On Sufficient Conditions to Identify in the Limit Classes of Grammars from Polynomial Time and Data

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,
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,
home page: http://www.dlsi.ua.es/~oncina

Abstract. Linearity and determinism seem to be two essential conditions for polynomial learning of grammars to be possible. We propose a general condition valid for certain subclasses of the linear grammars given which these classes can be polynomially identified in the limit from given data. This enables us to give new proofs of the identification of well known classes of grammars, and to propose a new (and larger) class of linear grammars for which polynomial identification is thus possible.

Keywords: Polynomial identification in the limit, linear grammars.

1 Introduction

The main focus of research in the field of grammatical inference has been set on learning regular grammars or deterministic finite automata (*dfa*). Algorithms have been provided dealing with learning such grammars in a variety of settings: when both examples and counter-examples are provided [OG92,dlH97,LPP98], or when the learning algorithm is allowed to question some teacher [Ang87]. Reasons justifying that most attention has been spent on this class of grammars are that this problem may seem simple enough but theoretical results make it already too hard for usual *Machine Learning* settings [KV89,PW93]; on the other hand the class of *dfa* seems to be in some way maximal for certain forms of polynomial learning. In the framework of *active learning* it is known that *dfa* can be inferred through a polynomial use of a *Minimal Adequate Teacher* [Ang87]. It is conjectured [Ang01] not to be the case for richer classes. In the setting of learning from a bunch of examples, this is even clearer as *dfa* can be polynomially identified from time and data, but not context-free grammars or non-deterministic finite automata.

Yet, even if all this seems to induce that the harder question of learning context-free grammars may only yield negative answers, there are many practical issues that induce us to continue research in this direction. Context-free grammars are much more general than regular grammars and can model many situations that are not within reach of the regular patterns: stacks, palindromes, parenthesis... Grammatical inference will only be able to tackle a certain number of applications if it can cope with some form of context-free grammars. This is probably even more the case with recent applications such as:

- XML grammars [Fer01]. XML documents can be described by special sorts of context-free grammars. Learning such grammars would enable to find specific documents type definitions for certain XML documents.
- In computational linguistics it is generally argued that the structure of natural language cannot be represented by automata, and that at least context-free grammars are needed [Sto95].
- There is a variety of problems linking grammatical inference with bio-informatics [BJVU98]. Problems related with secondary structure require a tree-like representation [Sak90], involving context-free grammars.

A survey on grammatical inference (with special interest on context-free grammar learning) can be found in [Sak97] and a specific survey covering a wide range of context-free grammar learning results can be found in [Lee96]. In order to extend positive results in grammatical inference to sub-classes of context-free grammars, one should analyze the reasons the main algorithms work:

- Determinism of the grammar is a necessary condition: in terms of grammars, this will be called *strong unambiguity*: at any one point of a parse, at most one rule should be applicable. This condition ensures that merging non-terminals wrongfully can be refuted.
- Linearity of the grammars is also an essential feature: it is easy [dlH97] to construct very simple non linear grammars that only generate exponentially (in the size of the grammar) long strings. Learning such grammars requires teaching sets too large to be considered.

Combining these two conditions with a third condition corresponding roughly to the existence of some constructive filter on the shape of the production rules, we obtain a sufficient condition for subclasses of linear grammars to be learned. We use here as learning paradigm that of *identification in the limit from polynomial time and data*: the learning algorithm must run in time polynomial in the learning data and there must always exist a data set of polynomial size from which identification is achieved. Moreover this is monotonic, in the sense that the inclusion of this *characteristic set* in the learning sample is sufficient.

The existence of sufficient conditions for polynomial learning to be possible has been shown in other settings: for active learning [HPRW96] conditions are given for a polynomial number of membership and equivalence queries to be sufficient for identification; in the context of learning from text (only positive examples are available) [Fer00] proposes the existence of distinguishing functions to define sub-classes of regular languages that could be identifiable.

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 $a, b, c \ldots$, strings by $u, v, \ldots, z$, 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 $Lu^{-1} = \{v : vu \in L\}$; $u^{-1}Lv^{-1} = \{w : uwv \in L\}$. The symmetric difference between two languages L_1 and L_2 will be denoted $L_1 \ominus L_2$.

A *context-free grammar* G is a quadruple $\langle \Sigma, V, P, S \rangle$ where Σ is an alphabet, 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. Unless stated otherwise, strings in $(\Sigma \cup V)^*$ are denoted by $\alpha, \beta, \ldots$.

We will denote $uTv \rightarrow uwv$ when $(T \rightarrow w) \in P$. $\xrightarrow{*}$ is the reflexive and transitive closure of $\rightarrow$. We denote by $L_G(T)$ the language $\{w \in \Sigma^* : T \xrightarrow{*} w\}$. When T is the axiom, we will also denote the language $L(G)$. Two grammars are equivalent if they generate the same language. A context-free grammar $G = \langle \Sigma, V, P, S \rangle$ 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 [Pit89]. We will consider here the size of G denoted $\|G\| = \sum_{(T,\alpha) \in P(G)} (|\alpha| + 1)$.

Example 1. We will consider as a running example the class of even linear grammars, *i.e.* those grammars[3] for which $P \subset V \times \Sigma^* \cup \Sigma V \Sigma$. The fact that this class could be identified from polynomial time and data has been proved by [Tak88,SG94]. We will use the following normal form for the grammars: $P \subset V \times \Sigma \cup \{\lambda\} \cup \Sigma V \Sigma$ and $(T \rightarrow aT'b) \in P \wedge (T \rightarrow aT''b) \in P \implies T' = T''$.

The learning model we will be working in (and which will be properly defined in the next sub-section) considers both examples of the language to be learned and counter-examples.

A *sample* X is a pair of sets (X_+, X_-) where $X_+ \cap X_- = \emptyset$. $\mathbb{S}$ is the set of all finite samples over Σ. If $X, X' \in \mathbb{S}$, we will write $X \subseteq X'$ if $X_+ \subseteq X'_+$ and $X_- \subseteq X'_-$

$X \in \mathbb{S}$ is a sample of a language L if $X \subseteq (L, \Sigma^* \setminus L)$. The set of all samples of L is denoted $\mathbb{S}_L$ or $\mathbb{S}_G$ when G is a grammar generating language L. Let $\mathcal{G}$ a

[3] Other characterisations are possible, but lead to the same class.

class of grammars, $\mathbb{S}_{\mathcal{G}}$ is the set of all samples of languages that can be generated by grammars in $\mathcal{G}$.

Different learning paradigms have been examined and used in the literature of machine learning [Val84,Gol78]. We will use identification in the limit from polynomial time and data [dlH97] as our learning model.

Definition 1. *A class $\mathcal{G}$ of grammars is identifiable in the limit from polynomial time and data if there exist two polynomials $p()$ and $q()$ and an algorithm $\mathcal{A}$ such that:*

1. *for each grammar G in $\mathcal{G}$, there exists a characteristic sample $(X_{c,+}, X_{c,-})$ of size less than $q(\|G\|)$ for which, if $X_{c,+} \subseteq X_+$, $X_{c,-} \subseteq X_-$, $\mathcal{A}$ returns a grammar G' equivalent with G;*
2. *given any sample (X_+, X_-) of some language $L(G)$ with G in $\mathcal{G}$, $\mathcal{A}$ returns a grammar G in $\mathcal{G}$ compatible with (X_+, X_-) in $O(p(\|X_+\| + \|X_-\|))$ time.*

With this definition it is known that deterministic finite automata [OG92] are identifiable in the limit from polynomial time and data whereas non-deterministic finite automata and linear (and hence context-free) grammars are not [dlH97].

3 Results

3.1 About Compatibility and Consistency

In a state-merging algorithm the basic operation consists in knowing if 2 states (or non-terminals) can be merged or not. A merge may be refused for different reasons: because there is no way, once the merge is done, to reach any grammar in the class (compatibility), or to reach some grammar consistent with the learning sample.

Definition 2 (Strongly unambiguous grammar). *Let $p \in P, Z_p = \{w\}$ if $p = (A \rightarrow w)$, $Z_p = u\Sigma^*v$ if $p = (A \rightarrow uBv)$. A linear grammar G is strongly unambiguous if $\forall (A \rightarrow \alpha), (A \rightarrow \beta) \in P$ $Z_{(A\rightarrow\alpha)} \cap Z_{(A\rightarrow\beta)} = \emptyset \iff \alpha \neq \beta$.*

That is, during the parse of any string, at any point there is at most one rule that can apply.

Example 2. Consider $G = \langle \{a, b\}, \{S\}, \{S \rightarrow aSa, S \rightarrow bSb, S \rightarrow \lambda\}, S\rangle$ (a grammar for even palindromes). G is strongly unambiguous: a string is either λ, and will be parsed if possible by the appropriate rule, or of non null length where again there is at most one rule that applies. This is true in general for any grammar in the normal form (example 1).

Lemma 1. *Let G be a strongly unambiguous grammar, let $A, B \in V : L(A) \neq L(B)$ then $\exists x \in L(A) \ominus L(B) : |x|$ is linear with $\|G\|$.*

Proof. Let us construct the grammar G_d in the following way:
$\forall A \to a_1 \dots a_n B b_1 \dots b_k \in P_G$ add the rules $A_{a_1 \dots a_{i-1}} \to a_i A_{a_1 \dots a_i}$ $1 \le i \le n$, and $A_{a_1 \dots a_n b'_1 \dots b'_{i-1}} \to b'_i A_{a_1 \dots a_n b'_1 \dots b'_i}$ $1 \le i \le k$ to G_d, and $\forall A \to a_1 \dots a_n \in P_G$ add the rules $A_{a_1 \dots a_{i-1}} \to a_i A_{a_1 \dots a_i}$ $1 \le i \le n$ $A_{a_1 \dots a_n} \to \lambda$, where the symbols b' are new symbols, each one corresponding to each symbol $b \in \Sigma$ and $A_\lambda = A$ and $A_{a_1 \dots a_n b'_1 \dots b'_k} = B$.

As the grammar G is strongly unambiguous, the associated grammar G_d is a deterministic regular grammar (the grammar representation of a *dfa*, see section 4.1 for details). Moreover, for each $x \in L(G)$ we can associate a single string $x' \in L(G_d)$ obtained by using the rules corresponding to those used during the parse of x. The converse also holds, so $x \in L(G) \iff x' \in L(G_d)$.

Let $A, B \in V_G : L(A) \neq L(B)$, as G_d is regular and deterministic we know that $\exists w' \in L_{G_d}(A) \ominus L_{G_d}(B)$ such that $|w'|$ is linear with $\|G\|$, then the corresponding string $w \in L_G(A) \ominus L_G(B)$, and $|w| = |w'|$.

Grammars may only partially parse strings: what is left will be called unparsed and can be linearly computed:

Definition 3 (Unparsed strings). *Let $Y \subseteq \Sigma^*$, let G be a strongly unambiguous linear grammar and let $A \in V_G$, then* $\texttt{unparsed}(Y, G, A) = \{uxv : u'uxvv' \in Y \wedge S \xrightarrow{*} u'Av' \wedge \nexists A \to uBv \in P_G\}$.

Definition 4 (Related states). *Let G, G' be two strongly unambiguous linear grammars, non-terminal A of G' is related with the non-terminal B of G ($B = \mathrm{r}(A)$) if $\forall \{u_i\}, \{v_i\} : S_{G'} \to_{G'} u_1 A_1 v_1 \to_{G'} \cdots \to_{G'} u_n A v_n$ then $S_G \to_G u_1 B_1 v_1 \to_G \cdots \to_G u_n B v_n$.*

For a given grammar it is possibly to consider the grammars that may lead, through merges to this grammar. Such grammars are composed of 2 types of non-terminals: *kernel* ones (in $\bar{K}$) that constitute the core of the grammar, and *frontier* ones (in $\bar{F}$) that are only used in the right-hand of one rule:

Definition 5. *$\bar{G} = \langle \Sigma, \bar{K}, \bar{F}, \bar{P}, \bar{S} \rangle$ is a* half-way *grammar of the strongly unambiguous linear grammar $G = \langle \Sigma, N, P, S \rangle$ ($\bar{G} \in \mathrm{hw}(G)$) if:*

1. $\forall A \in \bar{K}, \exists u, v : \bar{S} \xrightarrow{*} uAv$
2. $\forall A \in \bar{K},\ A \to uBv \in \bar{P} \iff \mathrm{r}(A) \to u\,\mathrm{r}(B)v \in P$
3. $\forall B \in \bar{F}\ \exists! A \in \bar{K} : A \to uBv \in \bar{P}$
4. $\forall B \in \bar{F}\ \nexists B \to uCv \in \bar{P}$

We shall denote $\bar{V} = \bar{K} \cup \bar{F}$, and use $\bar{G}$ as a linear grammar when convenient.

Definition 6. *By extension, let $\mathcal{G}$ be a class of strongly unambiguous linear grammars, $\bar{G} \in \mathrm{hw}(\mathcal{G})$ if $\exists G \in \mathcal{G} : \bar{G} \in G$.*

Example 3. $\bar{G} = \langle \{a, b\}, \{\bar{S}\}, \{\bar{T}\}, \{\bar{S} \to a\bar{T}a, \bar{S} \to b\bar{S}b\}, \bar{S} \rangle$ is a half-way grammar of G (from example 2), with $\bar{S}$ in the kernel and $\bar{T}$ as a frontier non-terminal. Let $X_+ = \{abba, baab, bb\}$ and $X_- = \{aba, baaa, bbab\}$. $\texttt{unparsed}(X_+, \bar{G}, \bar{T}) = \{bb, \lambda\}$

Definition 7 (merge function). *Let G an strongly unambiguous linear grammar, Let $\bar{G} \in \mathrm{hw}(G)$, let $A \in \bar{F}, B \in \bar{K}$ then, $\bar{G}' = \mathtt{merge}(\bar{G}, B, A)$ if*

1. $\bar{\Sigma}' = \bar{\Sigma}$
2. $\bar{K}' = \bar{K}$
3. $\bar{F}' = \bar{F} \setminus \{A\}$
4. $\bar{S}' = \bar{S}$
5. $\bar{P}' = \bar{P} \setminus \{C \to uAv\} \cup \{C \to uBv\}$

Note that as $\bar{G} \in \mathrm{hw}(G)$, if $A \in \bar{F}$, non-terminal C of point 5 is unique.

Lemma 2. *Let $\bar{G} \in \mathrm{hw}(G)$, $A \in \bar{F}, B \in \bar{K} : \mathrm{r}(A) = \mathrm{r}(B)$ then $\mathtt{merge}(\bar{G}, B, A) \in \mathrm{hw}(G)$.*

Proof. Let $\bar{G}' = \mathtt{merge}(\bar{G}, B, A)$. As $A \in \bar{F}$ we know, by definition of half-way (def. 5.3) that $\exists! C \in \bar{K} : C \to uAv \in \bar{P}$. Then, by 5.2 we know that $\mathrm{r}(C) \to u\,\mathrm{r}(A)v \in P$, and as $\mathrm{r}(B) = \mathrm{r}(A)$, $\mathrm{r}(C) \to u\,\mathrm{r}(B)v \in P$. Then substituting the rule $C \to uAv \in \bar{P}$ by $C \to uBv$ implies that 5.2 is met.

Definition 8 (Consistent with a sample). *Let $\bar{G} \in \mathrm{hw}(\mathcal{G})$, let $X \in \mathbb{S}_G$, $\bar{G}$ is $\mathtt{consistent}$ with the sample X ($\mathtt{consistent}(X, \bar{G})$) if $\forall A \in \bar{V}$, $\mathtt{unparsed}(X_+, \bar{G}, A) \cap \mathtt{unparsed}(X_-, \bar{G}, A) = \emptyset$.*

Lemma 3. *Let $\bar{G} \in \mathrm{hw}(G)$ and $X \subseteq \mathbb{S}_G$ then $\mathtt{consistent}(X, \bar{G})$.*

Proof. By contradiction.

Example 4. $\bar{G} = \langle \{a, b\}, \{\bar{S}\}, \{\bar{T}\}, \{\bar{S} \to a\bar{T}a, \bar{S} \to b\bar{S}b\}, \bar{S} \rangle$ is consistent with sample $X_+ = \{abba, baab, bb\}$ and $X_- = \{aba, baaa, bbab\}$.

Definition 9 (Compatible non-terminals). *Let $\bar{G} \in \mathrm{hw}(\mathcal{G})$, let $A \in \bar{F}, B \in \bar{K}$, we say that the non-terminals A and B are $\mathtt{compatible}$ (and write $\mathtt{compatible}(\bar{G}, B, A)$) if $\mathtt{merge}(\bar{G}, B, A) \in \mathrm{hw}(\mathcal{G})$.*

By extension: Let $G \in \mathcal{G}$, non-terminals $A, B \in V$ are $\mathtt{compatible}$ in G if $\forall \bar{G} \in \mathrm{hw}(G) \exists A' \in \bar{F}, \mathrm{r}(A') = A, \exists B' \in \bar{K}, \mathrm{r}(B') = B \Rightarrow \mathtt{compatible}(\bar{G}, B', A')$.

Example 5. For the class of even linear grammars `compatible` is a trivial function that always returns true.

Definition 10 (Non-redundant grammar in a class). *A grammar G is non-redundant in the class $\mathcal{G}$ if $\forall A, B \in V,\ A \neq B,\ \mathtt{compatible}(G, A, B) \Rightarrow L(A) \neq L(B)$.*

Lemma 4. *Let G be a non-redundant strongly unambiguous linear grammar in the class $\mathcal{G}$, with $\bar{G} \in \mathrm{hw}(G)$ and $A \in \bar{F}, B \in \bar{K}$ then $\exists X_{\bar{G},B,A} \in \mathbb{S}_G : \forall X \in \mathbb{S}_G,\ X_{\bar{G},B,A} \subseteq X :$*

$$\mathrm{r}(A) = \mathrm{r}(B) \iff \mathtt{compatible}(\bar{G}, B, A) \wedge \mathtt{consistent}(X, \bar{G}).$$

Moreover, $\|X_{\bar{G},B,A}\|$ is linear in $\|G\|$.

Proof. There are 2 cases:

$\mathrm{r}(A) = \mathrm{r}(B) \Rightarrow$ $\texttt{compatible}(\bar{G}, B, A) \wedge \texttt{consistent}(X, \texttt{merge}(\bar{G}, B, A))$. By lemma 2 we have that $\texttt{merge}(\bar{G}, B, A) \in \mathrm{hw}(G)$, so $\texttt{compatible}(\bar{G}, B, A)$ and by lemma 3 $\forall X \in \mathbb{S}_G$, $\texttt{consistent}(X, \texttt{merge}(\bar{G}, B, A))$.

$\mathrm{r}(A) \neq \mathrm{r}(B) \Rightarrow \neg\, \texttt{compatible}(\bar{G}, B, A) \vee \neg\, \texttt{consistent}(L, \texttt{merge}(\bar{G}, B, A))$.
We are going to show that $\exists X_{\bar{G},B,A} \in \mathbb{S}_G : \forall X \in \mathbb{S}_G, X_{\bar{G},B,A} \subseteq X$
$\mathrm{r}(A) \neq \mathrm{r}(B) \wedge \texttt{compatible}(\bar{G}, B, A) \Rightarrow \neg\, \texttt{consistent}(L, \texttt{merge}(\bar{G}, B, A))$.
Let $\bar{G}' = \texttt{merge}(\bar{G}, B, A)$. As G is non-redundant in the class, by definition of non-redundancy (def. 10), we know that $L(\mathrm{r}(A)) \neq L(\mathrm{r}(B))$. Then $\exists x \in L(\mathrm{r}(A)) \ominus L(\mathrm{r}(B))$, suppose $x \in L(\mathrm{r}(B))$ but $x \notin L(\mathrm{r}(A))$ (without loss of generality; the other case is similar). Let $\bar{S} \xrightarrow{*} u_A A v_A$ and $\bar{S} \xrightarrow{*} u_B B v_B$, then let $X_{\bar{G},B,A}$ such that $u_B x v_B \in X_{\bar{G},B,A,+}$ and $u_A x v_A \in X_{\bar{G},B,A,-}$.
Let $C : \texttt{unparsed}(\{u_B x v_B\}, \bar{G}', C) \neq \emptyset$, but in the grammar $\bar{G}'$, $S \xrightarrow{*} u_A B u_A$. Then $\texttt{unparsed}(\{u_A x v_A\}, \bar{G}', C) = \texttt{unparsed}(\{u_B x v_B\}, \bar{G}', C) \neq \emptyset$ holds and $\texttt{unparsed}(X_+, \bar{G}', C) \cap \texttt{unparsed}(X_-, \bar{G}', C) \neq \emptyset \; \forall X \supseteq X_{\bar{G},B,A}$ follows. Moreover, as in the derivation of the non-terminal symbols A and B we can avoid repetitions of the intermediate non-terminals, the lengths of the strings u_A, v_A, u_B, u_B are linear in $\|G\|$. And by lemma 1 the length of x is also linear in $\|G\|$.

3.2 Guessing the Next Rules

If we are capable of refuting a given rule, all we need to do is choose the new possible rules in an economical manner. Suppose we have a function (called `next`) which takes a finite sample as input, and returns a set of pairs (u_i, v_i) with intended meaning that $\forall i : T \rightarrow u_i T' v_i$ is a possible rule, what properties does this function have to respect in order to make identification possible?

Definition 11 (`next` function). *Let* ***next*** *be a function taking as arguments a finite set of strings, a* half-way *grammar and a frontier non-terminal, and returning a finite set of pairs of strings. We say that the* ***next*** *function is* good *for a class of grammars $\mathcal{G}$ if $\forall \bar{G} \in \mathrm{hw}(\mathcal{G}), \forall A \in \bar{F}$:*

1. $\forall X_+ \subseteq \Sigma^* \forall \{(u_i, v_i), (u_j, v_j)\} \subseteq \textit{next}(X_+, \bar{G}, A) u_i \Sigma^* v_i \cap u_j \Sigma^* v_j = \emptyset \forall i \neq j$
2. $\forall X_+ \subseteq \Sigma^*$, $\forall (u, v) \in \textit{next}(X_+, \bar{G}, A), \exists uxv \in \textit{unparsed}(X_+, \bar{G}, A)$
3. $\textit{next}(X_+, \bar{G}, A)$ *works in time polynomial in* $\|X_+\|$ *and* $\|\bar{G}\|$.
4. $\exists X_{\bar{G},A} \subseteq \Sigma^* : \forall X_+ \in \Sigma^*,\ X_{\bar{G},A} \subseteq X_+ \subseteq L(G)$
$(u, v) \in \textit{next}(X_+, \bar{G}, A) \iff \exists B : \mathrm{r}(A) \rightarrow uBv \in P$
5. $\|X_{\bar{G},A}\|$ *is polynomial in* $\|G\|$.

By condition 1 function `next` can only create rules that lead to strong unambiguity. Notice that if $\mathcal{G}$ contains linear grammars that are not strongly unambiguous, no good `next` function can exist. No rules can be invented out of thin air because of condition 2. A characteristic set from which all rules can be guessed must exist (condition 4): in practice this condition with the condition

2 are easy to meet; it is sufficient to have strings that use the rules of the intended grammar. The computation of the candidate rules must obviously take place in polynomial time (condition 3), and the characteristic sets must also have polynomial time (condition 5).

Example 6. `next` can be computed for even linear grammars as $\texttt{next}(X_+, \bar{G}, A) = \{(a, b) : uawbv \in X_+ \wedge \bar{S} \xrightarrow{*} uAv\}$.

Definition 12 (good **class of grammars).** *Let $\mathcal{G}$ be a strongly unambiguous class of linear grammars, $\mathcal{G}$ is a* good *class of grammars if:*

1. *there is a* good *`next` function for the class;*
2. *there is a polynomial `compatible` function;*
3. *$\forall G \in \mathcal{G}$ there is a non-redundant grammar $G' \in \mathcal{G}$ such that $L(G) = L(G')$.*
4. *there is a polynomial algorithm $\mathcal{A}_{trivial}$ such that: $\forall X \in \mathbb{S}_{\mathcal{G}}\ \mathcal{A}_{trivial}(X) = G \in \mathcal{G} : X \subseteq (L(G), \Sigma^* \setminus L(G))$*

Example 7 (Proving that all the conditions are met for even linear grammars).

1. `next` only introduces strongly unambiguous rules (11.1);
2. pairs (a, b) introduced by `next` correspond to strings in X (definition 11.2);
3. `next` given by algorithm **??** is polynomial (11.3);
4. `next` has a polynomial characteristic set: it is sufficient to introduce one string per rule (11.3 and 11.5);
5. function `compatible` is the trivial (and thus polynomial) function that always returns `true`; (12.2)
6. there is always a non-redundant normal form which is not too large (12.3). It also can be argued that when choosing to represent these languages by more general even linear grammars, this remains true, enabling the class of grammars that is identifiable to be larger;
7. any finite set of strings is a regular language, hence also an even linear language of which a polynomial grammar can be directly constructed.

Lemma 5. *Let $\mathcal{G}$ be a* good *class of grammars, let $G \in \mathcal{G}$ and let $X \in \mathbb{S}_G$, then algorithm 1 halts and gives a strongly unambiguous linear grammar G' such that $X \in \mathbb{S}_{G'}$.*

Proof. Suppose that in an iteration, in the **while** line, we have a half-way grammar $\bar{G}$ of a strongly unambiguous linear grammar G such that $\texttt{consistent}(X, \bar{G})$.

Let us call

$$\mathrm{l}_{\#}(\bar{G}) = \sum_{\forall A \in \bar{V}} \| \texttt{unparsed}(X_+, \bar{G}, A) \|.$$

Take now a non-terminal A from F (`extract` line).

There are two cases:

1. If $\exists B \in \bar{K} : \texttt{compatible}(\bar{G}, B, A) \wedge \texttt{consistent}(X, \texttt{merge}(\bar{G}, B, A))$
Let us call $\bar{G}' = \texttt{merge}(\bar{G}, B, A)$. It is obvious that $\bar{G}'$ is a strongly unambiguous linear grammar and is consistent with the sample. Observe that once A and B are merged $|\bar{F}'| = |\bar{F}'| - 1$ and $\mathrm{l}_{\#}(\bar{G}') \leq \mathrm{l}_{\#}(\bar{G})$.

Algorithm 1 Infer

Require: $X \in \mathbb{S}_G$
Ensure: $\bar{G} = \langle \Sigma, \bar{K}, \bar{F}, \bar{S}, \bar{P} \rangle$

```
  A = newNT
  S̄ = A
  K̄ = ∅
  P̄ = ∅
  F̄ = A
  while F̄ ≠ ∅ do
    A = extract(F̄)
    if ∃B ∈ K̄ : compatible(Ḡ, B, A) ∧ consistent(X, merge(Ḡ, B, A)) then
      Ḡ = merge(Ḡ, B, A)
    else
      K̄ = K̄ ∪ {A}
      for all (u, v) ∈ next(X+, Ḡ, A) do
        B = newNT
        P̄ = P̄ ∪ {A → uBv}
        F̄ = F̄ ∪ B
      end for
    end if
  end while
  for all A ∈ K̄ do
    for all w ∈ unparsed(X+, Ḡ, A) do
      P̄ = P̄ ∪ {A → w}
    end for
  end for
```

2. If $\nexists B \in \bar{K}$: $\texttt{compatible}(\bar{G}, B, A) \wedge \texttt{consistent}(X, \texttt{merge}(\bar{G}, B, A))$
   Then a new non-terminal is added to $\bar{K}$ and the grammar is expanded in such a way that it is compatible with the sample. By definition of `next` (def. 11.1) it is a half-way grammar of a strongly unambiguous linear grammar. Let us call $\bar{G}'$ such grammar. Observe that for each rule added there are some strings in $\texttt{unparsed}(X_+, \bar{G}, A)$ and $\texttt{unparsed}(X_-, \bar{G}, A)$ (def. 11.2) that are going to use it, then $\mathrm{l}_\#(\bar{G}') < \mathrm{l}_\#(\bar{G})$ but $|\bar{F}'| \geq |\bar{F}| - 1$.

In one case $|\bar{F}|$ diminishes but $\mathrm{l}_\#(\bar{G})$ can be stable, in the other case $|\bar{F}|$ can increase but then $\mathrm{l}_\#(\bar{G})$ has to decrease. As $\mathrm{l}_\#(\bar{G})$ can not be lower than zero, there must exist an iteration when $|\bar{F}| = 0$ and the loop will stop.

At the end of the **while** loop we have a half-way grammar such that $\bar{F} = \emptyset$.

In the last **for** loop all the rules of the form $A \to w$ will be added in such a way that the entire positive sample will be accepted.

Lemma 6. *Let $\mathcal{G}$ be a* good *class of linear grammars, let $G \in \mathcal{G}$, then $\exists X_c$: $\forall X \in \mathbb{S}_G, X_c \subseteq X$, let G_X be the grammar obtained by algorithm* 1 *using X, then $L(G_X) = L(G)$. Moreover, $\|X_c\|$ is polynomial in $\|G\|$.*

Proof. We are going to show that for any grammar G from some good class $\mathcal{G}$, there exists a sample $X_c \in \mathbb{S}_G$ of size polynomial in $\|G\|$, such that from any $X \in \mathbb{S}_G : X_c \subseteq X$, algorithm 1 returns a grammar G_X such that $L(G_X) = L(G)$.

Suppose that G is non-redundant in the class. If it is not, we know that there is a non-redundant grammar in the class that accepts the same language. Suppose that in an iteration, in the **while** line, we have a half-way grammar $\bar{G} \in \mathrm{hw}(G)$.

Take now a non-terminal A from $\bar{F}$ (line 7).

If we suppose that all the $X_{\bar{G},B,A} \subseteq X_c \forall B \in \bar{K}$, by lemma 4 we know that $\texttt{compatible}(\bar{G}, B, A) \wedge \texttt{consistent}(X, \texttt{merge}(\bar{G}, B, A)) \iff \mathrm{r}(A) = \mathrm{r}(B)$.

There are now two cases:

1. $\exists B \in \bar{K} : \texttt{compatible}(\bar{G}, B, A) \wedge \texttt{consistent}(X, \texttt{merge}(\bar{G}, B, A))$.
   Then we know that $\mathrm{r}(A) = \mathrm{r}(B)$ and the algorithm merges both non-terminals obtaining a new half-way grammar of G (lemma 2) (observe that if such a non-terminal B exists, it is unique).
2. $\nexists B \in \bar{K} : \texttt{compatible}(\bar{G}, B, A) \wedge \texttt{consistent}(X, \texttt{merge}(\bar{G}, B, A))$.
   Then $\forall B \in \bar{K} : \mathrm{r}(A) \neq \mathrm{r}(B)$, that is, A is an unseen non-terminal and it is added to the $\bar{K}$ set.
   Then condition 2 of def. 5 possibly does not hold anymore.
   If `next` is good, and $X_{c,+} \subseteq X_{\bar{G},A}$, as `next` is good (def. 11) all the rules needed to meet point 2 of the half-way definition (def. 5) are added and the new non-terminals added to $\bar{F}$ are added in a way that points 3 and 4 of half-way hold.
   Then we have a new half-way grammar of G.

At the end of the **while** loop we have a half-way grammar such that $F = \emptyset$, moreover `next` function has added all the rules of the form $A \to uBv$ and then all the non-terminals of G are in $\bar{G}$.

If $\forall A \to w \in P$ we have a string $uwv \in X_{c,+}$ such that $\bar{S} \xrightarrow{*} uAv$ then, in the last **for** loop all the rules of the form $A \to w$ will be added (note that $|u|$ and $|v|$ can be linear in $\|G\|$).

In the process we have called at most $|V| \cdot |P|$ times the `compatible` function, and $|V|$ times the `next` function. As we have seen, in order to identify the grammar a number of at most $|V| \cdot |P|$ sets $X_{\bar{G},A,B}$ is needed for `compatible` to work properly, a number of $|V|$ sets $X_{\bar{G},A}$ is needed so that function `next` works properly and $|V|$ strings are also needed in order to include all the $A \to w$ rules. As all those sets have a polynomial size, there exists a characteristic sample of polynomial size.

Theorem 1. *Any* good *class of grammars is identifiable in the limit from polynomial time and data.*

Proof. Condition 1 of definition 1 is that there must be a characteristic set. Here it is a consequence of lemma 6. Now in the general case (corresponding to condition 2 of definition 1, algorithm 1 is polynomial and returns a strongly unambiguous grammar. But this grammar may be incorrect (not in $\mathcal{G}$). In this case by condition 4 of definition 12 we can run the alternative algorithm $\mathcal{A}_{trivial}$ in order to return simply a consistent solution.

4 Examples

We have provided in the previous section the elements to prove that the class of (deterministic) even linear grammars are good. We will in this section give the main lines of the proof that (deterministic) regular grammars, deterministic linear grammars and alternated linear grammars are identifiable from polynomial time and data. In the first case the result is well known [OG92], the second is a recent result [dlHO02], and the last corresponds to a class of grammars for which no result was up to now known.

4.1 Regular Grammars

Regular deterministic grammars correspond exactly to deterministic finite automata. They are regular grammars for which $P \subset \Sigma V \cup \{\lambda\}$ and $(T \rightarrow aT') \in P \wedge (T \rightarrow aT'') \in P \implies T' = T''$. Checking conditions from definitions 11 and 12 is straightforward: $\texttt{next}(X_+, \bar{G}, A) = \{(a, \lambda) : uaw \in X_+ \wedge \bar{S} \xrightarrow{*}_{\bar{G}} uA\}$.

1. `next` clearly introduces strongly unambiguous rules only (definition 11.1);
2. `next` only introduces sustained rules (definition 11.2);
3. `next` can be computed in polynomial time (definition 11.3);
4. in all contexts, a set of strings exists (one string for each a) from which `next` finds all new rules (definition 11.4 and 11.5);
5. `next` has a polynomial characteristic set (definition 11.5);
6. the class admits a `compatible` function which is polynomial: in this case it is the function that always returns true (definition 12.2);
7. there is always a non-redundant normal form which is not too large (definition 12.3);
8. any finite set of strings is a regular language (definition 12.4).

4.2 Deterministic Linear Languages

There is a variety of classes linked with deterministic one-turn pushdown automata. In [dlHO02] we study the identification of the following class:

Definition 13 (Deterministic linear grammars). *A* deterministic linear (DL) grammar $G = \langle \Sigma, V, P, S \rangle$ *is a linear grammar where all rules are of the form* $(T \rightarrow aT'u)$ *or* $(T \rightarrow \lambda)$ *and* $\forall (T \rightarrow a\alpha), (T \rightarrow a\beta) \in P \Rightarrow \alpha = \beta$ *with* $\alpha, \beta \in V\Sigma^*$.

A canonical form for these grammars is obtained by choosing systematically the longest possible suffix:

The *longest common suffix* ($\text{lcs}(L)$) of L is the longest string u such that $(Lu^{-1})u = L$.

For the class of deterministic linear languages $\texttt{next}(X_+, \bar{G}, A) = \{(a, w) : a^{-1}\,\texttt{unparsed}(X_+, \bar{G}, A) \neq \emptyset$ with $w = \text{lcs}(a^{-1}(\texttt{unparsed}(X_+, \bar{G}, A) \cap a\Sigma^*))\}$.

And again all conditions from definitions 11 and 12 hold.

4.3 Alternated Linear Languages

The class of grammars proposed in the previous sub-section can generate all regular languages and certain non regular ones (for instance palindromes, $a^n b^n$) but suffer from an asymmetrical definition. To generalize the previous class we propose:

Definition 14 (Alternated linear grammars). *An* alternated linear grammar $G = \langle \Sigma, V_E \cup V_O, P, S \rangle$ *is a linear grammar where* $V_E \cap V_O = \emptyset$, $S \in V_E$ *and*

if $T \in V_E$*:*
1. $T \rightarrow \alpha \in P \implies \alpha \in \Sigma V_O \Sigma^* \cup \{\lambda\}$
2. *if* $T \rightarrow a\alpha \in P$ *and* $T \rightarrow a\beta \in P$ *then* $\alpha = \beta$
3. *either* $T \rightarrow \lambda \in P$ *or* $|\{a : T \rightarrow aT'w \in P\}| > 1$

if $T \in V_O$*:*
1. $T \rightarrow \alpha \in P \implies \alpha \in \Sigma^* V_E \Sigma \cup \{\lambda\}$
2. *if* $T \rightarrow \alpha a \in P$ *and* $T \rightarrow \beta a \in P$ *then* $\alpha = \beta$
3. *either* $T \rightarrow \lambda \in P$ *or* $|\{a : T \rightarrow wT'a \in P\}| > 1$

A canonical form for these grammars is obtained by choosing systematically the longest possible prefix or suffix depending on the parity of the non-terminal.

The *longest common prefix* ($\mathrm{lcp}(L)$) of L is the longest string u such that $u(u^{-1}L) = L$. Compatibility is not a trivial function this time, as we have to avoid merging an *even* (V_E) non-terminal with an *odd* (V_O) one. In a half-way grammar $\bar{G}$ computing the parity of the non-terminals is straight-forward: axiom is even, and parity changes with each application of a rule.

$$\texttt{next}(X_+, \bar{G}, A) =$$
$$\begin{cases} \{(a, w) : U = a^{-1}\,\texttt{unparsed}(X_+, \bar{G}, A), U \neq \emptyset, w = \mathrm{lcs}(U)\} \text{ if } A \in V_E, \\ \{(a, w) : U = \texttt{unparsed}(X_+, \bar{G}, A)a^{-1}, U \neq \emptyset, w = \mathrm{lcp}(U)\} \text{ if } A \in V_O \end{cases}$$

Example 8. $\{(aa{+}bb)^n(c{+}d)^n : n \in \mathbb{N}\}$ is a language that cannot be generated by any regular, even or deterministic linear grammar. Yet the following alternated linear grammar generates it:
$\langle \{a, b, c, d\}, \{S, T\}, \{S \rightarrow aT + bT + \lambda, T \rightarrow aSc + aSd\}, S \rangle$ with $V_E = \{S\}$ and $V_O = \{T\}$.

To prove that the class is identifiable in the limit from polynomial time and data it is sufficient to notice that:

1. `next` only introduces strongly unambiguous rules (11.1): in both cases the function acts as in the linear deterministic one;
2. `next` only introduces pairs corresponding to strings in X_+ (definition 11.2);
3. `next` given by algorithm **??** is polynomial (11.3);
4. `next` has a polynomial characteristic set (definition 11.5): for each even accessible non-terminal A of some half-way grammar $\bar{G}$ of G, it is sufficient to introduce two strings $uavwz$ and $uav'wz$ such that $S_{\bar{G}} \xrightarrow{*}_{\bar{G}} uAz$, $\mathrm{lcs}\{v, v'\} = \lambda$ and $r(A) \rightarrow_G ar(B)w \in P$. The case where A is odd is symmetrical;

5. the class admits a `compatible` function which is polynomial (12.2);
6. there is always a non-redundant normal form which is not too large (12.3);
7. it is easy to build an alternated linear grammar that only accepts the positive sample.

5 Conclusion

We provide in this paper a reasonable way to check if some linear language class can be identified from polynomial time and data. On the other hand necessary conditions are given in [dlH97]. Between the possible follow-ups of this work we can propose:

- Adaptation of these results to classes of tree grammars is certainly feasible, but, as often is the case, technical.
- Our generic algorithm, because of its genericity, may be slow for particular classes. Speeding-up can take place by avoiding to repeatedly call functions `unparsed` and `compatible`.
- Tightening the complexity bounds would be of certain use. This may apply in a more general way to many other grammatical inference algorithms.
- The reasons for only considering linear grammars are that through them we have a direct relationship between the size of the grammar and the length of the characteristic examples. Nevertheless, even if general context-free grammars are still out of reach for our techniques, certain subclasses (such as that of multi-linear grammars) may not be. Practically the difficulty is to be able to parse in an unambiguous way (and without lookahead) strings through the rules given by the `next` function.

References

[Ang87] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Control*, 39:337–350, 1987.

[Ang01] D. Angluin. Queries revisited. In *Proceedings of ALT 2001*, pages 12–31. Springer-Verlag, 2001.

[BJVU98] A. Brazma, I. Jonassen, J. Vilo, and E. Ukkonen. Pattern discovery in biosequences. In V. Honavar and G. Slutski, editors, *Grammatical Inference, ICGI '98*, number 1433 in LNCS, pages 257–270, Berlin, Heidelberg, 1998. Springer-Verlag.

[dlH97] C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27:125–138, 1997.

[dlHO02] C. de la Higuera and J. Oncina. Learning deterministic linear languages. In R. H. Sloan J. Kivinen, editor, *Proceedings of COLT 2002*, volume 2375 of *LNCS*, pages 185–200. Springer-Verlag, 2002.

[Fer00] H. Fernau. Identification of function distinguishable languages. In S. Jain H. Arimura and A. Sharma, editors, *Proceedings of the 11th International Conference on Algorithmic Learning Theory (ALT 2000)*, volume 1968, pages 116–130, Berlin, Heidelberg, 2000. Springer-Verlag.

[Fer01] H. Fernau. Learning XML grammars. In P. Perner, editor, *Machine Learning and Data Mining in Pattern Recognition MLDM'01*, number 2123, pages 73–87. Springer-Verlag, 2001.

[Gol78] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[HPRW96] Lisa Hellerstein, Krishnan Pillaipakkamnatt, Vijay Raghavan, and Dawn Wilkins. How many queries are needed to learn? *Journal of the ACM*, 43(5):840–862, 1996.

[KV89] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. In *21st ACM Symposium on Theory of Computing*, pages 433–444, 1989.

[Lee96] S. Lee. Learning of context-free languages: A survey of the literature. Technical Report TR-12-96, Center for Research in Computing Technology, Harvard University, Cambridge, Massachusetts, 1996.

[LPP98] K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference*, number 1433 in LNCS, pages 1–12. Springer-Verlag, 1998.

[OG92] 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.

[Pit89] L. Pitt. Inductive inference, DFA's, and computational complexity. In *Analogical and Inductive Inference*, number 397 in LNCS, pages 18–44. Springer-Verlag, Berlin, 1989.

[PW93] L. Pitt and M. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. *Journal of the Association for Computing Machinery*, 40(1):95–142, 1993.

[Sak90] Y. Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76:223–242, 1990.

[Sak97] Y. Sakakibara. Recent advances of grammatical inference. *Theoretical Computer Science*, 185:15–45, 1997.

[SG94] J. M. Sempere and P. García. A characterisation of even linear languages and its application to the learning problem. In R. C. Carrasco and J. Oncina, editors, *Grammatical Inference and Applications, ICGI-94*, number 862 in LNCS, pages 38–44, Berlin, Heidelberg, 1994. Springer-Verlag.

[Sto95] A. Stolcke. An efficient probablistic context-free parsing algorithm that computes prefix probabilities. *Comp. Linguistics*, 21(2):165–201, 1995.

[Tak88] Y. Takada. Grammatical inference for even linear languages based on control sets. *Information Processing Letters*, 28(4):193–199, 1988.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the Association for Computing Machinery*, 27(11):1134–1142, 1984.