sigma, V, P, S >$ be a* DL *grammar. We define recursively the associated* common suffix-free *languages* $\mathrm{CSF}_G(.)$, *and associated equivalence relation as follows:*

$$\begin{array}{rl|c} \mathrm{CSF}_G(\lambda) = L_G(S) & & \\ \mathrm{CSF}_G(xa) = L_G([xa]_G)\downarrow & & x \equiv_G y \iff \mathrm{CSF}_G(x) = \mathrm{CSF}_G(y) \end{array}$$

$\equiv_G$ is clearly an equivalence relation, in which all strings x such that $[x]_G$ is undefined are in a unique class. The following lemma establishes that $\equiv_G$ has finite index, when G is a deterministic linear grammar:

Lemma 1. *If $[x]_G = [y]_G$, $x \neq \lambda$ and $y \neq \lambda \Rightarrow x \equiv_G y$. Hence if G contains n non-terminals, $\equiv_G$ has at most $n+2$ classes.*

The proof is straightforward. There can be at most two possible extra classes corresponding to λ (when it is alone in its class) and the undefined class corresponding to those strings that are prefixes of no string in the language.

Proposition 1. *Let $G =< \Sigma, V, P, S >$ be a* DL *grammar, and denote $L = L_G(S)$. $\forall x \in \Sigma^*$, either $\mathrm{CSF}_L(x) = \mathrm{CSF}_G(x)$ or $\mathrm{CSF}_L(x) = \emptyset$*

Proof. By induction on the length of x.

Base: $x = \lambda$, then $\mathrm{CSF}_L(x) = L = \mathrm{CSF}_G(x)$.

Suppose: the proposition is true for all strings of length up to k, and consider string xa of length $k+1$. $\mathrm{CSF}_L(xa) = (a^{-1}\,\mathrm{CSF}_L(x))\downarrow$ (by definition 7).
If$\mathrm{CSF}_L(x) = \emptyset$, $\mathrm{CSF}_L(xa) = \emptyset$. If not ($\mathrm{CSF}_L(x) = \mathrm{CSF}_G(x)$) by induction hypothesis, $\mathrm{CSF}_L(xa) = (a^{-1}\,\mathrm{CSF}_L(x))\downarrow = (a^{-1}\,\mathrm{CSF}_G(x))\downarrow$ and there are two sub-cases:
if $x = \lambda$ $\mathrm{CSF}_G(x) = L_G([x]_G)$, so $\mathrm{CSF}_L(xa) = (a^{-1}L_G([x]_G))\downarrow$
if $x \neq \lambda$ $\mathrm{CSF}_G(x) = L_G([x]_G)\downarrow$, so: $CSF_L(xa) = (a^{-1}(L_G([x]_G)\downarrow))\downarrow$ (by definition 9), $=(a^{-1}(L_G([x]_G)))\downarrow$
In both cases follows: $CSF_L(xa) = L_G([xa]_G)\downarrow = \mathrm{CSF}_G(xa)$.

The following corollary makes use of lemma 1 and proposition 1:

Corollary 1. *Let $G =< \Sigma, V, P, S >$ be a deterministic linear grammar. So $\equiv_{L_G(S)}$ has finite index.*

The purpose of this section is to reach a computable normal form for DL grammars. For this we first define a normal form for these grammars (called *advanced* as the longest common suffixes appear as soon as possible), and then construct such a grammar from any deterministic linear language.

Definition 10 (Advanced form for deterministic linear grammars). *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. *$\forall (T, aT'w) \in P, w = \mathrm{lcs}(a^{-1}L_G(T))$;*
3. *all non-terminal symbols are accessible: $\forall T \in V\ \exists u, v \in \Sigma^* : S \xrightarrow{*} uTv$ and usefull: $\forall T \in V,\ L_G(T) \neq \emptyset$;*
4. *$\forall T, T' \in V,\ L_G(T) = L_G(T') \Rightarrow T = T'$.*

Definition 11 (A canonical grammar for deterministic linear languages). *Given any linear deterministic language L, the associated canonical grammar is $G_L =< \Sigma, V, P, S_{\mathrm{CSF}_L(\lambda)} >$ where:*

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

To avoid extra notations, we will denote (as in definition 8) by $[x]$ the non-terminal corresponding to x in the associated grammar (formally $S_{\mathrm{CSF}_L(x)}$ or $[x]_{G_L}$).

We first prove that G_L generates L: this is established through the following more general result (as the special case where $x = \lambda$):

Proposition 2. $\forall x \in \Sigma^*, L_{G_L}([x]) = \mathrm{CSF}_L(x)$.

Proof. We prove it by double inclusion.

$\forall x \in \Sigma^*,\ \mathrm{CSF}_L(x) \subseteq L_{G_L}([x])$

Proof by induction on the length of all strings in $\mathrm{CSF}_L(x)$.

Base case $|w| = 0 \Rightarrow w = \lambda$. If $\lambda \in \mathrm{CSF}_L(x), [x] \to \lambda$ so $\lambda \in L_{G_L}([x])$

Suppose now (induction hypothesis) that

$\forall x \in \Sigma^*, \forall w \in \Sigma^{\leqslant k}$: $w \in \mathrm{CSF}_L(x) \Rightarrow w \in L_{G_L}([x])$.

Let $|w| = k+1$ and let a be the first symbol of w.

Then $w = auv$, with $u \in \mathrm{CSF}_L(xa)$ and $v = \mathrm{lcs}(a^{-1}\,\mathrm{CSF}_L(x))$.

By induction hypothesis ($|u| \leqslant k$) $u \in L_G([xa])$.

As $[x] \to a[xa]v$ where $v = \mathrm{lcs}(a^{-1}\,\mathrm{CSF}_L(x))$ is a rule in P, $w \in L_G([x])$.

$\forall x \in \Sigma^*,\ L_{G_L}([x]) \subseteq \mathrm{CSF}_L(x)$

Proof by induction on the order (k) of the derivation

$\forall x \in \Sigma^*$, $\forall k \in \mathbb{N}$, $\forall w \in \Sigma^*$, $[x] \xrightarrow{k} w \Rightarrow w \in \mathrm{CSF}_L(x)$.

Base case $[x] \xrightarrow{0} w$. No rule is possible, so clearly

$\forall x \in \Sigma^*$, $\forall w \in \Sigma^*$: $[x] \xrightarrow{0} w \Rightarrow w \in \mathrm{CSF}_L(x)$.

Suppose now (induction hypothesis) that for any $n \leqslant k$:

$\forall x \in \Sigma^*$, $\forall w \in \Sigma^*$: $[x] \xrightarrow{n} w \Rightarrow w \in \mathrm{CSF}_L(x)$

Take $w \in \Sigma^*$ such that $[x] \xrightarrow{k+1} w$. There are two cases to consider:

- $[x] \to \lambda \xrightarrow{k} w$ but then $k = 0$, $w = \lambda$, and rule $[x] \to \lambda$ only exists if $\lambda \in \mathrm{CSF}_L(x)$
- $[x] \to a[xa]v \xrightarrow{k} w$.

Necessarily $w = auv$, with $[xa] \xrightarrow{k} u$. And $v = \mathrm{lcs}(a^{-1}\,\mathrm{CSF}_L(x))$

By the induction hypothesis $u \in \mathrm{CSF}_L(xa)$.

Thus $w \in a\,\mathrm{CSF}_L(xa)v \subseteq \mathrm{CSF}_L(x)$.

We turn now to some specificities of the associated canonical grammar:

Proposition 3. *Given any deterministic linear language L, G_L is advanced.*

Proof. We prove that G_L is advanced by showing that conditions 1 to 4 of definition 10 hold:

1. all rules are by construction in the form $(T, aT'w)$ and (T, λ);
2. $\forall (T, aT'w) \in P, w = \mathrm{lcs}(a^{-1}L_G(T))$ by definition 11;
3. only accessible non-terminal symbols are introduced;
4. by definition of G_L if $x \equiv_L y$ then $[x]_{G_L} = [y]_{G_L}$

Lemma 2. *Let G_L be the canonical grammar of a* DL *language L and let $x \neq \lambda$ such that $\mathrm{CSF}_L(x) \neq \emptyset$, then:*

1. $\mathrm{lcs}(L_{G_L}([x])) = \lambda$
2. $L_{G_L}([x])\downarrow = L_{G_L}([x])$
3. *let* $w : S \xrightarrow{*} x[x]w$ *then* $w = \mathrm{lcs}(x^{-1}L)$
4. *if* $\mathrm{CSF}_L(xa) \neq \emptyset$ *then* $\mathrm{lcs}(a^{-1}L_{G_L}([x])) = \mathrm{lcs}((ax)^{-1}L)(\mathrm{lcs}(x^{-1}L))^{-1}$

Proof.

1. As $x \neq \lambda$ let $x = ya$. As G_L is advanced and $\mathrm{CSF}_L(ya) \neq \emptyset$, we know that $[y] \to a[ya]w$ with $w = \mathrm{lcs}(a^{-1}L_{G_L}([y]))$. But $a^{-1}L_{G_L}([y]) = L_{G_L}([ya])w$ so $w = \mathrm{lcs}(L_{G_L}([ya])w)$ and then $\mathrm{lcs}(L_{G_L}([ya])) = \lambda$.
2. $L_{G_L}([x]) \downarrow = L_{G_L}([x])(\mathrm{lcs}\, L_{G_L}([x]))^{-1} = L_{G_L}([x])$
3. let $S \xrightarrow{*} x[x]w$ then $\mathrm{lcs}(x^{-1}L) = \mathrm{lcs}(L_{G_L}([x])w) = \mathrm{lcs}(L_{G_L}([x]))w = w$
4. As $\mathrm{CSF}_L(xa) \neq \emptyset$ let $S \xrightarrow{*} x[x]v \to xa[xa]uv$. Then $u = (uv)v^{-1}$ and as $u = \mathrm{lcs}(a^{-1}L([x])$, $uv = \mathrm{lcs}((ax)^{-1})$ and $v = \mathrm{lcs}(x^{-1}L)$, so $\mathrm{lcs}(a^{-1}L_{G_L}([x]) = \mathrm{lcs}((ax)^{-1}L))(\mathrm{lcs}(x^{-1}L))^{-1}$.

Lemma 3. *Let G be a* DL*-grammar,*

$$\forall x \in \Sigma^* : \mathrm{CSF}_L(x) \neq \emptyset \Rightarrow \exists y \in \Sigma^* : L_G([x]_G) = L_{G_L}([x]_{G_L})y.$$

Proof. If $\mathrm{CSF}_L(x) \neq \emptyset$, both $[x]_G$ and $[x]_{G_L}$ are necessarily defined. But by definition 9 $L_G([x]_G) = CSF_G(x)\,\mathrm{lcs}(L_G([x]_G))$, and in the special case the grammar is canonical $L_{G_L}([x]_{G_L}) = CSF_{G_L}(x)\,\mathrm{lcs}(L_{G_L}([x]_{G_L}))$. By lemma 2(1) $\mathrm{lcs}(L_{G_L}([x]_{G_L})) = \lambda$. ¿From proposition 1, $CSF_G(x) = CSF_{G_L}(x) = CSF_L(x)$. Hence we have $L_{G_L}([x]_{G_L})\,\mathrm{lcs}\, L_G([x]_G) = L_G([x]_G)$.

The last theorem of this section tells us that the canonical grammar is at most quadratically larger than any equivalent Gen-DL or DL grammar:

Theorem 3. *Given a* Gen-DL *grammar $G =< \Sigma, V_G, P_G, S_G >$, let G_L be the canonical grammar that generates $L = L_G(S_G)$, $\|G_L\| \leqslant \|G\|^2$.*

Proof. We denote $L = L_G(S_G)$ in the proof.

- Let G be a Gen-DL grammar, algorithm 1 produces an equivalent DL grammar whose size is at most $2\|G\|$. We thus only have to prove the case where $G =< \Sigma, V_G, P_G, S_G >$ is a DL grammar.
- By proposition 1, V has at least $k-1$ non terminals, where k is the number of classes for $\equiv_L$. We now prove that $\equiv_L$ and $\equiv_{G_L}$ coincide: there are two cases. The case $x = \lambda$ is straightforward and depends uniquely on the definitions 7 and 9. If $x \neq \lambda$, $\mathrm{CSF}_{G_L}(x) = L_{G_L}([x]_{G_L})\downarrow = L_{G_L}([x]_{G_L}) = \mathrm{CSF}_L(x)$ respectively by definition 9, by lemma 2(2) and by proposition 2. It follows that G_L has at most $|V_G| + 1$ non terminals.
- Any rule in the canonical grammar is either $([x]_{G_L}, \lambda)$ or $([x]_{G_L}, a[xa]_{G_L}w)$, with $w = \mathrm{lcs}(a^{-1}L_{G_L}([x]_{G_L}))$. Clearly $|w| \leqslant \min\{|z| : z \in a^{-1}L_{G_L}([x]_{G_L})\}$, so it follows from lemma 3 that $|w| \leqslant \min\{|z| : z \in a^{-1}L_G([x]_G)\}$. Such a string must be derived without repeating non-terminals (if not a shorter string exists), so $|w| \leqslant \sum_{(T,u)\in P_G}(|u|)$.
  ¿From this $\|G_L\| \leqslant \|\Sigma\|(|V| + 1)\sum_{(T,u)\in P_G}(|u|) \leqslant \|G\|^2$.

4 Learning DL grammars

As DL languages admit a small canonical form it will be 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* [20, 1].

Definition 12. *Let L be a* DL *language, and $\leqslant$ a length lexicographic order relation over Σ^*, the* shortest prefix *set of L is defined as* $\mathrm{Sp}_L = \{x \in \mathrm{Pr}(L) : \mathrm{CSF}_L(x) \neq \emptyset \wedge y \equiv_L x \Rightarrow x \leqslant y\}$

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:

$$\mathrm{next}(x) = \{xa : \exists xay \in L\} \qquad \mathrm{equiv}(x, y) \iff x \equiv_L y$$
$$\mathrm{right}(xa) = \mathrm{lcs}(a^{-1}\,\mathrm{CSF}_L(x)) \qquad \mathrm{isfinal}(x) \iff \lambda \in \mathrm{CSF}_L(x)$$

Algorithm 2 visits the prefixes of the language L in length lexicographic order, and constructs the canonical grammar responding to definition 11. 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 correponding 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 13.

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

Algorithm 2 Computing G using functions next, right, equiv and isfinal

Require: functions next, right, equiv and isfinal, language L
Ensure: $L(G) = L$ with $G = <\Sigma, V, P, S_\lambda>$

```
Sp = {λ}; V = {S_λ}
W = next(λ)
while W ≠ ∅ do
  xa = min_≤ W
  W = W − {xa}
  if ∃y ∈ Sp : equiv(xa, y) then
    add S_x → aS_y right(xa) to P
  else
    Sp = Sp ∪ {xa}; V = V ∪ {S_xa}
    W = W ∪ next(xa)
    add S_x → aS_xa right(xa) to P
  end if
end while
for all x ∈ Sp : isfinal(x) do
  add S_x → λ to P
end for
```

Lemma 4. *Let* $G_L =< \Sigma, V, P, S >$ *be the canonical grammar of a* DL *language* 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 2 with a sample $S = (S_+, S_-)$ 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 $S = (S_+, S_-)$ rather than of L:

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

Definition 14 (characteristic sample). *Let* $S = (S_+, S_-)$ *be a sample of the* DL *language* L. S *is a* characteristic sample *(*CS*) of* L *if:*

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

Lemma 5. *Let* $G_L =< \Sigma, V, P, S >$ *be the canonical grammar of a* DL *language* 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* $|z| \leq \|G\|^2$.

Proof. First note that $\forall x\, \exists w : [x] \xrightarrow{*} w, |w| < \|G\|$. There are now 4 cases:

1. $[x] \to \lambda$ but $[y] \not\to \lambda$ (or viceversa), then $z = \lambda$.
2. $[x] \to a[xa]vbu \wedge [y] \to a[ya]v'cu$. Let $w : [x] \to w \wedge |w| < \|G\|$ then $z = awvbu$.
3. $[x] \to a[xa]u \wedge [y] \to a[ya]vbu$ (or viceversa). As G is advanced $L([xa]) \not\subset \Sigma^* b$. So $\exists w \in L([xa]) \setminus \Sigma^* b : |w| < \|G\|$, and $z = awu$.
4. $([x] \to \lambda \iff [y] \to \lambda) \wedge ([x] \to a[xa]u \iff [y] \to a[ya]u)$. As $L([x]) \neq L([y])$ there exists a string w such that for $[xw]$ and $[yw]$ we are in one of the previous cases. Denote by $w_{i,j}$ the substring of w from position i to position j and let $|w| = n$. So $[u] \xrightarrow{k} w_{1,k}[uw_{1,k}]\,\mathrm{tail}_L(uw_{1,k})$. But if $n > |V|$ it has to exist i, j such that $[xw_{1,i}] = [xw_{1,j}]$ and $[yw_{1,i}] = [yw_{1,j}]$ and then $[xw_{1,i}w_{j+1,n}] = [xw]$ and $[yw_{1,i+1}w_{j,n}] = [yw]$. Then $|w| \leq |V|^2$ and $|\,\mathrm{tail}_L(uw)| \leq \|G\|^2$

With $O((|\,\mathrm{Sp}_L| \cdot |\Sigma|)^2)$ strings, a CS can be constructed. Furthermore to comply with conditions 1, 2 and 4 of definition 14, these strings can be linearly bounded. The length of the strings necessary for the third condition can be quadratically bounded by lemma 5. If more correctly labeled strings are added to the sample, the sample remains characteristic for this language.

We will now prove that provided we have a characteristic sample, functions next, right, equiv and isfinal return the same result as when having access to an oracle.

Lemma 6. *Let S be a characteristic sample of a* DL *language L and let $x \in \mathrm{Sp}_L$, then* $\mathrm{next}(x) = \{xa : xay \in L\}$.

Proof. Clear by definition 14(1).

Lemma 7. *Let S be a characteristic sample of a* DL *language L, let $x \in \mathrm{Sp}_L$ and $a \in \Sigma$ such that* $\mathrm{CSF}_L(xa) \neq \emptyset$, *then* $\mathrm{right}(xa) = \mathrm{lcs}(a^{-1}\,\mathrm{CSF}_L(x))$.

Proof. $\mathrm{right}(xa) = (\mathrm{tail}_{S_+}(xa))(\mathrm{tail}_{S_+}(x))^{-1} = (\mathrm{tail}_L(xa))(\mathrm{tail}_L(x))^{-1} = \mathrm{lcs}(a^{-1}\,\mathrm{CSF}_L(x))$ by definition of function right, definition 14(2) and lemma 4(1).

Lemma 8. *Let S be a characteristic sample of a* DL *language L, let $x, y \in \mathrm{Sp}_L$ and $a \in \Sigma$ such that* $\mathrm{CSF}_L(xa) \neq \emptyset$, *then* $\mathrm{equiv}(xa, y) \iff xa \equiv_L y$.

Proof. Suppose $xa \not\equiv_L y$, by definition 14(3) there is a string v such that:
$xav\,\mathrm{tail}_L(xa) \in S_+ \wedge yv\,\mathrm{tail}_L(y) \in S_- \;\vee\; yv\,\mathrm{tail}_L(y) \in S_+ \wedge xav\,\mathrm{tail}_L(xa) \in S_-$
Suppose the first case holds (the second is similar); by definition 14(2) $\exists v :$ $xav\,\mathrm{tail}_{S_+}(xa) \in S_+ \wedge yv\,\mathrm{tail}_{S_+}(y) \in S_-$ and then $\mathrm{equiv}(xa, y)$ evaluates to false.

Conversily suppose $xa \equiv_L y$ ($L_{G_L}([xa]) = L_{G_L}([y])$) and that $\mathrm{equiv}(xa, y)$ evaluates to false. Then there exists $v : xav\,\mathrm{tail}_{S_+}(xa) \in S_+ \wedge yv\,\mathrm{tail}_{S_+}(y) \in S_-$ (the other case is similar). Then, by definition 14(2) we have that $xav\,\mathrm{tail}_L(xa) \in S_+ \subseteq L$, and by lemma 4(2) follows that $v \in L([xa]) = L([y])$. Again by lemma 4(2) $yv\,\mathrm{tail}_L(y) \in L$ and we have a contradiction.

Lemma 9. *Let S be a characteristic sample of a* DL *language L, let $x \in \mathrm{Sp}_L$, then* $\mathrm{isfinal}(x) \iff \lambda \in \mathrm{CSF}_L(x)$

Proof. $\mathrm{isfinal}(x) \iff x\,\mathrm{tail}_{S_+}(x) \in S_+ \iff x\,\mathrm{tail}_L(x) \in S_+ \subseteq L \iff \lambda \in \mathrm{CSF}_L(x)$ respectively by definitions of function isfinal, 14(2) and lemma 4(2).

It is easy to see that the algorithm is going to compute $O(|\mathrm{Sp}|)$ times the function isfinal, $O(|\mathrm{Sp}| \cdot |\Sigma|)$ times the functions next and right, and $O(|\mathrm{Sp}|^2 \cdot |\Sigma|)$ times the function equiv. As all those functions can be calculated in $O(\|S\|)$ and, in the worst case, $|\mathrm{Sp}|$ can be as big as $\|S_+\|$, the whole complexity will be $O(\|S\|^3 \cdot |\Sigma|)$.

When the sample is not characteristic one of three things may happen:

- the obtained grammar may be consistent with the sample;
- the obtained grammar may not be able to parse conveniently all the strings;
- no merge is possible and the results is not even a deterministic linear grammar

Only the first case seems to comply with the first condition of definition 6, *i.e.* that the result is consistent with the sample. Nevertheless, as any finite language is clearly DL, and a trivial DL grammar for such a language can be constructed in polynomial time (for instance a regular grammar), we can call algorithm 2, check if the result is consistent with the sample, and if not, return the trivial DL grammar generating exactly the positive strings in the sample.

By doing so the procedure complies with both conditions of definition 6.

Theorem 4. DL *languages, when represented by* Gen-DL *or* DL *grammars, are identifiable in the limit from polynomial time and data.*

Proof. ¿From theorem 3 we have the fact that any Gen-DL or DL grammar admits an equivalent canonical advanced DL grammar of polynomial size. The corresponding characteristic set is also of polynomial size, and algorithm 2 returns a grammar complying with definition 6. Finally lemmas 6-9 prove that the characteristic set is sufficient for the grammar to be identified.

5 Summary and Open Questions

We have described a type of grammars that generate a large class of language including regular languages, palindrome languages, linear LL(1) languages and other typical linear languages as $\{a^n b^n, 0 \leqslant n\}$. This class is proved to admit a canonical form, that even if not minimal, is at most polynomially larger than any equivalent grammar in the class. By providing an adequate algorithm, we proved that this class was polynomially identifiable in the limit from polynomial tome and data.

Further work should consist in:

- Algorithm 2 is no good when it is given a learning sample that is not characteristic. An obvious extension is for it to return in that case a trivial grammar. This does allow identification in the limit from polynomial time an data, but is not satisfactory. For it to be of practical use, it should also generalize in that case. The proposed extension would be of the same sort as that of *RPNI*[3] [1] on Gold's original algorithm [20].
- Learning in this model is directly linked with learning in the simple-*PAC* setting [22]. Links with active learning are more unclear. It can be proved that there are classes that can be learned through polynomially many membership and equivalence queries and that are not identifiable in the limit from polynomial time and data. Of the converse nothing is known. A polynomial algorithm to learn deterministic linear grammars from a *MAT* [4] seems difficult.
- Our definition of linear deterministic grammars, although less restrictive than that of [16, 17], does not fully capture the one by deterministic one turn pda. The learnability of this class of languages (which is also the one of linear LR(1) languages) is a first open problem.

[3] Regular Positive and Negative Inference

[4] Minimally Adequate Teacher [13]

- Clearly determinism can be defined from the left (as in this work) or from the right. A natural extension of this work is that linear right-deterministic grammars can be identified in the limit from polynomial time and data. An open problem is that of simultaniously inferring linear right and left deterministic grammars.
- The natural extension of linear languages is that of multilinear languages [23], which correspond to k-turn automata. These languages do not have the expansiveness problem, so would be a natural candidate for non linear languages to be learnable.

References

1. Oncina, J., García, P.: Identifying regular languages in polynomial time. In Bunke, H., ed.: Advances in Structural and Syntactic Pattern Recognition. Volume 5 of Series in Machine Perception and Artificial Intelligence. World Scientific (1992) 99–108
2. Pitt, L., Warmuth, M.: The minimum consistent DFA problem cannot be approximated within any polynomial. Journal of the Association for Computing Machinery **40** (1993) 95–142
3. Kearns, M., Valiant, L.: Cryptographic limitations on learning boolean formulae and finite automata. In: 21st ACM Symposium on Theory of Computing. (1989) 433–444
4. Sakakibara, Y.: Recent advances of grammatical inference. Theoretical Computer Science **185** (1997) 15–45
5. Lee, S.: Learning of context-free languages: A survey of the literature (1996)
6. de la Higuera, C.: Characteristic sets for polynomial grammatical inference. Machine Learning **27** (1997) 125–138
7. Sempere, J., García, P.: 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, Springer Verlag (1994) 38–44
8. Takada, Y.: A hierarchy of language families learnable by regular language learners. In: Grammatical Inference and Applications, ICGI'94. Number 862 in Lecture Notes in Artificial Intelligence, Springer Verlag (1994) 16–24
9. Sakakibara, Y., Kondo, M.: Ga-based learning of context-free grammars using tabular representations. In: Proceedings of 16th International Conference on Machine Learning (ICML-99). (1999) 354–360
10. Sakakibara, Y.: Learning context-free grammars from structural data in polynomial time. Theoretical Computer Science **76** (1990) 223–242
11. Sakakibara, Y., Brown, M., Hughley, R., Mian, I., Sjolander, K., Underwood, R., Haussler, D.: Stochastic context-free grammars for trna modeling. Nuclear Acids Res. **22** (1994) 5112–5120
12. Ishizaka, I.: Learning simple deterministic languages. In: Proceedings of COLT 89. (1989)
13. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Control **39** (1987) 337–350
14. Pitt, L.: Inductive inference, DFA's, and computational complexity. In: Analogical and Inductive Inference. Number 397 in Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin (1989) 18–44

15. Autebert, J., Berstel, J., Boasson, L.: Context-free languages and pushdown automata. In Salomaa, A., Rozenberg, G., eds.: Handbook of Formal Languages. Volume 1, Word Language Grammar. Springer-Verlag, Berlin (1997) 111–174
16. Nasu, M., Honda, N.: Mappings induced by pgsm-mappings and some recursively unsolvable problems of finite probabilistic automata. Information and Control **15** (1969) 250–273
17. Ibarra, O.H., Jiang, T., Ravikumar, B.: Some subclasses of context-free languages in nc1. Information Processing Letters **29** (1988) 111–117
18. Holzer, M., Lange, K.J.: On the complexities of linear $ll(1)$ and $lr(1)$ grammars. In: Proceedings of the 9th International Conference on Fundamentals of Computation Theory. Number 710 in Lecture Notes in Computer Science, Springer Verlag (1993) 299–308
19. Valiant, L.: A theory of the learnable. Communications of the Association for Computing Machinery **27** (1984) 1134–1142
20. Gold, E.: Complexity of automaton identification from given data. Information and Control **37** (1978) 302–320
21. Parekh, R., Honavar, V.: Learning dfa from simple examples. Machine Learning Journal **44** (2001) 9–35
22. Parekh, R., Honavar, V.: On the relationship between models for learning in helpful environments. In Oliveira, A., ed.: Proceedings of ICGI 2000. Volume 1891 of Lecture Notes in Artificial Intelligence., Berlin Heidelberg New York, Springer Verlag (2000) 207–220
23. Ginsburg, S., Spanier, E.: Finite-turn pushdown automata. SIAM Control **4** (1966) 429–453