HAL open science

# Using Contextual Representations to Efficiently Learn Context-Free Languages

Alexander Clark, Rémi Eyraud, Amaury Habrard

**To cite this version:**

Alexander Clark, Rémi Eyraud, Amaury Habrard. Using Contextual Representations to Efficiently Learn Context-Free Languages. Journal of Machine Learning Research, 2010, 11, pp.2707-2744. hal-00607098

**HAL Id: hal-00607098**

**https://hal.science/hal-00607098v1**

Submitted on 7 Jul 2011

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Journal of Machine Learning Research 1 (2010) 1-40 Submitted 11/09; Published 10/00

# Using Contextual Representations to Efficiently Learn Context-Free Languages

**Alexander Clark** ALEXC@CS.RHUL.AC.UK
*Department of Computer Science,*
*Royal Holloway, University of London*
*Egham, Surrey, TW20 0EX*
*United Kingdom*

**Rémi Eyraud** REMI.EYRAUD@LIF.UNIV-MRS.FR
**Amaury Habrard** AMAURY.HABRARD@LIF.UNIV-MRS.FR
*Laboratoire d'Informatique Fondamentale de Marseille*
*CNRS UMR 6166, Aix-Marseille Université*
*39, rue Frédéric Joliot-Curie, 13453 Marseille cedex 13,*
*France*

**Editor:** editor

## Abstract

We present a polynomial update time algorithm for the inductive inference of a large class of context-free languages using the paradigm of positive data and a membership oracle. We achieve this result by moving to a novel representation, called Contextual Binary Feature Grammars (CBFGs), which are capable of representing richly structured context-free languages as well as some context sensitive languages. These representations explicitly model the lattice structure of the *distribution* of a set of substrings and can be inferred using a generalisation of distributional learning. This formalism is an attempt to bridge the gap between simple learnable classes and the sorts of highly expressive representations necessary for linguistic representation: it allows the learnability of a large class of context-free languages, that includes all regular languages and those context-free languages that satisfy two simple constraints. The formalism and the algorithm seem well suited to natural language and in particular to the modeling of first language acquisition. Preliminary experimental results confirm the effectiveness of this approach.

**Keywords:** grammatical inference, context-free language, positive data only, membership queries

## 1. Introduction

In natural language processing, many applications require the learning of powerful grammatical models. One of the central concerns of generative linguistics is the definition of an adequate formalism that needs to satisfy two different objectives. On the one hand, such a formalism must be expressive enough to describe natural languages. On the other hand, it has to be sufficiently constrained to be learnable from the sort of linguistic data available to the child learner (Chomsky, 1986). In this context, there are two possible research strategies. One is to take a descriptively adequate formalism such as *Tree Adjoining Grammars* (Joshi and Schabes, 1997) or some other mildly context sensitive grammatical

©2010 Alexander Clark, Rémi Eyraud and Amaury Habrard.

formalism and try to construct learning algorithms for that class. However, such a strategy is unlikely to be successful because classes that are so powerful are difficult to handle from a machine learning point of view. The other approach, which we adopt in this paper, consists in switching to a formalism that is in some sense intrinsically learnable, and seeing whether we can represent linguistically interesting formal languages in that representation.

Grammatical inference is the machine learning domain which aims at studying learnability of formal languages. While many learnability results have been obtained for regular languages (Angluin, 1987; Carrasco and Oncina, 1994), this class is not sufficient to correctly represent natural languages. The next class of languages to consider is the class of context-free languages (CFL). Unfortunately, there exists no learnability results for the whole class. This may be explained by the fact that this class relies on syntactic properties instead of intrinsic properties of the language like the notion of residuals for regular languages (Denis et al., 2004). Thus, most of the approaches proposed in the literature are either based on heuristics (Nakamura and Matsumoto, 2005; Langley and Stromsten, 2000) or are theoretically well founded but concern very restricted subclasses of context-free languages (Eyraud et al., 2007; Yokomori, 2003; Higuera and Oncina, 2002). Some of these approaches are built from the idea of *distributional learning*[1], normally attributed to Harris (1954). The basic principle – as we reinterpret it in our work – is to look at the set of contexts that a substring can occur in. The distribution of a substring is the linguistic way of referring to this set of contexts. This idea has formed the basis of many heuristic algorithms for learning context-free grammars (see (Adriaans, 2002) for instance). However, a recent approach by Clark and Eyraud (2007), has presented an accurate formalisation of distributional learning. From this formulation, a provably correct algorithm for context-free grammatical inference was given in the identification in the limit framework, albeit for a very limited subclass of languages, the substitutable languages. From a more general point of view, the central insight is that it is not necessary to find the non-terminals of the context-free grammar (CFG): it is enough to be able to represent the congruence classes of a sufficiently large set of substrings of the language and to be able to compute how they combine. This result was extended to a PAC-learning result under a number of different assumptions (Clark, 2006) for a larger class of languages, and also to a family of classes of learnable languages (Yoshinaka, 2008).

Despite their theoretical bases, these results are still too limited to form the basis for models for natural language. There are two significant limitations to this work: first it uses a very crude measure for determining the syntactic congruence, and secondly the number of congruence classes required will in real cases be prohibitively large. If each non-terminal corresponds to a single congruence class (the NTS languages (Boasson and Senizergues, 1985)), then the problem may be tractable. However in general the contexts of different non-terminals overlap enormously: for instance the contexts of adjective phrases and noun phrases in English both contain contexts of the form (*"it is"*, *"."*). Problems of lexical ambiguity also cause trouble. Thus for a CFG it may be the case that the number of congruence classes corresponding to each non-terminal may be exponentially large (in the

1. Note here that the word distributional does not refer to stochastic distributions, but to the occurrence of strings into contexts. The distribution of a string corresponds to all the possible contexts in which the string can appear.

size of the grammar). But the situation in natural language is even worse: the CFG itself may have an exponentially large number of non-terminals to start off with! Conventional CFGs are simply not sufficiently expressive to be cognitively plausible representations of natural language: to write a CFG requires a multiplication of the numbers of non-terminals to handle phenomena like *subject verb agreement*, *gender features*, *displaced constituents*, etc. This requires the use of a formalism like GPSG (Generalised Phrase Structure Grammar) (Gazdar et al., 1985) to write a meta-grammar — a compact way of specifying a very large CFG with richly structured non-terminals. Thus we cannot hope to learn natural languages by learning one congruence class at a time: it is vital to use a more structured representation.

This is the objective of the approach introduced in this article: for the first time, we can bridge the gap between theoretically well founded grammatical inference methods and the sorts of structured representations required for modeling natural languages.

In this paper, we present a family of representations for highly structured context-free languages and show how they can be learned. This is a paper in learning, but superficially it may appear to be a paper about grammatical representations: much of the work is done by switching to a more tractable formalism, a move which is familiar to many in machine learning. From a machine learning point of view, it is a commonplace that switching to a better representation – for example, through a non-linear map into some feature space – may make a hard problem very easy.

The contributions of this paper are as follows: we present in Section 3 a rich grammatical formalism, which we call *Contextual Binary Feature Grammars* (CBFG). This grammar formalism is defined using a set of contexts which play the role of features with a strict semantics attached to these features. Though not completely original, since it is closely related to a number of other formalisms such as Range Concatenation Grammars (Boullier, 2000), it is of independent interest. We consider then the case when the contextual features assigned to a string correspond to the contexts that the string can occur in, in the language defined by the grammar. When this property holds, we call it an *exact CBFG*. The crucial point here is that for languages that can be defined by an exact CBFG, the underlying structure of the representation relies on intrinsic properties of the language easily observable on samples by looking at context sets.

The learning algorithm is defined in Section 4. We provide some conditions, both on the context sets and the learning set, to ensure the learnability of languages that can be represented by CBFG. We prove that this algorithm can identify in the limit this restricted class of CBFGs from positive data and a membership oracle.

Some experiments are provided in Section 5: these experiments are intended to demonstrate that even quite naive algorithms based on this are efficient and effective at learning context-free languages.

Section 6 contains a theoretical study on the expressiveness of CBFG representations. We investigate the links with the classical Chomsky hierarchy, some well known grammatical representations used in natural language processing. An important result about the expressive power of the class of CBFG is obtained: it contains all the context-free languages and some non context-free languages. This makes this representation a good candidate for representing natural languages. However exact CBFG do not include all context-free languages but do include some non context-free ones, thus they are orthogonal with the

classic Chomsky hierarchy and can represent a large class of languages. This expressiveness is strengthened by the fact that exact CBFG contains most of the existing learnable classes of languages.

## 2. Basic Definitions and Notations

We begin by some standard notations and definitions used all along the paper.

We consider a finite alphabet $\Sigma$ as a finite non-empty set of symbols also called letters. A string (also called word) $u$ over $\Sigma$ is a finite sequence of letters $u = u_1 \cdots u_n$. Let $|u|$ denote the length of $u$. The set of all strings over $\Sigma$ is denoted by $\Sigma^*$, corresponding to the free monoid generated by $\Sigma$. $\lambda$ denotes the empty string and $\Sigma^+ = \Sigma^* \backslash \{\lambda\}$. A language $L$ is any subset $L \subseteq \Sigma^*$.

We will write the concatenation of $u$ and $v$ as $uv$, and similarly for sets of strings. $u \in \Sigma^*$ is a substring of $v \in \Sigma^*$ if there are strings $l, r \in \Sigma^*$ such that $v = lur$. Define $Sub(u)$ to be the set of non-empty substrings of $u$. For a set of strings $S$ define $Sub(S) = \bigcup_{u \in S} Sub(u)$.

A context is an element of $\Sigma^* \times \Sigma^*$. For a string $u$ and a context $f = (l, r)$ we write $f \odot u = lur$; the insertion or wrapping operation. We extend this to sets of strings and contexts in the natural way. We define by $Con(w) = \{(l, r) | \exists u \in \Sigma^+ : lur = w\}$ i.e. the set of all contexts of a word $w$. Similarly, for a set of strings, we define: $Con(S) = \bigcup_{w \in S} Con(w)$.

We give now a formal definition of the set of contexts since it represents a notion often used in the paper.

**Definition 1** *The set of contexts, or context distribution, of a string $u$ in a language $L$ is, $C_L(u) = \{(l, r) \in \Sigma^* \times \Sigma^* | lur \in L\}$. We will often drop the subscript where there is no ambiguity.*

**Definition 2** *Two strings $u$ and $v$ are syntactically congruent with respect to a language $L$, denoted $u \equiv_L v$, if and only if $C_L(u) = C_L(v)$.*

The equivalence classes under this relation are the *congruence* classes of the language.

After these basic definitions and notations, we recall here the definition of a context-free grammar which is a class which is close to the language class studied in this paper.

**Definition 3** *A context-free grammar (CFG) is a quadruple $G = (\Sigma, V, P, S)$. $\Sigma$ is a finite alphabet of terminal symbols, $V$ is a set of non terminals s.t. $\Sigma \cap V = \emptyset$, $P \subseteq V \times (V \cup \Sigma)^+$ is a finite set of productions, $S \in V$ is the start symbol.*

We denote a production of $P$: $N \to \alpha$ with $N \in V$ and $\alpha \in (V \cup \Sigma)^+$. We will write $uNv \Rightarrow_G u\alpha v$ if there is a production $N \to \alpha$ in $G$. $\overset{*}{\Rightarrow}_G$ denotes the reflexive transitive closure of $\Rightarrow_G$.

The language defined by a CFG $G$ is $L(G) = \{w \in \Sigma^* | S \overset{*}{\Rightarrow}_G w\}$. In the following, we will consider the CFG are represented in the *Chomsky normal form* (CNF), *i.e.* with right hand side of production rules composed of exactly two non terminals or with exactly one terminal symbol.

In general we will assume that $\lambda$ is not a member of any language.

## 3. Contextual Binary Feature Grammars (CBFG)

Distributional learning, in our view, involves explicitly modeling the distribution of the substrings of the language – we would like to model $C_L(w)$. Clearly a crucial element of this distribution is the empty context $(\lambda, \lambda)$: $(\lambda, \lambda) \in C_L(w)$ if and only if $w \in L$. Our goal is to construct a representation that allows us to recursively compute a representation of the distribution of a string $w$, $C_L(w)$, from the (representations of) the distributions of its substrings.

The representation by *contextual binary feature grammars* relies on the inclusion relation between sets of contexts of language $L$. In order to introduce this formalism, we propose, for a start, to present some preliminary results on context inclusion. These results will lead us to define a relevant representation for modeling these inclusion dependencies by the notion of *contextual binary feature grammars*.

### 3.1 Preliminary Results about Context Inclusion

The objective of this section is to give some information about contexts that will help to give an intuition about the representation. The basic insight behind CBFGs is that there is a relation between the contexts of a string $w$ and the contexts of its substrings. This is given by the following trivial lemma:

**Lemma 4** *For any language $L$ and for any strings $u, u', v, v'$ if $C(u) = C(u')$ and $C(v) = C(v')$, then $C(uv) = C(u'v')$.*

**Proof** We write out the proof completely as the ideas will be used later on. Suppose we have $u, v, u', v'$ that satisfy the conditions. If $(l, r) \in C(uv)$, then $(l, vr) \in C(u)$ and thus $(l, vr) \in C(u')$. As a consequence, $(lu', r) \in C(v)$ and then $(lu', r) \in C(v')$ which implies that $(l, r) \in C(u'v')$. Symmetrically, by using the same arguments, we can show that $(l, r) \in C(u'v')$ implies $(l, r)$ in $C(uv)$. Thus $C(uv) = C(u'v')$. ■

This establishes that the syntactic monoid $\Sigma^* / \equiv_L$ is well-defined; from a learnability point of view this means that if we want to compute the contexts of a string $w$ we can look for a split into two strings $uv$ where $u$ is congruent to $u'$ and $v$ is congruent to $v'$; if we can do this and we know how $u'$ and $v'$ combine, then we know that the contexts of $uv$ will be exactly the contexts of $u'v'$. There is also a slightly stronger result:

**Lemma 5** *For any language $L$ and for any strings $u, u', v, v'$ if $C(u) \subseteq C(u')$ and $C(v) \subseteq C(v')$, then $C(uv) \subseteq C(u'v')$.*

**Proof** See proof of Lemma 4. ■

$C(u) \subseteq C(u')$ means that we can replace any occurrence in a sentence of $u$ with a $u'$, without affecting the grammaticality, but not necessarily vice versa. Note that none of these strings need to correspond to non-terminals: this is valid for any fragment of a sentence.

We will give a simplified example from English syntax: the pronoun "*it*" can occur everywhere that the pronoun "*him*" can, but not vice versa[2]. Thus given a sentence "*I gave him away*", we can substitute "*him*" for "*it*", to get the grammatical sentence "*I gave it away*", but we cannot reverse the process. For example, given the sentence "*it is raining*", we cannot substitute "*him*" for "*it*", as we will get the ungrammatical sentence "*him is raining*". Thus we observe $C(him) \subsetneq C(it)$.

Looking at Lemma 5 we can also say that, if we have some finite set of strings $K$, where we know the contexts, then:

**Corollary 6** *For any language $L$ and for any set of strings $K$, we have:*

$$C(w) \supseteq \bigcup_{\substack{u',v':\\ u'v'=w}} \bigcup_{\substack{u\in K:\\ C(u)\subseteq C(u')}} \bigcup_{\substack{v\in K:\\ C(v)\subseteq C(v')}} C(uv).$$

This is the basis of our representation: a word $w$ is characterised by its set of contexts. We can compute the representation of $w$, from the representation of its parts $u', v'$, by looking at all of the other matching strings $u$ and $v$ where we understand how they combine (with subset inclusion). Rather than representing just the congruence classes, we will represent the lattice structure of the set of contexts using subset inclusion; sometimes called Dobrušin-domination (Marcus, 1967).

The key relationships are given by context set inclusion. *Contextual binary feature grammars* allow a proper definition of the combination of context inclusion.

### 3.2 Contextual Binary Feature Grammars

The formalism of *contextual binary feature grammars* has some resemblance with *Generalized Phrase Structure Grammar (GPSG)* (Gazdar et al., 1985), and most importantly the class of *Range Concatenation Grammars (RCG)* (Boullier, 2000); these relationships will be detailed in Section 6. As we will see later, note that our formalism defines a class orthogonal to the class of context-free grammars, indeed the use of subsets inclusion allows to model non context-free languages (although not all the context-free languages can be represented).

**Definition 7** *A Contextual Binary Feature Grammar (CBFG) $G$ is a tuple $\langle F, P, P_L, \Sigma\rangle$:*

- *$F$ is a finite set of contexts, (i.e. $F \subset \Sigma^* \times \Sigma^*$) called features, where we write $E = 2^F$ for the power set of $F$ defining the categories of the grammar, and where $(\lambda, \lambda) \in F$.*
- *$P \subseteq E \times E \times E$ is a finite set of productions that we write $x \to yz$ where $x, y, z \in E$,*
- *$P_L \subseteq E \times \Sigma$ is a set of lexical rules, written $x \to a$,*
- *$\Sigma$ denotes the alphabet.*

2. This example does not account for a number of syntactic and semantic phenomena, particularly the distribution of reflexive anaphors.

Given a CBFG $G$ we can recursively define a function $f_G$ from $\Sigma^* \to E$ as follows:

$$f_G(\lambda) = \emptyset \tag{1}$$

$$f_G(w) = \bigcup_{(c \to w) \in P_L} c \qquad \text{iff } |w| = 1 \tag{2}$$

$$f_G(w) = \bigcup_{u,v:uv=w} \bigcup_{\substack{x \to yz \in P: \\ y \subseteq f_G(u) \wedge \\ z \subseteq f_G(v)}} x \qquad \text{iff } |w| > 1. \tag{3}$$

Given a CBFG $G$ and a string $w$ it is possible to compute $f_G(w)$ in time $\mathcal{O}(|F||P||w|^3)$ using standard dynamic programming techniques. A straightforward modification of the Cocke-Kasami-Younger algorithm for parsing Context-Free Grammars will suffice.

Thus a CBFG, $G$, defines for every string $u$ a set of contexts $f_G(u)$: this will be a representation of the context distribution. $f_G(u)$ will be a subset of $F$: we will want $f_G(u)$ to approximate $C_L(u) \cap F$. The natural way to define the membership of a string $w$ in $L(G)$ is to have the context $(\lambda, \lambda) \in f_G(w)$.

**Definition 8** *The language defined by a CBFG $G$ is the set of all strings that are assigned the empty context: $L(G) = \{u | (\lambda, \lambda) \in f_G(u)\}$.*

We give here more explanation about the function $f_G$. A rule $x \to yz$ is applied to analyse a string $w$ if there is a split or *cut* of the string $w$ into two strings $u$ and $v$ such that $uv = w$ s.t. $y \subseteq f_G(u)$ and $z \subseteq f_G(v)$ — recall that $y$ and $z$ are sets of features.

One way of viewing a production $x \to yz$ is as an implication: if two strings $u$ and $v$ are such that they have the features $y$ and $z$, then their concatenation will have the features $x$. As features correspond to contexts, intuitively, the relation given by the production rule is linked with Lemma 5: $x$ is included in the set of features of $w = uv$. From this relationship, for any $(l, r) \in x$ we have $lwr \in L(G)$.

The complete computation of $f_G$ is then justified by Corollary 6: $f_G(w)$ defines all the possible contextual features associated by $G$ to $w$ with all the possible cuts $uv = w$ (*i.e.* all the possible derivations).

Note the relation between the third clause above and Corollary 6. In general we will apply more than one production at each step of the analysis.

We will discuss the relation between this class and the class of CFGs in some detail in Section 6. For the moment, we will just make the following points. First, the representation is quite close to that of a CFG where the non-terminals correspond to sets of contexts (subsets of $F$). There are, however, crucial differences: the very fact that they are represented by sets means that the non-terminals are no longer atomic symbols but rather structures; the formalism can combine different rules together at each step. Secondly, the function $f_G$ can aggregate different trees together. It is not the case that every feature assigned to $w$ must come from the same split of $w$ into $u$ and $v$. Rather some features could come from one split, and some from another: these two sets of features can be combined in a single derivation even though they come from different trees. It is this property that takes the class of languages out of the class of context-free languages. In the special case where all of the productions involve only singleton sets then this will reduce to a CFG —

the non-terminals will correspond to the individual features, and $f_G(w)$ will correspond to the set of non-terminals that can derive the string $w$.

Clearly by the definition of $L(G)$ we are forcing a correspondence between the occurrence of the context $(\lambda, \lambda)$ in $C_L(w)$ and the occurrence of the feature $(\lambda, \lambda)$ in $f_G(w)$. But ideally we can also require that $f_G$ defines exactly the possible features that can be associated to a given string according to the underlying language. Indeed, we are interested in cases where there is a correspondence between the language theoretic interpretation of a context, and the occurrence of that context as a feature in the grammar: in this case the features will be observable which will lead to learnability.

This is formalised via the following definitions.

**Definition 9** *Given a finite set of contexts* $F = \{(l_1, r_1), \ldots, (l_n, r_n)\}$ *and a language* $L$ *we can define the context feature function* $F_L : \Sigma^* \to 2^F$ *which is just the function* $u \mapsto \{(l, r) \in F | lur \in L\} = C_L(u) \cap F$.

Using this definition, we now need a correspondence between the language theoretic context feature function $F_L$ and the representation in our CBFG, $f_G$.

**Definition 10** *A CBFG* $G$ *is* exact *if for all* $u \in \Sigma^*$, $f_G(u) = F_{L(G)}(u)$.

*Example.* Let $L = \{a^n b^n | n > 0\}$. Let $\langle F, (\lambda, \lambda), P, P_L, \Sigma\rangle$ a CBFG s.t.
$F = \{(\lambda, \lambda), (a, \lambda), (aab, \lambda), (\lambda, b), (\lambda, abb)\}$. The lexical productions in $P_L$ are:
$\{(\lambda, b), (\lambda, abb)\} \to a$ and $\{(a, \lambda), (aab, \lambda)\} \to b$. Note that these lexical productions are of the form $x \to a$, where $x$ is a subset of $F$, that is to say, a set of features. The rule $\{(\lambda, b), (\lambda, abb)\} \to a$ therefore says that the letter $a$ will be assigned both of the features/contexts $(\lambda, b)$ and $(\lambda, abb)$. Since this is the only lexical rule for $a$, we will have that $f_G(a) = \{(\lambda, b), (\lambda, abb)\}$. The productions in $P$, denoted by $x \to yz$, where $x, y, z$ are again sets of contexts, are defined as: $\{(\lambda, \lambda)\} \to \{(\lambda, b)\}\{(aab, \lambda)\}$, $\{(\lambda, \lambda)\} \to \{(\lambda, abb)\}\{(a, \lambda)\}$, $\{(\lambda, b)\} \to \{(\lambda, abb)\}\{(\lambda, \lambda)\}$, $\{(a, \lambda)\} \to \{(\lambda, \lambda)\}\{(aab, \lambda)\}$.

In each of these rules, in this trivial case, the sets of contexts are singleton sets. In general, these productions may involve sets that have more than one element. This defines an exact CBFG for $L$. Indeed, the grammar assigns only $(\lambda, \lambda)$ to the elements of the language; for all elements $w$ of $\{a^n b^{n+1} : n > 1\}$ we have $f_G(w) = \{(a, \lambda)\} = F_L(w)$ and for all all elements $w$ of $\{a^{n+1} b^n : n > 1\}$, $f_G(w) = \{(\lambda, b)\} = F_L(w)$; The lexical rules assign correct contexts to each letter.

*Exact CBFGs* are a more limited formalism than CBFGs themselves; without any limits on the interpretation of the features, we can define a class of formalisms that is equal to the class of *Conjunctive Grammars* (see Section 6.2.3). However, exactness is an important property because it allows to associate the intrinsic structure of a language to the structure of the representation. Contexts are easily observable from a sample and moreover it is only when the features correspond to the contexts that distributional learning algorithms can infer the structure of the language.

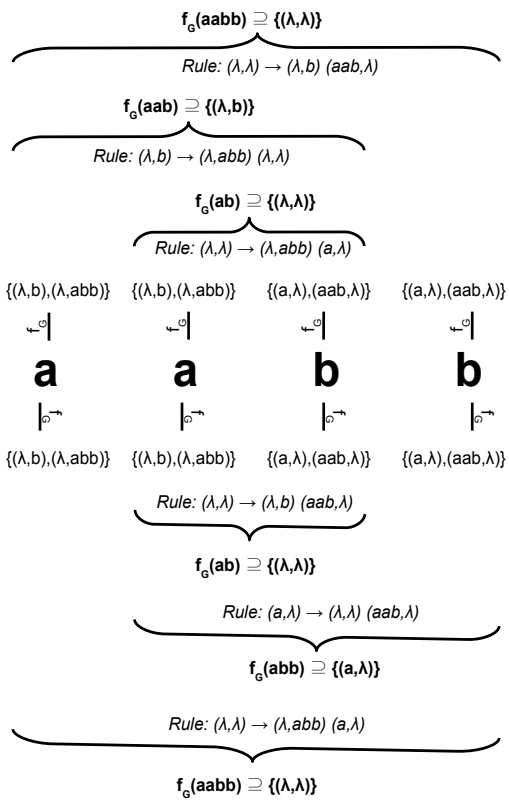


Figure 1: The two derivations to obtain $(\lambda, \lambda)$ in $f_G(aabb)$ in the grammar $G$.

### 3.3 A Parsing Example

To clarify the relationship with CFG parsing, we will give a simple worked example. Consider the CBFG $G = \langle \{(\lambda,\lambda),(aab,\lambda),(\lambda,b),(\lambda,abb),(a,\lambda)\},\ P, P_L, \{a,b\}\rangle$ with

$$P_L = \{ \quad \{(\lambda,b),(\lambda,abb)\} \to a, \quad \{(a,\lambda),(aab,\lambda)\} \to b \quad \}. \qquad \text{and} \quad P = \{ \quad \begin{array}{l} \{(\lambda,\lambda)\} \to \{(\lambda,b)\}\{(aab,\lambda)\}, \\ \{(\lambda,\lambda)\} \to \{(\lambda,abb)\}\{(a,\lambda)\}, \\ \{(\lambda,b)\} \to \{(\lambda,abb)\}\{(\lambda,\lambda)\}, \\ \{(a,\lambda)\} \to \{(\lambda,\lambda)\}\{(aab,\lambda)\} \end{array} \quad \}.$$

If we want to parse a string $w$ the usual way is to have a bottom-up approach. This means that we recursively compute the $f_G$ function on the substrings of $w$ in order to check whether $(\lambda,\lambda)$ belongs to $f_G(w)$.

For example, suppose $w = aabb$. Figure 1 graphically gives the main steps of the computation of $f_G(aabb)$. Basically there are two ways to split $aabb$ that allow the derivation of the empty context: $aab|b$ and $a|abb$. The first one corresponds to the top part of the figure while the second one is drawn at the bottom. We can see for instance that the empty context belongs to $f_G(ab)$ thanks to the rule $\{(\lambda,\lambda)\} \to \{(\lambda,abb)\}\{(a,\lambda)\}$: $\{(\lambda,abb)\} \subseteq f_G(a)$ and

$\{(a, \lambda)\} \subseteq f_G(b)$. But for symmetrical reasons the result can also be obtained using the rule $\{(\lambda, \lambda)\} \rightarrow \{(\lambda, b)\}\{(aab, \lambda)\}$.

As we trivially have $f_G(aa) = f_G(bb) = \emptyset$, since no right-hand side contains the concatenation of the same two features, an induction proof can be written to show that $(\lambda, \lambda) \in f_G(w) \Leftrightarrow w \in \{a^n b^n : n > 0\}$.

This is a simple example that illustrates the parsing of a string given a CBFG. This example does not fully express the power of CBFG since no element of the right hand side of a rule is composed of more than one context. A more complex example, corresponding to a context-sensitive language, will be presented in Section 6.1.3.

We stop here the presentation of the CBFG formalism and we present our learning algorithm in the next section. However, if the reader wishes to become more familiar with CBFGs a study on their expressiveness is provided in Section 6.

## 4. Learning Algorithm

We have carefully defined the representation so that the inference algorithm will be almost trivial. Given a set of strings, and a set of contexts, we can simply write down a CBFG that will approximate a particular language.

### 4.1 Building CBFGs from Sets of Strings and Contexts

**Definition 11** *Let $L$ be a language, $F$ be a finite set of contexts such that $(\lambda, \lambda) \in F$, $K$ a finite set of strings, $P_L = \{F_L(u) \rightarrow u | u \in K \wedge |u| = 1\}$ and $P = \{F_L(uv) \rightarrow F_L(u)F_L(v) | u, v, uv \in K\}$. We define $G_0(K, L, F)$ as the CBFG $\langle F, P, P_L, \Sigma\rangle$.*

Often $K$ will be closed under substrings: i.e. $Sub(K) = K$. This grammar is a CBFG, since $K$ and $F$ are finite, and so $P$ and $P_L$ are too by construction. In general it will not be exact.

We will call $K$ here the *basis* for the language. The set of productions is defined merely by observation: we take the set of all productions that we observe as the concatenation of elements of the small set $K$.

Let us explain the construction in more detail. $P_L$ is the set of lexical productions – analogous to rules of the form $N \rightarrow a$ in a CFG in Chomsky normal form. These rules just assign to the terminal symbols their observed distribution – this will obviously be correct in that $f_G(a) = F_L(a)$. $P$ is the interesting set of productions: these allow us to predict the features of a string $uv$ from the features of its part $u$ and $v$. To construct $P$ we take all triples of strings $u, v, uv$ that are in our finite set $K$. We observe that $u$ has the contexts $F_L(u)$ and $v$ has the set of contexts $F_L(v)$: our rule then states that we can combine any string that has all of the contexts in $F_L(u)$ together with any string that has the contexts in $F_L(v)$ and the result will have all of the contexts in $F_L(uv)$.

We will now look at a simple example. Let $L = \{a^n b^n \mid n > 0\}$, $F$, the set of features is $\{(\lambda, \lambda), (a, \lambda), (\lambda, b)\}$ and $K$, the basis, is $\{a, b, ab, aa, aab\}$. For each of the elements of $K$ we can compute the set of features that it has:

- $F_L(a)$ is just $\{(\lambda, b)\}$ – this is the only one of the three contexts in $F$ such that $f \odot a \in L$.

- $F_L(b) = \{(a, \lambda)\}$
- $F_L(aa) = \emptyset$
- $F_L(ab) = \{(\lambda, \lambda)\}$
- $F_L(aab) = \{(\lambda, b)\}$.

$G_0$ will therefore have the following lexical productions $P_L = \{(\lambda, b)\} \to a,\ \{(a, \lambda)\} \to b$. We can now consider the productions in $P$. Looking at $K$ we will see that there are only four possible triples of strings of the form $uv, u, v$: these are $(aa, a, a)$, $(ab, a, b)$, $(aab, aa, b)$ and $(aab, a, ab)$. Each of these will give rise to an element of $P$:

- The rule given by $ab = a \circ b$: $\{(\lambda, \lambda)\} \to \{(\lambda, b)\}\{(a, \lambda)\}$,
- $aa = a \circ a$ gives $\emptyset \to \{(\lambda, b)\}\{(\lambda, b)\}$,
- $aab = aa \circ b$ gives $\{(\lambda, b)\} \to \emptyset\{(a, \lambda)\}$,
- $aab = a \circ ab$ gives $\{(\lambda, b)\} \to \{(\lambda, b)\}\{(\lambda, \lambda)\}$.

Given $K, F$ and an oracle for $L$ we can thus simply write down a CBFG. However, in this case, the language $L(G_0)$ is not the same as $L$; moreover, the resulting grammar is not exact. Applying the rules for the recursive computation of $f_G$, we can see that $f_{G_0}(aab) = \{(\lambda, b)\}$ and $f_{G_0}(abb) = f_{G_0}(aabb) = \{(\lambda, b), (\lambda, \lambda)\}$ but $F_{L(G_0)}(abb) = \{(a, \lambda), (\lambda, b), (\lambda, \lambda)\}$ and thus $G_0$ is not exact. The problem here is caused by the fact that the production $\{(\lambda, b)\} \to \emptyset\{(a, \lambda)\}$ allows any string to occur in the place specified by the $\emptyset$: indeed since $\emptyset \subseteq f_{G_0}(aab)$ and $\{(a, \lambda)\} \subseteq f_{G_0}(b)$ the rule holds for $aabb$ and thus $\{(\lambda, b)\} \subseteq f_{G_0}(aabb)$. This is actually caused by the fact that there are no contexts in $F$ that correspond to the string $aa$ in $K$.

Fixing $L$ for the moment, clearly the language defined depends on two factors: the set of strings $K$ and the set of features $F$. Given $K$ and $F$, and access to a membership oracle, we can write down a CBFG with almost no computation, but we still have the problem of finding suitable $K$ and $F$ – it might be that searching for exactly the right combination of $K$ and $F$ is intractably hard. It turns out that it is also very easy to find suitable sets.

In the next section we will establish two important lemmas that show that the search for $K$ and $F$ is fundamentally tractable: first, that as $K$ increases the language defined by $G_0(K, L, F)$ will increase, and secondly that as $F$ increases the language will decrease.

Let us consider one example that illustrates these properties. Consider the language $L = \{a^n b \mid n \geq 0\} \cup \{ba^m \mid m \geq 0\} \cup \{a\}$.

First, let $K = \{a, b, ab\}$ and $F = \{(\lambda, \lambda)\}$; then, by the definition of $G_0$, we have the following productions:

- $\{(\lambda, \lambda)\} \to a$
- $\{(\lambda, \lambda)\} \to b$

- $\{(\lambda,\lambda)\} \rightarrow \{(\lambda,\lambda)\}\{(\lambda,\lambda)\}$.

It is easy to see that $L(G_0) = \Sigma^+$.

Now, suppose that $F = \{(\lambda,\lambda),(\lambda,b)\}$ with $K$ unchanged; then, by construction $G_0$ will have the following productions:

- $\{(\lambda,\lambda),(\lambda,b)\} \rightarrow a$,
- $\{(\lambda,\lambda)\} \rightarrow b$,
- $\{(\lambda,\lambda)\} \rightarrow \{(\lambda,\lambda),(\lambda,b)\}\{(\lambda,\lambda)\}$.

The language defined by $G_0$ contains $a^n b$ and also $a^n$ since $\{(\lambda,\lambda)\} \subset \{(\lambda,\lambda),(\lambda,b)\}$ allowing the third production to accept strings ending with an $a$. Thus, the language has been reduced such that $L(G_0) = \{a^n b \mid n \geq 0\} \cup \{a^m \mid m \geq 0\}$.

We continue by leaving $F = \{(\lambda,\lambda),(\lambda,b)\}$ and we enlarge $K$ such that $K = \{a, b, ab, ba\}$. The productions in $G_0$ are:

- $\{(\lambda,\lambda),(\lambda,b)\} \rightarrow a$,
- $\{(\lambda,\lambda)\} \rightarrow b$,
- $\{(\lambda,\lambda)\} \rightarrow \{(\lambda,\lambda),(\lambda,b)\}\{(\lambda,\lambda)\}$; the rule given by $ab = a \circ b$,
- $\{(\lambda,\lambda)\} \rightarrow \{(\lambda,\lambda)\}\{(\lambda,\lambda),(\lambda,b)\}$; the rule given by $ba = b \circ a$.

The addition of the last rule allows the grammar to recognize $ba^n$ and it can be easily shown that by a combination of the last two productions $a^n b a^m$ belongs to the language defined by the grammar. Then, $L(G_0)$ has been increased such that $L(G_0) = \{a^n b a^k \mid n, k \geq 0\} \cup \{a^m \mid m \geq 0\}$.

In this example, the addition of $(\lambda,b),(a,\lambda)$ and $(\lambda,a)$ to $F$ and the addition of $aab$ and $baa$ to $K$ will then define the correct language. In fact this illustrates one principle of our approach: in the infinite data limit, the construction $G_0$ will define the correct language. In the following lemma we abuse notation and use $G_0$ for when we have infinite $K$, and $F$: in this lemma we let $K$ be the set of all non-empty strings and we let $F$ be the set of all possible contexts $(\Sigma^* \times \Sigma^*)$. Recall that in this case for every string $w$ $C_L(w) = F_L(w)$.

**Lemma 12** *For any language $L$, let $G = G_0(\Sigma^+, L, \Sigma^* \times \Sigma^*)$. Then for all $w \in \Sigma^+$ $f_G(w) = C_L(w)$ and therefore $L(G) = L$.*

**Proof** By induction on the length of $w$. If $|w| = 1$, and $w = a$ then there is a lexical production $C_L(a) \rightarrow a$ and by the definition of $f_G(a) = C_L(a)$. Suppose this is true for all $w$ with $|w| \leq k$. Let $w$ be some string of length $k+1$. Consider any split of $w$ into $u, v$ such that $w = uv$. $f_G(w)$ is the union over all these splits of a function. We will show that every such split will give the same result of $C_L(w)$. By inductive hypothesis $f_G(u) = C_L(u), f_G(v) = C_L(v)$. Since $u, v, w$ are in $K = \Sigma^+$ we will also have an element of $P$ of the form $C_L(w) \rightarrow C_L(u)C_L(v)$, so we know that $f_G(w) \supseteq F_L(w)$. We now show that $f_G$ will not predict any extra contexts. Consider every production in $P$,

$F_L(u'v') \rightarrow F_L(u')F_L(v')$, that applies to $u, v$, *i.e.* with $F_L(u') \subseteq f_G(u) = C_L(u)$ and $F_L(v') \subseteq f_G(v) = C_L(v)$. Lemma 5 shows that in this case $F_L(u'v') \subseteq F_L(w)$ and thus we deduce that $f_G(w) \subseteq F_L(w)$, which establishes the lemma. ■

Informally if we take $K$ to be every string and $F$ to be every context, then we can accurately define any language. Of course, we are just interested in those cases where this can be defined finitely and we have a CBFG, in which case $L$ will be decidable, but this infinite limit is a good check that the construction is sound.

### 4.2 Monotonicity Lemmas

We now prove two lemmas that show that the size of the language, and more particularly the features predicted will increase or decrease monotonically as a function of the basis $K$, and the feature set $F$, respectively. In fact, they give also a framework for approaching a target language from $K$ and $F$.

**Lemma 13** *Suppose we have two CBFGs defined by $G = G_0(K, L, F)$ and $G' = G_0(K, L, F')$ where $F \subseteq F'$. Then for all $u$, $f_G(u) \supseteq f_{G'}(u) \cap F$.*

**Proof** Let $G'$ have a set of productions $P', P'_L$, and $G$ have a set of productions $P, P_L$. Clearly if $x \rightarrow yz \in P'$ then $x \cap F \rightarrow (y \cap F)(z \cap F)$ is in $P$ by the definition of $G_0$, and likewise for $P_L, P'_L$. By induction on $|u|$ we can show that any feature in $f_{G'}(u) \cap F$ will be in $f_G(u)$. The base case is trivial since $F'_L(a) \cap F = F_L(a)$; if it is true for all strings up to length $k$, then if $f \in f_{G'}(u) \cap F$; there must be a production in $F'$ with $f$ on the head. By the inductive hypothesis, the right hand sides of the corresponding production in $P$ will be triggered, and so $f$ must be in $f_G(u)$. ■

**Corollary 14** *Suppose we have two CBFGs defined by $G = G_0(K, L, F)$ and $G' = G_0(K, L, F')$ where $F \subseteq F'$; then $L(G) \supseteq L(G')$.*

**Proof** It is sufficient to remark that if $u \in L(G')$ then $(\lambda, \lambda) \in f_{G'}(u) \subseteq f_G(u)$ and thus $u \in L(G)$. ■

Conversely, we can show that as we increase $K$, the language and the map $f_G$ will increase. This is addressed by the next lemma.

**Lemma 15** *Suppose we have two CBFGs defined by $G = G_0(K, L, F)$ and $G' = G_0(K', L, F)$ where $K \subseteq K'$. Then for all $u$, $f_{G_0(K,L,F)}(u) \subseteq f_{G_0(K',L,F)}(u)$.*

**Proof** Clearly the sets of productions of $G_0(K, L, F)$ will be a subset of the set of productions of $G_0(K', L, F)$, and so anything that can be derived by the first can be derived by the second, again by induction on the length of the string. ■

A simple result is that when $K$ contains all of the substrings of a word, then $G_0(K, L, F)$ will generate all of the correct features for this word.

**Lemma 16** *For any string $w$, if $Sub(w) \subset K$, and let $G = G_0(K, L, F)$, then $F_L(w) \subseteq f_G(w)$.*

**Proof** By recursion on the size of $w$. Let $G = G_0(K, L, F) = \langle F, P, P_L, \Sigma \rangle$. First, notice that if $w$ is of length 1 then we have $F_L(w) \to w$ in $P_L$ and thus the lemma holds. Then suppose that $|w| = k \geq 2$. Let $u$ and $v$ in $\Sigma^+$ be such that $w = uv$. As $Sub(w) \subset K$ we have $u, v$ in $K$. Therefore the rule $F_L(w) \to F_L(u)F_L(v)$ belongs to $P$. As $|u| < |w|$ and $|v| < |w|$, by recursion we get $F_L(u) \subseteq f_G(u)$ and $F_L(v) \subseteq f_G(v)$. Thus the rule can be applied and then $F_L(w) \subseteq f_G(w)$. ■

In particular if $w \in L$, and $Sub(w) \subseteq K$, then $w \in L(G)$. This means that we can easily increase the language defined by $G$ just by adding $Sub(w)$ to $K$. In general we do not need to add every element of $Sub(w)$ – it is enough to have one binary bracketing.

To establish learnability, we need to prove that for a target language $L$, if we have a sufficiently large $F$ then $L(G_0(K, L, F))$ will be contained within $L$ and that if we have a sufficiently large $K$, then $L(G_0(K, L, F))$ will contain $L$.

### 4.3 Fiducial Feature Sets and Finite Context Property

We need to be able to prove that for any $K$ if we have enough features then the language defined will be included within the target language $L$. We formalise the idea of having enough features in the following way:

**Definition 17** *For a language $L$ and a string $u$, a set of features $F$ is fiducial on $u$ if for all $v \in \Sigma^*$, $F_L(u) \subseteq F_L(v)$ implies $C_L(u) \subseteq C_L(v)$.*

Note that if $F$ is fiducial on $u$ and $F \subset F'$ then $F'$ is fiducial on $u$. Therefore we can naturally extend this to sets of strings.

**Definition 18** *For a language $L$ and a set of strings $K$, a set of features $F$ is fiducial on $K$ if for all $u \in K$, $F$ is fiducial on $u$.*

Clearly, for any string $w$, $C_L(w)$ will be fiducial on $w$; but this is vacuous – we are interested in cases where there is a finite set of contexts which is fiducial for $w$, but where $C_L(w)$ is infinite. If $u$ and $v$ are both in $K$ then having the same features means they are syntactically congruent. However if two strings, neither of which are in $K$, have the same features this does not mean they are necessarily congruent (for instance if $F_L(v) = F_L(v') = \emptyset$). For non finite state languages, the set of congruence classes will be infinite, and thus we cannot have a finite fiducial set for the set of all strings in $Sub(L)$, but we can have a feature set that is correct for a finite subset of strings, or more generally for an infinite set of strings, if they fall into a finite number of congruence classes.

Let us consider our running example $L = \{a^n b^n | n > 0\}$. Take the string $ab$. $C_L(ab)$ is infinite and contains contexts of the form $(\lambda, \lambda), (a, b), (aa, bb)$ and so on. Consider a set with just one of these contexts, say $F = \{(a, b)\}$. This set is clearly fiducial for $ab$, since the only strings that will have this context are those that are congruent to $ab$. Consider now the string $aab$; clearly $\{(\lambda, b)\}$ is fiducial for $aab$, even though the string $a$, which is not

congruent to $aab$ also occurs in this context. Indeed, this does not violate fiduciality since $C_L(a) \supset C_L(aab)$. However, looking at string $a$, $\{(\lambda, b)\}$ is not fiducial, since $aab$ has this context but does not include all the contexts of $a$ such as, for example, $(\lambda, abb)$.

In these trivial examples, a context set of cardinality one is sufficient to be fiducial, but this is not the case in general. Consider the finite language $L = \{ab, ac, db, ec, dx, ey\}$, and the string $a$. It has two contexts $(\lambda, b)$ and $(\lambda, c)$ neither of which is fiducial for $a$ on its own. However, the set of both contexts is: $\{(\lambda, b), (\lambda, c)\}$ is fiducial for $a$.

We now define the finite context property, which is one of the two conditions that languages must satisfy to be learnable in this model; this condition is a purely language theoretic property.

**Definition 19** *A language $L$ has the Finite Context Property (FCP) if every string has a finite fiducial feature set.*

Clearly if $L$ has the FCP, then any finite set of substrings, $K$, has a finite fiducial feature set which will be the union of the finite fiducial feature sets for each element of $K$. If $u \notin Sub(L)$ then any set of features is fiducial since $C_L(u) = \emptyset$.

We note here that all regular languages have the FCP. We refer the reader to the Section 6.1.1 about CBFG and regular languages where the Lemma 35 and the associated construction proves this claim.

We can now state the most important lemma: this lemma links up the definition of the feature map in a CBFG, with the fiducial set of features to show that only correct features will be assigned to substrings by the grammar. It states that the features assigned by the grammar will correspond to the language theoretic interpretation of them as contexts.

**Lemma 20** *For any language $L$, given a set of strings $K$ and a set of features $F$, let $G = G_0(K, L, F)$. If $F$ is fiducial on $K$, then for all $w \in \Sigma^*$ $f_G(w) \subseteq F_L(w)$.*

**Proof** We proceed by induction on length of the string. *Base case:* strings of length 1. $f_G(w)$ will be the set of observed contexts of $w$, and since we have observed these contexts, they must be in the language. *Inductive step:* let $w$ a string of length $k$.

Take a feature $f$ on $f_G(w)$; by definition this must come from some production $x \to yz$ and a split $u, v$ of $w$. The production must be from some elements of $K$, $u', v'$ and $u'v'$ such that $y = F_L(u'), z = F_L(v')$ and $x = F_L(u'v')$. If the production applies this means that $F_L(u') = y \subseteq f_G(u) \subseteq F_L(u)$ (by inductive hypothesis), and similarly $F_L(v') \subseteq F_L(v)$. By fiduciality of $F$ this means that $C(u') \subseteq C(u)$ and $C(v') \subseteq C(v)$. So by Lemma 5 $C(u'v') \subseteq C(uv)$. Since $f \in C(u'v')$ then $f \in C(uv) = C(w)$. Therefore, since $f \in F$ and $C(w) \cap F = F_L(w)$, $f \in F_L(w)$, and therefore $f_G(w) \subseteq F_L(w)$.

■

**Corollary 21** *If $F$ is fiducial on $K$ then $L(G_0(K, F, L)) \subseteq L$.*

Therefore for any finite set $K$ from an FCP language, we can find a set of features so that the language defined by those features on $K$ is not too big.

### 4.4 Kernel and Finite Kernel Property

We will now show a complementary result, namely that for a sufficiently large $K$ the language defined by $G_0$ will include the target language. We will start by formalising the idea that a set $K$ is large enough, by defining the idea of a *kernel*.

**Definition 22** *A finite set $K \subseteq \Sigma^*$ is a kernel for a language $L$, if for any set of features $F$, $L(G_0(K, F, L)) \supseteq L$.*

Consider again the language $L = \{a^n b^n | n \geq 0\}$. The set $K = \{a, b, ab\}$ is not a kernel, since if we have a large enough set of features, then the language defined will only be $\{ab\}$ which is a proper subset of $L$. However $K = \{a, b, ab, aab, abb, aabb\}$ is a kernel: no matter how large a set of features we have the language defined will always include $L$. Consider a language $L' = L \cup \{b^{16}\}$. In this case, a kernel for $L'$ must include, as well as a kernel for $L$, some set of substrings of $b^{16}$: it is enough to have $b^{16}, b^8, b^4, bb, b$.

To prove that a set is a kernel, it suffices to show that if we consider all the possible features for building the grammar, we will contain the target language; any smaller set of features defines then a larger language. In our case, we can take the infinite set of all contexts and define productions based on the congruence classes. If $F$ is the set of all contexts then we have $F_L(u) = C_L(u)$, thus the productions will be exactly of the form $C(uv) \to C(u)C(v)$. This is a slight abuse of notation since feature sets are normally finite.

**Lemma 23** *Let $F = \Sigma^* \times \Sigma^*$; if $L(G_0(K, L, F)) \supseteq L$ then $K$ is a kernel.*

**Proof** By monotonicity of $F$: any finite feature set will be a subset of $F$. ■

Not all context-free languages will have a finite kernel. For example $L = \{a^+\} \cup \{a^n b^m | n < m\}$ is clearly context-free, but does not have a finite kernel. Assume that the set $K$ contains all strings of length less than or equal to $k$. Assume w.l.o.g. that the fiducial set of features for $K$ includes all features $(\lambda, b^i)$, where $i \leq k + 1$. Consider the rules of the form $F_L(a^k) \to F_L(a^j)F_L(a^{k-j})$; we can see that no matter how large $k$ is, the derived CBFG will undergenerate as $a^k$ is not congruent to $a^{k-1}$.

**Definition 24** *A context-free grammar $G_T = \langle V, S, P, \Sigma \rangle$ has the Finite Kernel Property (FKP) iff for every non-terminal $N \in V$ there is a finite set of strings $K(N)$ such that $a \in K(N)$ if $a \in \Sigma$ and $N \to a \in P$ and such that for all $k \in K(N), N \overset{*}{\Rightarrow} k$ and where for every string $w \in \Sigma^*$ such that $N \overset{*}{\Rightarrow} w$ there is a string $k \in K(N)$ such that $C(k) \subseteq C(w)$. A CFL $L$ has the FKP, if there is a grammar in CNF for it with the FKP.*

Notice that all regular languages have the FKP since they have a finite number of congruence classes.

**Lemma 25** *Any context-free language with the FKP has a finite kernel.*

**Proof** Let $G_T = \langle V, S, P, \Sigma \rangle$ be such a CNF CFG with the FKP. Define

$$K(G_T) = \bigcup_{N \in V} \left( K(N) \cup \bigcup_{X \to MN \in P} K(M)K(N) \right). \tag{4}$$

We claim that $K(G_T)$ is a kernel. Assume that $F = \Sigma^* \times \Sigma^*$ and let $G$ be such that $G = G_0(K(G_T), L(G_T), F) = \langle F, (\lambda, \lambda), P, P_L, \Sigma \rangle$.

We will show, by induction on the length of derivation of $w$ in $G_T$, that for all $N, w$ if $N \overset{*}{\Rightarrow} w$ then there is a $k$ in $K(N)$ such that $f_G(w) \supseteq C(k)$. If length of derivation is 1, then this is true since $|w| = 1$ and thus $w \in K(N)$: therefore $C(w) \to w \in P_L$. Suppose it is true for all derivations of length less than $j$. Take a derivation of length $j$; say $N \overset{*}{\Rightarrow} w$. There must be a production in $G_T$ of the form $N \to PQ$, where $P \Rightarrow^* u$ and $Q \Rightarrow^* v$, and $w = uv$. By inductive hypothesis; we have $f_G(u) \supseteq C(k_u)$ and $f_G(v) \supseteq C(k_v)$. By construction $k_u k_v \in K(G_T)$ and then there will be a rule $C(k_u k_v) \to C(k_u)C(k_v)$ in $P$. Therefore $f_G(uv) \supseteq C(k_u k_v)$. Since $N \overset{*}{\Rightarrow} k_u k_v$ there must be some $k_{uv} \in K(N)$ such that $C(k_{uv}) \subseteq C(k_u k_v)$. Therefore $f_G(w) \supseteq C(k_u k_v) \supseteq C(k_{uv})$.

Now we can see that if $w \in L$, then $S \overset{*}{\Rightarrow} w$, then there is a $k \in K(S)$ such that $f_G(w) \supseteq C(k)$ and $S \overset{*}{\Rightarrow} k$, therefore $(\lambda, \lambda) \in f_G(w)$ since $(\lambda, \lambda) \in C(k)$, thus $w \in L(G)$ and therefore $K$ is a kernel. ■

### 4.5 Learning Algorithm

Before we present the algorithm, we will discuss the learning model that we use. The class of languages that we will learn is suprafinite and thus we cannot get a positive data only identification in the limit result (Gold, 1967). Ultimately we are interested in a more realistic probabilistic learning paradigm, but for mathematical convenience it is appropriate to establish the basic results in a symbolic paradigm. The ultimate goal is to model natural languages, where negative data, or equivalence queries are generally not available or are computationally impossible. Accordingly, we have decided to use the model of positive data together with membership queries: an oracle can tell the learner whether a string is in the language or not (Angluin, 1988). The presented algorithm runs in time polynomial in the size of the sample $S$: since the strings are of variable length, this size must be the sum of the lengths of the strings in $S$, $\sum_{w \in S} |w|$. We should note that this is not a strong enough result: Pitt (1989) showed that any algorithm can be made polynomial, by only processing a small prefix of the data. It is hard to tighten the model sufficiently: the suggestion of de la Higuera (1997) for a polynomial characteristic set is inapplicable for representations, such as the ones in this paper, that are powerful enough to define languages whose shortest strings are exponentially long. Accordingly we do not require in this model a polynomial dependence on the size of the representation. We note that the situation is unsatisfactory, but we do not intend to propose a solution in this paper. We merely point out that the algorithm is genuinely polynomial and processes all of the data in the sample without "delaying tricks" of the type discussed by Pitt.

**Definition 26** *A class of languages $\mathbb{L}$ is identifiable in the limit (IIL) from positive data and a membership oracle with polynomial time and queries iff there exist two polynomials $p(), q()$ and an algorithm $A$ such that:*

- *Given an infinite presentation of positive examples $S$, where $S_n$ is the first $n$ examples of the presentation, and $S_n$ has total size $m$,*

1. *A returns a representation $G = A(S_n)$ in time $p(m)$.*
2. *A asks at most $q(m)$ queries to build $A(S_n)$.*

- *For each language $L \in \mathbb{L}$, for each presentation $S$ of $L$, there exists an index $n$ such that for all $N \geq n$: $A(S_N) = A(S_n)$ and $L(A(S_n)) = L$.*

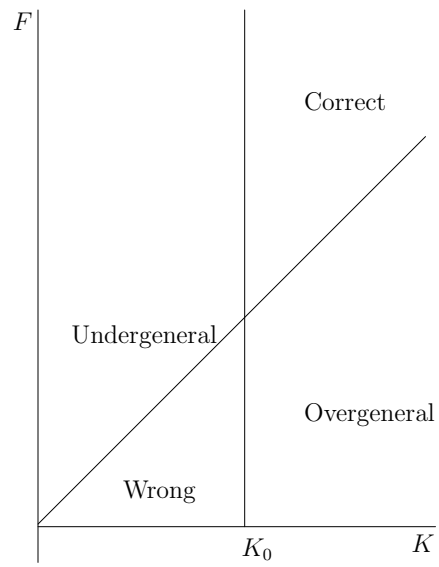


Figure 2: The relationship between $K$ and $F$: The diagonal line is the line of fiduciality: above this line means that $F$ is fiducial on $K$. $K_0$ is a kernel for the language.

Before we present the algorithm we hope that it is intuitively obvious how the approach will work. Figure 2 diagrammatically shows the relationship between $K$ and $F$. When we have a large enough $K$, we will be to the right of the vertical line; when we have enough features for that $K$ we will be above the diagonal line. Thus the basis of the algorithm is to move to the right, until we have enough data, and then to move up vertically, increasing the feature set until we have a fiducial set.

We can now define our learning algorithm in Algorithm 1. Informally, $D$ is the list of all strings that have been seen so far and $G_n$ is the current grammar obtained with the first $n$ strings of $D$. The algorithm uses two tests: one test is just to determine if the current hypothesis undergeneralises. This is trivial, since we have a positive presentation of the data, and so eventually we will be presented with a string in $L \setminus L(G_n)$. In this case we need to increase $K$; we accordingly increase $K$ to the set of all substrings that we have observed so far. The second test is a bit more delicate. We want to detect if our algorithm overgeneralises. This requires us to search through a polynomially bounded set of strings looking for a string that is in $L(G_n) \setminus L$. An obvious candidate set is $Con(D) \odot Sub(D)$; but though we conjecture that this is adequate, we have not yet been able to prove that is correct, as it might be that the overgenerated string does not lie in $Con(L) \odot Sub(L)$.

Here we use a slightly stricter criterion: we try to detect whether $F$ is fiducial for $K$: we search through a polynomially bounded set of strings, $Sub(D)$, to find a violation of the fiduciality condition. If we find such a violation, then we know that $F$ is not fiducial for $K$, and so we increase $F$ to the set of all contexts that we have seen so far, $Con(D)$.

In Algorithm 1, $G_0(K, \mathcal{O}, F)$ denotes the same construction as $G_0(K, L, F)$, except that we use membership queries with the oracle $\mathcal{O}$ to compute $F_L$ for each element in $K$. We give the identification in the limit version of the algorithm, *i.e.* that admits an infinite positive presentation of strings in input.

**Algorithm 1**: CBFG learning algorithm IIL

**Data**: A sequence of strings $S = \{w_1, w_2 \ldots, \}$, membership oracle $\mathcal{O}$
**Result**: A sequence of CBFGs $G_1, G_2, \ldots$
$K \leftarrow \emptyset$ ; $D \leftarrow \emptyset$ ; $F \leftarrow \{(\lambda, \lambda)\}$ ; $G = G_0(K, \mathcal{O}, F)$ ;
**for** $w_i$ **do**
  $D \leftarrow D \cup \{w_i\}$; $C \leftarrow Con(D)$; $S \leftarrow Sub(D)$;
  **if** $\exists w \in D \setminus L(G)$ **then**
    $K \leftarrow S$ ; $F \leftarrow C$ ;
  **end**
  **else if** $\exists v \in S, u \in K, f \in C$ *such that* $F_L(u) \subseteq F_L(v)$ *and* $f \odot u \in L$ *but* $f \odot v \notin L$ **then**
    $F \leftarrow C$ ;
  **end**
  $G = G_0(K, \mathcal{O}, F)$ ;
  Output $G_i = G$ ;
**end**

**Theorem 27** *Algorithm 1 runs in polynomial time in the size of the sample, and makes a polynomial number of calls to the membership oracle.*

**Proof** The value of $D$ will just be the set of observed strings; $Sub(D)$ and $Con(D)$ are both polynomially bounded by the size of the sample, and therefore so are $|K|$ and $|F|$. Therefore the number of calls to the oracle is clearly polynomial, as it is bouned by $|K||F|$. Computing $G_0$ is also polynomial, since $|P| \leq |K|^2$, and all strings involved are in $Sub(D)$. ■

### 4.6 Identification in the limit result

In the following, we consider the class of context-free languages having the FCP and the FKP, represented by CBFG. $K_n$ denotes the value of $K$ at the $n^{th}$ loop, and similarly for $F$, $D$.

**Definition 28** $\mathcal{L}_{CFG}$ *is the class of all context-free languages that satisfy the FCP and the FKP.*

In what follows we assume that $L$ is an element of this class, and that $w_1, \ldots, w_n, \ldots$ is a infinite presentation of the language. The proof is straightforward and merely requires an

analysis of a few cases. We will proceed as follows: there are 4 states that the model can be in, that correspond to the four regions of the diagram in Figure 2.

1. $K$ is a kernel and $F$ is fiducial for $K$; in this case the model has converged to the correct answer. This is the region labeled *correct* in Figure 2.

2. $K$ is a kernel and $F$ is not fiducial for $K$: then $L \subseteq L(G)$, and at some later point, we will increase $F$ to a fiducial set, and we will be in state 1: this is the region labeled *overgeneral*.

3. $K$ is not a kernel and $F$ is fiducial. Either $L(G) = L$, in which case we have converged to a correct answer or, if not, we will define a proper subset of the target language. In the later case we will change hypothesis at some later point, increase $K$ to a kernel, and move to state 2 or state 1. This is the area labeled *undergeneral*.

4. $K$ is not a kernel and $F$ is not fiducial: in this case at some point we will move to states 1 or 2. This is the area labeled *wrong*.

We will start by making some basic statements about properties of the algorithm:

**Lemma 29** *If there is some $n$, such that $F_n$ is fiducial for $K_n$ and $L(G_n) = L$, then the algorithm will not change its hypothesis: i.e. for all $n > N$, $K_n = K_N, F_n = F_N$ and therefore $G_n = G_N$.*

**Proof** If $L(G_n)$ is correct, then the first condition of the loop will never be met; if $F_n$ is fiducial for $K_n$, then the second condition will never be satisfied. ■

**Lemma 30** *If there is some $N$ such that $K_N$ is a kernel, then for all $n > N$, $K_n = K_N$.*

**Proof** Immediate by definition of a kernel, and of the algorithm. ■

We now prove that if $F$ is not fiducial then the algorithm will be able to detect this.

**Lemma 31** *If there is some $n$ such that $F_n$ is not fiducial for $K_n$, then there is some index $n' \geq n$ at which $F_n$ will be increased.*

**Proof** If $F_n$ is not fiducial, then by definition there is some $u \in K$, $v \in \Sigma^+$ such that $F_L(u) \subseteq F_L(v)$, but there is an $f \in C_L(u)$ that is not in $C_L(v)$. By construction $F_L(u)$ is always non-empty, and so is $F_L(v)$. Thus $v \in Sub(L)$. Note $f \odot u \in L$, so $f \in Con(L)$. Let $n'$ be the smallest index such that $v \in Sub(D_n)$ and $f \in Con(D_n)$: at this point, either $F_n$ will have changed, or not, in which case it will be increased at this point. ■

We now prove that we will always get a fiducial feature set.

**Lemma 32** *For any $n$, there is some $n'$ such that $F_{n'}$ is fiducial for $K_n$.*

**Proof** If $F_n$ is fiducial then $n' = n$ satisfies the condition. Assume otherwise. Let $F$ be a finite set of contexts that is fiducial for $K_n$. We can assume that $F \subseteq Con(L)$. Let $n_1$ be the first index such that $Con(D_{n_1})$ contains $F$. At this point we are not sure that $F_{n_1} = Con(D_{n_1})$ since the conditions for changing the set of contexts may not be reached. Anyhow, if it is the case then $F_{n_1}$ is fiducial, then $n_1 = n'$ satisfies the condition. If not, then by the preceding lemma, there must be some point $n_2$ at which we will increase the set of contexts of the current grammar; $F_{n_2} = Con(n_2)$ must contain $F$ since $Con(D_{n_1}) \subset Con(D_{n_2})$, and is therefore fiducial, and so $n_2 = n'$ satisfies the condition. ■

**Lemma 33** *For every positive presentation of an $L \in \mathcal{L}_{CFG}$, there is some $n$ such that either the algorithm has converged to a correct grammar or $K_n$ is a kernel.*

**Proof** Let $m$ be the smallest number such that $Sub(D_m)$ is a kernel. Recall that any superset of a kernel is a kernel, and that all CFL with the FKP have a finite kernel (Lemma 25), and that such a kernel is a subset of $Sub(L)$, so such an $m$ must exist.

Consider the grammar $G_m$; there are three possibilities:
1. $L(G_m) = L$, and $F_m$ is fiducial, in which case the grammar has converged.
2. $L(G_m)$ is a proper subset of $L$ and $F_m$ is fiducial. Let $m'$ be the first point at which $w_{m'}$ is in $L \setminus L(G_m)$; at this point $K_{m'}$ will be increased to include $Sub(D_m)$ and it will therefore be a kernel.
3. $F_m$ is not fiducial: in this case by Lemma 32; there is some $n$ at which $F_n$ is fiducial for $K_m$. Either $K_n = K_m$ in which case this reduces to Case 2; or $K_n$ is larger than $K_m$ in which case it must be a kernel, since it will include $Sub(D_m)$ which is a kernel.

■

We now can prove the main result of the paper:

**Theorem 34** *Algorithm 1 identifies in the limit the class of context-free languages with the finite context property and the finite kernel property.*

**Proof** By Lemma 33 there is some point at which it converges or has a kernel. If $K_n$ is a kernel then by Lemma 32, there is some point $n'$ at which we have a fiducial feature set. Therefore $L(G_{n'}) = L$, and the algorithm has converged. ■

### 4.7 Examples

We will now give a worked example of the algorithm.

Suppose $L = \{a^n b^n | n > 0\}$.

$G_0$ will be the empty grammar, with $K = \emptyset, F = \{(\lambda, \lambda)\}$ and an empty set of productions. $L(G_0) = \emptyset$.

1. Suppose $w_1 = ab$. $D = \{ab\}$. This is not in $L(G_0)$ so we set
   - $K = Sub(D) = \{a, b, ab\}$.

- $F = Con(D) = \{(\lambda, \lambda), (a, \lambda), (\lambda, b)\}$.

This gives us one production: $F_L(ab) \to F_L(a)F_L(b)$ which corresponds to $\{(\lambda, \lambda)\} \to \{(\lambda, b)\}\{(a, \lambda)\}$, and the lexical productions $\{(\lambda, b)\} \to a, \{(a, \lambda)\} \to b$. The language defined is thus $L(G_1) = \{ab\}$.

2. Suppose $w_2 = aabb$. $D = \{ab, aabb\}$. This is not in $L(G_1)$ so we set

   - $K = Sub(D) = \{a, b, ab, aa, bb, aab, abb, aabb\}$.
   - $F = Con(D) = \{(\lambda, \lambda), (a, \lambda), (\lambda, b), (aa, \lambda), (a, b), (\lambda, bb), (aab, \lambda), (aa, b), (a, bb), (\lambda, abb)\}$.

   We then have the following productions:

   - $F_L(ab) \to F_L(a), F_L(b)$ which is $\{(\lambda, \lambda), (a, b)\} \to \{(a, bb), (\lambda, abb), (\lambda, b)\}, \{(aa, b), (aab, \lambda), (a, \lambda)\}$
   - $F_L(aab) \to F_L(a), F_L(ab)$ which is $\{(a, bb), (\lambda, b)\} \to \{(a, bb), (\lambda, abb), (\lambda, b)\}, \{(\lambda, \lambda), (a, b)\}$
   - $F_L(aab) \to F_L(aa), F_L(b)$ which is $\{(a, bb), (\lambda, b)\} \to \{(\lambda, bb)\}, \{(aa, b), (aab, \lambda), (a, \lambda)\}$
   - $F_L(bb) \to F_L(b), F_L(b)$ which is $\{(aa, \lambda)\} \to \{(aa, b), (aab, \lambda), (a, \lambda)\}, \{(aa, b), (aab, \lambda), (a, \lambda)\}$
   - $F_L(aa) \to F_L(a), F_L(a)$ which is $\{(\lambda, bb)\} \to \{(a, bb), (\lambda, abb), (\lambda, b)\}, \{(a, bb), (\lambda, abb), (\lambda, b)\}$
   - $F_L(aabb) \to F_L(a), F_L(abb)$ which is $\{(\lambda, \lambda), (a, b)\} \to \{(a, bb), (\lambda, abb), (\lambda, b)\}, \{(aa, b), (a, \lambda)\}$
   - $F_L(aabb) \to F_L(aa), F_L(bb)$ which is $\{(\lambda, \lambda), (a, b)\} \to \{(\lambda, bb)\}, \{(aa, \lambda)\}$
   - $F_L(aabb) \to F_L(aab), F_L(b)$ which is $\{(\lambda, \lambda), (a, b)\} \to \{(a, bb), (\lambda, b)\}, \{(aa, b), (aab, \lambda), (a, \lambda)\}$
   - $F_L(abb) \to F_L(a), F_L(bb)$ which is $\{(aa, b), (a, \lambda)\} \to \{(a, bb), (\lambda, abb), (\lambda, b)\}, \{(aa, \lambda)\}$
   - $F_L(abb) \to F_L(ab), F_L(b)$ which is $\{(aa, b), (a, \lambda)\} \to \{(\lambda, \lambda), (a, b)\}, \{(aa, b), (aab, \lambda), (a, \lambda)\}$

   and the two lexical productions:

   - $F_L(a) \to a$ which is $\{(a, bb), (\lambda, abb), (\lambda, b)\} \to a$
   - $F_L(b) \to b$ which is $\{(aa, b), (aab, \lambda), (a, \lambda)\} \to b$.

   $K$ is now a kernel and $L(G) = L$, but $F$ is not fiducial for $K$, since $(\lambda, bb)$ is not fiducial for $aa$ (consider $aaab$).

3. Suppose $w_3 = aaabbb$. Now $|Con(D_3)| = 21$; there are now several elements of $Con(D_3)$ that are similar. For example $(\lambda, \lambda), (a, b)$ and $(aa, bb)$ are identical but as it is harmless for the resulting grammar, it does not mind. Now we will detect that $F$ is not fiducial: we will find $v = aaab$, $u = aa$ and $f = (\lambda, abbb)$; $F_L(aa) = \{(\lambda, bb)\} = F_L(aaab)$, but $f \odot aaab = aaababbb$ which is not in $L$. We will therefore increase $F$ to be $Con(D_3)$, and then the algorithm will have converged. The final grammar will have 10 productions and 2 lexical productions; $|K| = 8$ and $|F| = 21$.

## 5. Practical Behavior of the Algorithm

In this section, we propose to study the behavior of our algorithm from a practical point of view. We focus more specifically on two important issues. The first one deals with the learning ability of the algorithm when the conditions for the theoretical learning result are not reached. Indeed, although the identification in the limit paradigm proves that with sufficient data it is possible to obtain exact convergence, it says nothing about the convergence when fewer learning examples are available: does the output get closer and closer to the target until it reaches it or does it stay far from the expected solution until receives enough data? The second question concerns the learning behavior of the algorithm: does it tend to over-generalise or to under-generalise?

For our experimental setup, we need to select appropriate datasets. In grammatical inference little has been done concerning benchmarking. The main available corpora are those of the on line competitions organised by the International Colloquium on Grammatical Inference. Three different competitions have recently taken place: the *Abbadingo One* (Lang et al., 1998) which was about regular languages, the *Omphalos* competition on context-free languages (Starkie et al., 2004) and the *Tenjino* competition (Starkie et al., 2006) dealing with transducers learning. Note that some of these datasets correspond to extremely hard learning problems since their main objective was to push the state of the art (some problems of the *Abbadingo One* competition are still unsolved more than ten years after its official end!)

However, these datasets can not be directly used for evaluating our algorithm because the solutions or the target models are not available. Our algorithm needs an oracle and thus we need a way to give answers to membership queries. In order to overcome this drawback, we chose to build synthetically some data sets following the experimental setup proposed by these competitions. More precisely, we decided to randomly generate target context-free grammars following what has been done for the *Omphalos* competition. Each grammar is then used either to generate training and test sets or as an oracle for answering membership queries.

In the following paragraphs we describe first the generation of the target context-free grammars, then the experimental setup with learning and test datasets used and finally the results and conclusions that can be drawn.

### 5.1 Generation of Target Context-free Grammars

To generate the target grammars we follow the process used for the *Omphalos* competition (Starkie et al., 2004). We built 50 different grammars randomly according the following principles. For each grammar, we first fix the number of non-terminals and terminals which are randomly chosen between 4 and 7 for the non-terminals (including the start symbol) and between 2 and 4 for terminal symbols. Then we randomly generate 20 context-free rules in Chomsky normal form such that every non-terminal appears at least once in the left hand side of a grammar rule. In order to avoid the presence of useless rules, we apply two simple procedures: if a non-terminal can not generate any terminal string, a new terminal rule generating one terminal symbol is created for this non-terminal; if a non-terminal can not be reached from the start symbol, we erase it from the grammar (*i.e.* we remove all the rules containing this non-terminal). From these grammars without useless rules, we force them to generate non finite languages by checking that the start symbol is used at least once in a right hand side of a grammar rule (in average this symbol appears in a right hand side of a rule more than 3 times per grammar).

The main difference with the *Omphalos* generation process is that we do not especially need non-regular languages. Indeed, one of the aim of these experiments is to give an idea on the behavior of the algorithm when its theoretical assumptions are not likely to be valid. From this standpoint, all randomly generated non-finite languages are good candidates as learning targets. However, with a similar principle used for the *Omphalos* competition, we checked that some of the generated grammars can not be easily solved by methods for regular languages. Although we can not decide if these grammars define non regular context-free languages, it ensures us that the target models are at least not too simple.

### 5.2 Experimental Setup

For each target grammar we generate a learning and a test sample following the *Omphalos* competition requirements. We build the learning sample by first creating a *structurally complete set* of strings for each grammar. This set is built such that for each rule of the target grammar, at least one string of the set can be derived using this rule (Parekh and Honavar, 1996). This would guarantee that the complete learning set would have the minimal amount of information for finding the structure of the grammar. We then complete this learning set by randomly generating new strings from the grammar in order to have a total of 50 examples. We chose arbitrarily this value for two reasons: first it is sufficient to ensure that each sample contains strictly a structurally complete set for each target grammar and secondly we are likely to be far from the guarantees of the identification in the limit framework.

The construction of the test set needs particular attention. Since the learning phase uses a membership oracle, when the hypothesis is being constructed, some new strings may be built and queried for the oracle by picking a prefix, a substring and a suffix from the learning sample. Thus, even if the test set does not contain any string of the learning sample, the construction $G_0$ may consider some strings present in the test set. In order to avoid this drawback, *i.e.* to guarantee that no string of the test could be seen during the construction of the CBFG, each test string has a length of at least 3 times the maximal length of the strings in the learning set. According to this procedure, we randomly generate a test set

of 1000 strings over the alphabet of terminal symbols used to define the target grammar (1000 examples is twice the size of the small test sets of the *Omphalos* competition). The test sequences are then labeled positive or negative depending on their membership to the language defined by the grammar. We repeat this process until we have the desired number of strings. The ratio between strings in the language and strings outside the language is fixed to be between 40% and 60%.

In order to study the behavior of our algorithm, we define the following setup. For each target context-free grammar, we construct a CBFG by applying the construction $G_0(K, \mathcal{O}, F)$ with $K = Sub(S)$ and $F = Con(S)$ where $S$ is a set of strings drawn from the learning set and using the target grammar as the oracle $\mathcal{O}$ for the membership queries. We generate different sets $S$ by drawing an increasing number of learning examples (from 2 to 50) from the learning sample of the considered grammar. Then, we evaluate the learned CBFG on the test sample by measuring the accuracy of correct classification. We present the results averaged on the 50 test sets of the different target context-free grammars.

### 5.3 Results and Discussion

Figure 3 shows the averaged accuracy over the different target grammars according to the number of strings in the learning sample. We can note that a high correct classification rate (nearly 90%) is reached with 20 examples and with only 5 examples an accuracy of 75% is obtained. These results indicate that a relevant hypothesis can be found even with few examples. The standard deviations represented by vertical bars show a good stability of the results from learning sets of 20 strings. This confirms that our algorithm is able to learn partly correct representations even when learning sets may not have a kernel or a fiducial learning set and thus are far from the identification in the limit assumptions.

The analysis of the behavior of the algorithm in terms of false positive and false negative rates is shown in Table 5.3. The proportion of false negatives (*i.e.* positive strings classified as negative) is higher than the proportion of false positives (*i.e.* negative strings classified as positive), whatever the size of the learning sample is. Thus the output of the algorithm tends more to under-generalise than the converse. As it is generally admitted that over-generalisation is the main trouble when learning from positive examples, this tendency confirms that the algorithm behaves well. However, it is difficult to draw firm conclusions without a natural distribution over negative examples.

The preceding results show that despite its simplicity the algorithm behaved nicely during these experiments, in particular concerning over-generalisation. We focus now on the amount of queries needed by the algorithm for building the CBFG. The growth of the number of requested queries according to the average size of the learning sample is shown in Figure 4 (recall that here the size of the sample means the sum of the string lengths of the sample). While a very worst case analysis of the grammar construction used by $G_0$ would lead to a complexity in $O(|S|^5)$, we can observe that the number of queries seems to be quadratic, at least in the case of the grammars we consider here. However, the volume of queries used is large, which can be explained by the simplicity of the algorithm. From a practical standpoint, it is clear that much work has to be done in order to try to minimise the number of queries needed by selecting the most informative examples, but this point is

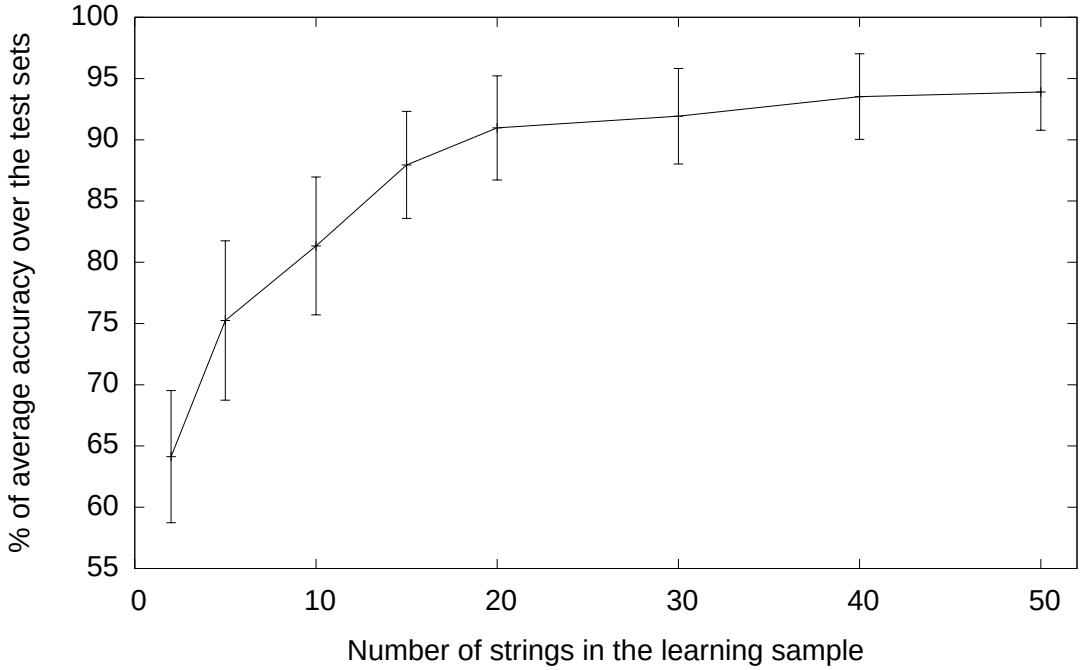


Figure 3: Evolution of the average percentage of correct classification according to the number of learning examples.

| Number of strings in $S$ | false positive | false negative |
|---|---|---|
| 02 | 07.2 % ± 12.3 | 36.4 % ± 7.0 |
| 05 | 04.5 % ± 4.7 | 28.4 % ± 9.1 |
| 10 | 03.8 % ± 2.9 | 22.4 % ± 8.4 |
| 15 | 03.9 % ± 2.6 | 14.1 % ± 6.6 |
| 20 | 04.1 % ± 3.0 | 09.4 % ± 6.1 |
| 30 | 04.2 % ± 2.8 | 07.9 % ± 5.5 |
| 40 | 03.9 % ± 2.6 | 05.6 % ± 4.7 |
| 50 | 04.4 % ± 1.6 | 04.8 % ± 3.9 |

Table 1: Average percentage of false positive and false negative rates obtained over the test samples.

out of the scope of the paper.

Finally, we can note that these experiments suffer of the lack of comparison with other approaches. This is due to the fact that, as far as we know, no other algorithm uses a positive learning sample and a membership oracle only. Indeed, since the work of Angluin about the Minimum Adequate Teacher (Angluin, 1988) all algorithms using membership queries are designed with the additional help of equivalence queries. The point of view adopted in this

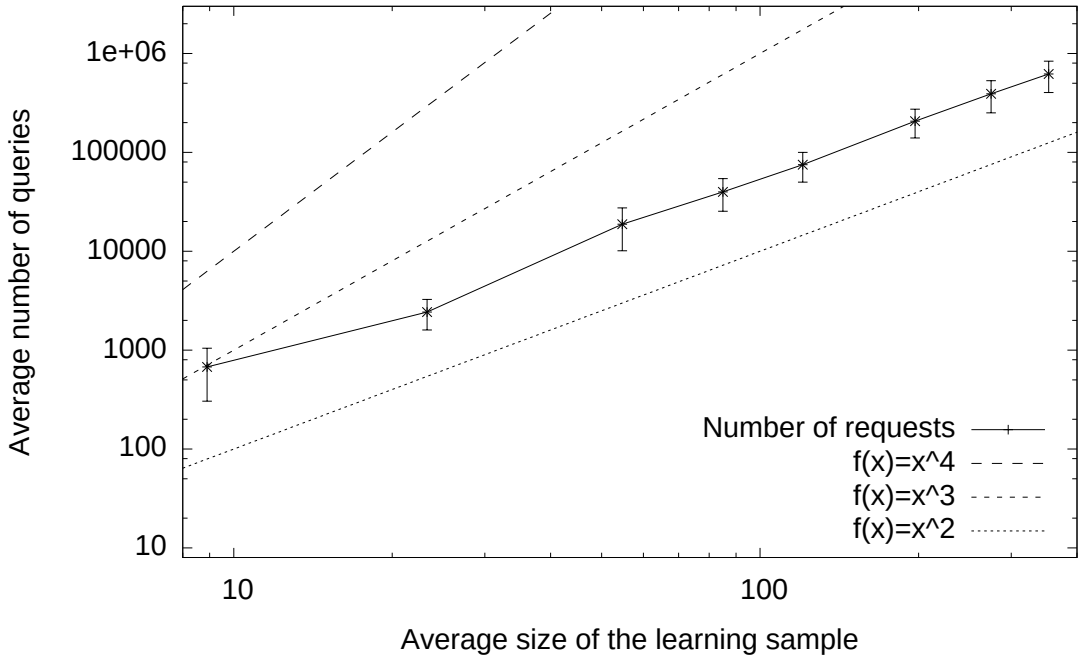


Figure 4: Growth of the number of membership queries versus the average of the total size of the learning sample (using log log scale).

paper is rather theoretical since our aim was to show the relevance of CBFG representations for language learning. However, a perspective of our work is to try to avoid the use of the oracle (by using statistical or simulation methods) which will allow us to compare more easily our approach with other methods.

## 6. Expressiveness of CBFG

In this section, we compare the expressiveness of CBFG with other well known representations. As noted earlier, we are primarily interested in the class of exact CBFGs – these are CBFGs where the presence of a contextual feature in the representation corresponds exactly to the language theoretic interpretation of the context. The class of unrestricted CBFG is significantly larger, but less relevant.

The algorithm presented in this paper cannot learn the entire class of exact CBFGs, but we conjecture that there are more powerful algorithms that can; see Clark (2009) for some steps in this direction.

### 6.1 Exact CBFGs and the Chomsky Hierarchy

We start by examining the class of languages defined by exact CBFGs. We will show that this class

- includes all regular languages

- does not include all context free languages
- includes some non-context-free languages.

This class is thus orthogonal to the Chomsky hierarchy.

#### 6.1.1 Regular Languages

Any regular language can be defined by an exact CBFG. We will show a way of constructing an exact CBFG for any regular language. Suppose we have a regular language $L$: we consider the left and right residual languages:

$$u^{-1}L = \{w|uw \in L\}, \tag{5}$$

$$Lu^{-1} = \{w|wu \in L\}. \tag{6}$$

For any $u \in \Sigma^*$, let $l_{min}(u)$ be the lexicographically shortest element such that $l_{min}^{-1}L = u^{-1}L$. The number of such $l_{min}$ is finite by the Myhill-Nerode theorem, we denote by $L_{min}$ this set, i.e. $\{l_{min}(u)|u \in \Sigma^*\}$. We define symmetrically $R_{min}$ for the right residuals ($Lr_{min}^{-1} = Lu^{-1}$).

We define the set of contexts as:

$$F(L) = L_{min} \times R_{min}. \tag{7}$$

$F(L)$ is clearly finite by construction.

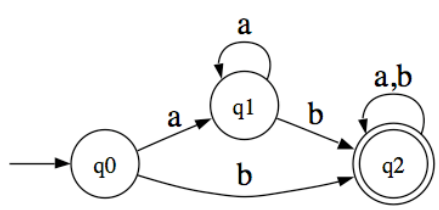


Figure 5: Example of a DFA. The left residuals are defined by $\lambda^{-1}L$, $a^{-1}L$, $b^{-1}L$ are the right ones by $L\lambda^{-1}$, $Lb^{-1}$, $Lab^{-1}$ (note here that $La^{-1} = L\lambda^{-1}$).

If we consider the regular language defined by the deterministic finite automata of Figure 5, we obtain $L_{min} = \{\lambda, a, b\}$ and $R_{min} = \{\lambda, b, ab\}$ and thus
$F(L) = \{(\lambda, \lambda), (a, \lambda), (b, \lambda), (\lambda, b), (a, b), (b, b), (\lambda, ab), (a, ab), (b, ab)\}$.

By considering this set of features, we can prove the following lemma:

**Lemma 35** *For any strings $u, v$ such that $F_L(u) \supset F_L(v)$ then $C_L(u) \supset C_L(v)$.*

**Proof** Suppose $F_L(u) \supset F_L(v)$ and let $(l, r)$ be a context in $C_L(v)$. Let $l'$ be the lexicographically shortest element of $\{u : u^{-1}L = l^{-1}L\}$ and $r'$ the lexicographically shortest element of $\{u : Lu^{-1} = Lr^{-1}\}$. By construction we have $(l', r') \in F(L)$ and $l'vr' \in L$, as $vr' \in l'^{-1}L = l^{-1}L$. $F_L(v)$ is contained in $F_L(u)$ therefore we have $l'ur' \in L$. $l'^{-1}L = l^{-1}L$ implies $lur' \in L$. As $r'$ is congruent to $r$, $lur \in L$. ■

This lemma means that the set of features $F$ is sufficient to represent context inclusion.

Note that the number of congruence classes of a regular language is finite. Each congruence class is represented by a set of contexts $F_L(u)$. Let $K_L$ be finite set of strings formed by taking the lexicographically shortest string from each congruence class. The final grammar can be obtained by combining elements of $K_L$. For every pair of strings $u, v \in K_L$, we define a rule

$$F_L(uv) \rightarrow F_L(u)F_L(v) \tag{8}$$

and we add lexical productions of the form $F_L(a) \rightarrow a$, $a \in \Sigma$.

The following lemma shows the correctness and the exactness of the grammar.

**Lemma 36** *For all $w \in \Sigma^*$, $f_G(w) = F_L(w)$.*

**Proof** *(Sketch)* The proof is in two steps: $f_G(w) \subseteq F_L(w)$ and $F_L(w) \subseteq f_G(w)$. Each step is made by induction on the length of $w$ and uses the rules created to build the grammar, the derivation process of a CBFG and the fiduciality for the second step.

First, we show $\forall w \in \Sigma^*, f_g(w) \subseteq F_L(w)$ by induction on the length of $w$. For $|w| = 1$, the inclusion is trivial since all the lexical rules $F_L(a) \rightarrow a$ are included in the grammar. Suppose that a string $w$, $|w| = n > 1$, is parsed by the CBFG $G$, then there exists a cut of $w$ in $uv = w$ and a rule $z \rightarrow xy$ in $G$ such that $x \subseteq f_G(u)$ and $y \subseteq f_G(v)$. By induction hypothesis, $x \subseteq F_L(u)$ and $y \subseteq F_L(v)$. By construction of the grammar, there exists two strings $u', v' \in K_L$ such that $u$, resp. $v$, belongs to same congruence class than $u'$, resp. $v'$ and the rule $F_L(u'v') \rightarrow F_L(u')F_L(v')$ belongs to the productions of the grammar. By induction hypothesis, $x \subseteq F_L(u) = F_L(u')$ and $y \subseteq F_L(v) = F_L(v')$ and thus $f_G(w) \subseteq F_L(w)$.
Second, we prove that $\forall w \in \Sigma^*, F_L(w) \subseteq f_G(w)$ by induction on the length of $w$. The key point relies on the fact that when a string $w$ is parsed by a CBFG $G$, there exists a cut of $w$ into $uv = w$ ($u$, $v \in \Sigma^*$) and a rule $z \rightarrow xy$ in $G$ such that $x \subseteq f_G(u)$ and $y \subseteq f_G(v)$. The rule $z \rightarrow xy$ is also obtained from a substring from the set used to build the grammar using the $F_L$ function. By the inductive hypothesis we obtain inclusion between $f_G$ and $F_L$ on $u$ and $v$. ■

For the language of Figure 5, the following set is sufficient to build an exact CBFG: $\{a, b, aa, ab, ba, aab, bb, bba\}$ (this corresponds to all the substrings of $aab$ and $bba$). We have:
$F_L(a) = F(L)\backslash\{(\lambda, \lambda), (a, \lambda)\} \rightarrow a$,
$F_L(b) = F(L) \rightarrow b$,
$F_L(aa) = F_L(a) \rightarrow F_L(a)F_L(a)$,
$F_L(ab) = F(L) \rightarrow F_L(a)F_L(b) = F_L(a)F(L)$,
$F_L(ba) = F(L) \rightarrow F_L(b)F_L(a) = F(L)F_L(a)$,

$F_L(bb) = F(L) \rightarrow F_L(b)F_L(b) = F(L)F(L)$,
$F_L(aab) = F_L(bba) = F_L(ab) = F_L(ba)$.

The approach presented here gives a canonical form for representing a regular language by an exact CBFG. Moreover, this is is *complete* in the sense that every context of every substring will be represented by some element of $F$: this CBFG will completely model the relation between contexts and substrings.

#### 6.1.2 Exact CBFGs do not include all CFLs

First, it is clear that the class of exact CBFGs includes some non-regular context-free languages: the grammar defined in Section 3.3 is an exact CBFG for the context-free and non regular language $\{a^n b^n | n > 0\}$, showing the class of exact CBFG has some elements properly in the class of CFGs.

We give now a context-free language $L$ that can not be defined by an exact CBFG:

$$L = \{a^n b | n > 0\} \cup \{a^n c^m | m > n > 0\}.$$

Suppose that there exists an exact CBFG that recognizes it and let $N$ be the length of the biggest feature (*i.e.* the longest left part of the feature). For any sufficiently large $k > N$, the sequences $c^k$ and $c^{k+1}$ share the same features: $F_L(c^k) = F_L(c^{k+1})$. Since the CBFG is exact we have $F_L(b) \subseteq F_L(c_k)$. Thus any derivation of $a^{k+1}b$ could be a derivation of $a^{k+1}c^k$ which does not belong to the language.

However, this restriction does not mean that the class of exact CBFG is too restrictive for modeling natural languages. Indeed, the example we have given is highly unnatural and such phenomena appear not to occur in attested natural languages.

#### 6.1.3 CBFG and Non Context-Free Languages

CBFGs are more powerful than CFGs in two respects. First, CBFGs can compactly represent languages like the finite language of all $n!$ permutations of an $n$-letter alphabet, that have no concise representation as a CFG (Asveld, 2006). Secondly, as we now show, there are some exact CBFGs that are not context-free. In particular, we define a language closely related to the *MIX language* (consisting of strings with an equal number of a's, b's and c's in any order) which is known to be non context-free, and indeed is conjectured to be outside the class of indexed grammars (Boullier, 2003).

Let $M = \{\{a, b, c\}^+\}$, the set of all strings of length at least one that can be built on the alphabet $\{a, b, c\}$. We consider now the language
$L = L_{abc} \cup L_{ab} \cup L_{ac} \cup \{a'a, b'b, c'c, dd', ee', ff'\}$:
$L_{ab} = \{wd | w \in M, |w|_a = |w|_b\}$,
$L_{ac} = \{we | w \in M, |w|_a = |w|_c\}$,
$L_{abc} = \{wf | w \in M, |w|_a = |w|_b = |w|_c\}$.
In order to define a CBFG recognizing $L$, we have to select features (contexts) that can represent exactly the intrinsic components of the languages composing $L$. We propose to use the following set of features for each sublanguage:

- For $L_{ab}$: $(\lambda, d)$ and $(\lambda, ad), (\lambda, bd)$.

- For $L_{ac}$: $(\lambda, e)$ and $(\lambda, ae), (\lambda, ce)$.
- For $L_{abc}$: $(\lambda, f')$.
- For the letters $a', b', c', a, b, c$ we add: $(\lambda, a), (\lambda, b), (\lambda, c), (a', \lambda), (b', \lambda), (c', \lambda)$.
- For the letters $d, e, f, d', e', f'$ we add; $(\lambda, d'), (\lambda, e'), (\lambda, f'), (d, \lambda), (e, \lambda), (f, \lambda)$.

Here, $L_{ab}$ will be represented by $(\lambda, d)$, but we will use $(\lambda, ad), (\lambda, bd)$ to define the internal derivations of elements of $L_{ab}$. The same idea holds for $L_{ac}$ with $(\lambda, e)$ and $(\lambda, ae), (\lambda, ce)$.

For the lexical rules and in order to have an exact CBFG, note the special case for $a, b, c$:
$\{(\lambda, bd), (\lambda, ce), (a', \lambda)\} \to a$
$\{(\lambda, ad), (b', \lambda)\} \to b$
$\{(\lambda, ae), (c', \lambda)\} \to c$
For the nine other letters, each one is defined with only one context, for example using the rule $\{(\lambda, d')\} \to d$.

For the production rules, the most important one is: $(\lambda, \lambda) \to \{(\lambda, d), (\lambda, e)\}, \{(\lambda, f')\}$.

Indeed, this rule, with the presence of two contexts in one of categories, means that an element of the language has to be derived so that it has a prefix $u$ such that $f_G(u) \supseteq \{(\lambda, d), (\lambda, e)\}$. This means $u$ is both an element of $L_{ab}$ and $L_{ac}$. This rule represents the language $L_{abc}$ since $\{(\lambda, f')\}$ can only represent the letter $f$.

The other parts of the language will be defined by the following rules:
$(\lambda, \lambda) \to \{(\lambda, d)\}, \{(\lambda, d')\}$,
$(\lambda, \lambda) \to \{(\lambda, e)\}, \{(\lambda, e')\}$,
$(\lambda, \lambda) \to \{(\lambda, a)\}, \{(\lambda, bd), (\lambda, ce), (a', \lambda)\}$,
$(\lambda, \lambda) \to \{(\lambda, b)\}, \{(\lambda, ad), (b', \lambda)\}$,
$(\lambda, \lambda) \to \{(\lambda, c)\}, \{(\lambda, ae), (c', \lambda)\}$,
$(\lambda, \lambda) \to \{(\lambda, d')\}, \{(d, \lambda)\}$,
$(\lambda, \lambda) \to \{(\lambda, e')\}, \{(e, \lambda)\}$,
$(\lambda, \lambda) \to \{(\lambda, f')\}, \{(f, \lambda)\}$.

This set of rules is incomplete, since for representing $L_{ab}$, the grammar must contain the rules ensuring to have the same number of a's and b's, and similarly for $L_{ac}$. To lighten the presentation here, the complete grammar is presented in Appendix.

We claim this is an exact CBFG for a context-sensitive language. $L$ is not context-free since if we intersect $L$ with the regular language $\Sigma^* d$, we get an instance of the non context-free MIX language (with $d$ appended). The exactness comes from the fact that we chose the contexts in order to ensure that strings belonging to a sublanguage can not belong to another one and that the derivation of a substring will provide all the possible correct features with the help of the union of all the possible derivations.

Note that the MIX language on its own is not definable by an exact CBFG: it is only when other parts of the language can distributionally define the appropriate partial structures that we can get context sensitive languages. Far from being a limitation of this formalism (a bug), we argue this is a feature: it is only in rather exceptional circumstances

that we will get properly context sensitive languages. This formalism thus potentially accounts not just for the existence of non context-free natural languages but also for their rarity.

### 6.2 Inexact CBFGs

We are less interested in the class of all CBFGs: these are CBFGs where the contexts are just arbitrary features and there is no relation between $f_G(u)$ and $C_L(u)$ except for the presence of $(\lambda, \lambda)$. However, it is important to understand the language theoretic power of this class as this upper bounds the hypothesis class of the algorithm, and is easier to analyse.

#### 6.2.1 Context-free grammars

First, we note that this class contains all context-free languages. Given a context-free language, that does not include the empty string, we can take a CFG in Chomsky normal form and convert it directly into a CBFG. Let $V$ be the set of non-terminals of such a CFG. We pick an arbitrary set of distinct contexts to represent the elements of $V$, subject only to the constraint that $S$ corresponds to $(\lambda, \lambda)$. Let $C(N)$ be the context corresponding to the non-terminal $N$. For every production rule in the CFG of the form $N \to PQ$, we add a CBFG production $\{C(N)\} \to \{C(P)\}, \{C(Q)\}$. For every production in the CFG of the form $N \to a$, we add a CBFG production to $P_L$ of the form $\{C(N)\} \to a$. It is easy to see that this will define the same language.

#### 6.2.2 Range Concatenation Grammars

While CBFG formalism has some relationship to a context-free grammar, and some to a semi-Thue system (also known as a string rewriting system), it is not formally identical to either of these. One exact equivalence is to a restricted subset of Range Concatenation Grammars; a very powerful formalism (Boullier, 2000). We include the following relationship, but suggest that the reader unfamiliar with RCGs proceeds to the discussion of the relationship with the more familiar class of context-free grammars.

**Lemma 37** *For every CBFG $G$, there is a non-erasing positive range concatenation grammar of arity one, in 2-var form that defines the same language.*

**Proof** Suppose $G = \langle F, P, P_L, \Sigma \rangle$. Define a RCG with a set of predicates equal to $F$ and the following clauses, and the two variables $U, V$. For each production $x \to yz$ in $P$, for each $f \in x$, where $y = \{g_1, \dots g_i\}$, $z = \{h_1, \dots h_j\}$ add clauses

$$f(UV) \to g_1(U), \dots g_i(U), h_1(V), \dots h_j(V).$$

For each lexical production $\{f_1 \dots f_k\} \to a$ add clauses

$$f_i(a) \to \epsilon.$$

It is straightforward to verify that $f(w) \vdash \epsilon$ iff $f \in f_G(w)$. ■

#### 6.2.3 Conjunctive Grammar

A tighter correspondence is to the class of Conjunctive Grammars (Okhotin, 2001), invented independently of RCGs.

**Definition 38** *A conjunctive grammar is defined as a quadruple $\langle \Sigma, N, P, S \rangle$, in which: $\Sigma$ is the alphabet; $N$ is the set of non terminal symbols; $P$ is the set of rules, each of the form $A \rightarrow \alpha_1 \& ... \& \alpha_m$, where $A \in V$ and $\forall i < m$, $\alpha_i \in (V \cup \Sigma)^*$; $S \in N$ is the start symbol.*

In this formalism, a string $w$ is derived from $A \in V$ iff there exists a rule $A \rightarrow \alpha_1 \& ... \& \alpha_m$ in $P$ and for all $i < m$, $\alpha_i$ derives $w$.

We claim that for every every language $L$ generated by a conjunctive grammar there is a CBFG representing $L\#$ (where the special character $\#$ is not included in the original alphabet).

Suppose we have a conjunctive grammar $G = \langle \Sigma, N, P, S \rangle$ in binary normal form (as defined in (Okhotin, 2003)). We construct the equivalent CBFG $G' = \langle F, P', P_L, \Sigma \rangle$ as followed:

- For every letter $a$ we add a context $(l_a, r_a)$ to $F$ such that $l_a a r_a \in L$;
- For every rules $X \rightarrow a$ in $P$, we create a rule $\{(l_a, r_a)\} \rightarrow a$ in $P_L$.
- For every non terminal $X \in N$, for every rule $X \rightarrow P_1 Q_1 \& \ldots \& P_n Q_n$ we add distinct contexts $\{(l_{P_i Q_i}, r_{P_i Q_i})\}$ to F, such that for all $i$ it exists $u_i$, $l_{P_i Q_i} u_i r_{P_i Q_i} \in L$ and $P_i Q_i \overset{*}{\Rightarrow}_G u_i$;
- Let $F_{X,j} = \{(l_{P_i Q_i}, r_{P_i Q_i}) : \forall i\}$ the set of contexts corresponding to the $j^{th}$ rule applicable to $X$. For all $(l_{P_i Q_i}, r_{P_i Q_i}) \in F_{X,j}$, we add to $P'$ the rules $(l_{P_i Q_i}, r_{P_i Q_i}) \rightarrow F_{P_i,k} F_{Q_i,l}$ $(\forall k, l)$.
- We add a new context $(w, \lambda)$ to $F$ such that $S \overset{*}{\Rightarrow}_G w$ and $(w, \lambda) \rightarrow \#$ to $P_L$;
- For all $j$, we add to $P'$ the rule $(\lambda, \lambda) \rightarrow F_{S,j}\{(w, \lambda)\}$.

It can be shown that this construction gives an equivalent CBFG.

## 7. Discussion and Conclusion

One of the main objective of our approach is to provide a framework that helps to bridge the gap between theoretical methods of grammatical inference and the structured representations required in linguistics. We provide a conclusion and a discussion of our work according to these two standpoints.

### 7.1 Grammatical Inference

In this paper, we have presented a new formalism the *Contextual Binary Feature Grammars* and shown its relevance for representing a large class of languages. We have proposed a learning algorithm using only membership queries and shown that this algorithm can identify in the limit the class of context-free languages satisfying the FCP and FKP assumptions.

First of all, we should establish how large the class of languages with the FCP and the FKP is: it includes all finite languages and all regular languages, since the set of congruence classes is finite for finite state languages. It similarly includes the context-free substitutable languages, (Clark and Eyraud, 2007), since every string in a substitutable language belongs to only one syntactic congruence class. As already stated it does not include all CFLs since not all CFLs have the FCP and/or the FKP. However it does include languages like the Dyck languages of arbitrary order, Lukacevic language and most other classic simple examples. As a special case consider the equivalence relation between contexts $f \cong_L f'$ iff $\forall u$ we have that $f \odot u \in L$ iff $f' \odot u \in L$. The class of CFLs where the context distribution of every string is a finite union of equivalence classes of contexts clearly has both the FKP and the FCP.

If we now focus on the algorithm proposed: it is relatively simple but has two main drawbacks. First, the algorithm is not conservative since once we have found the correct language, the representation may change – if the feature set found is not fiducial – until the fiduciality is reached. Second, the CBFG output by the algorithm may not be consistent with some answers provided by the oracle. Indeed, when the algorithm checks the fiduciality of the feature set $F$, the membership of new strings is tested. These strings do not appear in the list of learning examples given to the oracle but are built from all the possible contexts and substrings that can be extracted from this list. Then, it is possible that, among these new strings, some of them belong to the target language but are not recognized by the current grammar. In this case, the output grammar is nevertheless not modified. We can imagine a procedure that changes the grammar by adding these new positive strings for building the CBFG, however this could lead to having to deal with an exponential number of strings. Thus, a more reasonable procedure is to wait for these strings in the positive data presentation. One proposal for future work, from these two remarks, is a new learning algorithm that overcomes these drawbacks.

One important point is whether this result can be extended to a result which also bounds the number of samples as a polynomial function of the size of the representation. A preliminary result in this direction is presented in Clark (2010), which presents a polynomial result using the Minimally Adequate Teacher model of Angluin (1987). It seems likely that it will be possible to extend that result, which uses only context-free grammars, to the class of CBFGs.

Our approach to context-free grammatical inference is based on a generalisation of distributional learning, following the work of (Clark and Eyraud, 2007). The current state of the art in context-free inductive inference from flat unstructured examples only has been rather limited. When learning from stochastic data or using a membership oracle, it is possible to have powerful results, if we allow exponential computation (see for example (Horning, 1969)). The main contribution of this paper is to show that efficient learning is possible, with an appropriate representation. We currently rely on using a membership oracle, but under suitable assumptions about distributions, it should be possible to get a PAC-learning result for this class along the lines of (Clark, 2006), placing some bounds on the number of features required. Another interesting and needed issue is the adaptation of this approach to stochastic languages.

We have focused on context-free grammatical inference, however, we have shown that our representation is also relevant for modeling non context-free languages. Then, another

perspective of this work is to study learnability results for larger classes of languages. This would allow us to compare with other formalisms such as External Contextual Grammars (Boullier, 2001; Mitrana, 2005) and other learning methods dealing with non context-free languages (Oates et al., 2006; Yoshinaka, 2009).

### 7.2 Linguistics

The field of grammatical inference has close relations to the study of language acquisition. Attempts to model natural languages with context-free grammars require additional machinery: natural language categories such as noun phrases contain many overlapping subclasses with features such as case, number, gender and similarly for verbal categories. Modelling this requires either an exponential explosion of the number of non-terminals employed or a switch to a richer set of features. Our formalism can be seen as a first step to integrate such features. While we have implemented the algorithm described here, and verified that it works in accordance with theory on small artificial examples, there are a number of modifications that would need to be made before it can be applied to real grammar induction on natural language. First, the algorithm is very naive; in practice a more refined algorithm could select both the kernel and the feature set in a more sophisticated way. Secondly, considering features that correspond to individual contexts may be too narrow a definition for natural language given the well known problems of data sparseness and it will be necessary to switch to features corresponding to sets of contexts, which may overlap. Thus for example one might have features that correspond to sets of contexts of the form $F(u, v) = \{(lu, vr) | l, r \in \Sigma^*\}$. This would take this approach closer to methods that have been shown to be effective in unsupervised learning in NLP (Klein and Manning, 2004) where typically $|u| = |v| = 1$. In any event, we think such modifications will be necessary for the acquisition of non context-free languages. Finally, at the moment the algorithm has polynomial update time, but in the worst case, there are deterministic finite state automata such that the size of the smallest kernel will be exponential in the number of states. There are, however, natural algorithms for generalising the productions by removing features from the right hand sides of the rules; this would have the effect of accelerating the convergence of the algorithm, and removing the requirement for the Finite Kernel Property.

## Acknowledgments

This work was supported in part by the IST Programme of the European Community, under the PASCAL2 Network of Excellence, IST-2007-216886. This publication only reflects the authors' views.

We would like to thank Ryo Yoshinaka for very helpful discussions, and the anonymous reviewers for suggestions and comments that have greatly improved the paper. We also want to thank Jean-Marie Madiot who worked on the first version of the experiments presented in this paper.

## Appendix

We give here an explicit exact CBFG for the following non context-free language

$$L = L_{abc} \cup L_{ab} \cup L_{ac} \cup \{a'a, b'b, c'c, dd', ee', ff'\}$$

defined on the alphabet $\Sigma = \{a, b, c, d, e, f, a', b', c', d', e', f'\}$ and such that:
$L_{ab} = \{wd | w \in \{a, b, c\}^+, |w|_a = |w|_b\}$,
$L_{ac} = \{we | w \in \{a, b, c\}^+, |w|_a = |w|_c\}$,
$L_{abc} = \{wf | w \in \{a, b, c\}^+, |w|_a = |w|_b = |w|_c\}$.

Here is the list of productions of the grammar:
$(\lambda, \lambda) \to \{(\lambda, d), (\lambda, e)\}, \{(\lambda, f')\}$
$(\lambda, \lambda) \to \{(\lambda, d)\}, \{(\lambda, d')\}$
$(\lambda, \lambda) \to \{(\lambda, e)\}, \{(\lambda, e')\}$
$(\lambda, \lambda) \to \{(\lambda, a)\}, \{(\lambda, bd), (\lambda, ce), (a', \lambda)\}$
$(\lambda, \lambda) \to \{(\lambda, b)\}, \{(\lambda, ad), (b', \lambda)\}$
$(\lambda, \lambda) \to \{(\lambda, c)\}, \{(\lambda, ae), (c', \lambda)\}$
$(\lambda, \lambda) \to \{(\lambda, d')\}, \{(d, \lambda)\}$
$(\lambda, \lambda) \to \{(\lambda, e')\}, \{(e, \lambda)\}$
$(\lambda, \lambda) \to \{(\lambda, f')\}, \{(f, \lambda)\}$

$(\lambda, d) \to \{(\lambda, d)\}, \{(\lambda, d)\}$
$(\lambda, d) \to \{(\lambda, ad)\}, \{(\lambda, bd)\}$
$(\lambda, d) \to \{(\lambda, bd)\}, \{(\lambda, ad)\}$
$(\lambda, d) \to \{(\lambda, d)\}, \{(\lambda, ad), (\lambda, ae), (c', \lambda)\}$
$(\lambda, d) \to \{(\lambda, ad), (\lambda, ae), (c', \lambda)\}, \{(\lambda, d)\}$

$(\lambda, ad) \to \{(\lambda, ad), (\lambda, ae), (c', \lambda)\}, \{(\lambda, ad)\}$
$(\lambda, ad) \to \{(\lambda, ad)\}, \{(\lambda, ad), (\lambda, ae), (c', \lambda)\}$
$(\lambda, ad) \to \{(\lambda, ad), (b', \lambda)\}, \{(\lambda, d)\}$
$(\lambda, ad) \to \{(\lambda, d)\}, \{(\lambda, ad), (b', \lambda)\}$

$(\lambda, bd) \to \{(\lambda, ad), (\lambda, ae), (c', \lambda)\}, \{(\lambda, bd)\}$
$(\lambda, bd) \to \{(\lambda, bd)\}, \{(\lambda, ad), (\lambda, ae), (c', \lambda)\}$
$(\lambda, bd) \to \{(\lambda, bd), (\lambda, ce), (a', \lambda)\}, \{(\lambda, d)\}$
$(\lambda, bd) \to \{(\lambda, d)\}, \{(\lambda, bd), (\lambda, ce), (a', \lambda)\}$

$(\lambda, e) \to \{(\lambda, e)\}, \{(\lambda, e)\}$
$(\lambda, e) \to \{(\lambda, ae)\}, \{(\lambda, ce)\}$
$(\lambda, e) \to \{(\lambda, ce)\}, \{(\lambda, ae)\}$
$(\lambda, e) \to \{(\lambda, e)\}, \{(\lambda, ad), (b', \lambda)\}$
$(\lambda, e) \to \{(\lambda, ad), (b', \lambda)\}, \{(\lambda, e)\}$

$(\lambda, ae) \to \{(\lambda, ad), (b', \lambda)\}, \{(\lambda, ae)\}$
$(\lambda, ae) \to \{(\lambda, ae)\}, \{(\lambda, ad), (b', \lambda)\}$

$(\lambda, ae) \to \{(\lambda, ad), (\lambda, ae), (c', \lambda)\}, \{(\lambda, e)\}$
$(\lambda, ae) \to \{(\lambda, e)\}, \{(\lambda, ad), (\lambda, ae), (c', \lambda)\}$

$(\lambda, ce) \to \{(\lambda, ad), (b', \lambda)\}, \{(\lambda, ce)\}$
$(\lambda, ce) \to \{(\lambda, ce)\}, \{(\lambda, ad), (b', \lambda)\}$
$(\lambda, ce) \to \{(\lambda, bd), (\lambda, ce), (a', \lambda)\}, \{(\lambda, e)\}$
$(\lambda, ce) \to \{(\lambda, e)\}, \{(\lambda, bd), (\lambda, ce), (a', \lambda)\}$

$\{(\lambda, bd), (\lambda, ce), (a', \lambda)\} \to a$
$\{(\lambda, ad), (b', \lambda)\} \to b$
$\{(\lambda, ae), (c', \lambda)\} \to c$
$\{(\lambda, d')\} \to d$
$\{(\lambda, e')\} \to e$
$\{(\lambda, f')\} \to f$

$\{(\lambda, a)\} \to a'$
$\{(\lambda, b)\} \to b'$
$\{(\lambda, c)\} \to c'$
$\{(d, \lambda)\} \to d'$
$\{(e, \lambda)\} \to e'$
$\{(f, \lambda)\} \to f'$.

## References


P. Adriaans. Learning shallow context-free languages under simple distributions. *Algebras, Diagrams and Decisions in Language, Logic and Computation*, 127, 2002.

D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

D. Angluin. Queries and concept learning. *Mach. Learn.*, 2(4):319–342, 1988. ISSN 0885-6125. doi: http://dx.doi.org/10.1023/A:1022821128753.

P.R.J. Asveld. Generating all permutations by context-free grammars in Chomsky normal form. *Theoretical Computer Science*, 354(1):118–130, 2006.

L. Boasson and G. Senizergues. NTS languages are deterministic and congruential. *J. Comput. Syst. Sci.*, 31(3):332–342, 1985. ISSN 0022-0000. doi: http://dx.doi.org/10.1016/0022-0000(85)90056-X.

P. Boullier. From contextual grammars to range concatenation grammars. *Electronic Notes on Theoretical Computer Science*, 53, 2001.

P. Boullier. A Cubic Time Extension of Context-Free Grammars. *Grammars*, 3:111–131, 2000.

P. Boullier. Counting with range concatenation grammars. *Theoretical Computer Science*, 293(2):391–416, 2003.

R.C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In R.C. Carrasco and J. Oncina, editors, *Proceedings ICGI'94*, volume 862 of *LNAI*, pages 139–150. Springer, 1994.

N. Chomsky. *Knowledge of Language : Its Nature, Origin, and Use*. Praeger, 1986.

A. Clark. Distributional learning of some context-free languages with a minimally adequate teacher. In José M. Sempere and Pedro García, editors, *Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI*, pages 24–37. Springer, 2010.

A. Clark. PAC-learning unambiguous NTS languages. In *Proceedings of the 8th International Colloquium on Grammatical Inference (ICGI)*, pages 59–71, 2006.

A. Clark. A learnable representation for syntax using residuated lattices. In *Proceedings of the conference on Formal Grammar*, Bordeaux, France, 2009.

A. Clark and R. Eyraud. Polynomial identification in the limit of substitutable context-free languages. *Journal of Machine Learning Research*, 8:1725–1745, 2007.

C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27(2):125–138, 1997. ISSN 0885-6125.

F. Denis, A. Lemay, and A. Terlutte. Learning regular languages using rfsas. *Theor. Comput. Sci.*, 313(2):267–294, 2004. ISSN 0304-3975.

R. Eyraud, C. de la Higuera, and J.C. Janodet. Lars: A learning algorithm for rewriting systems. *Machine Learning*, 66(1):7–31, 2007.

G. Gazdar, E. Klein, G. Pullum, and I. Sag. *Generalised Phrase Structure Grammar*. Basil Blackwell, 1985.

E.M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

Z. Harris. Distributional structure. *Word*, 10(2-3):146–62, 1954.

C. De La Higuera and J. Oncina. Inferring deterministic linear languages. In *COLT '02: 15th Annual Conference on Computational Learning Theory*, pages 185–200. Springer-Verlag, 2002.

J.J. Horning. *A Study of Grammatical Inference*. PhD thesis, Stanford University, Computer Science Department, California, 1969.

A.K. Joshi and Y. Schabes. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–124. Springer, Berlin, New York, 1997.

D. Klein and C.D. Manning. Corpus-based induction of syntactic structure: models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pages 478–485, 2004.

K.J. Lang, B.A. Pearlmutter, and R. Price. Results of the abbadingo one dfa learning competition and a new evidence driven state merging algorithm. In *Grammatical Inference: Algorithms and Applications; 4th International Colloquium on Grammatical Inference*, pages 1–12. Springer-Verlag, 1998. URL `http://abbadingo.cs.unm.edu/`.

P. Langley and S. Stromsten. Learning context-free grammars with a simplicity bias. In *Proceedings of the Eleventh European Conference on Machine Learning*, pages 220–228. Springer-Verlag, 2000.

S. Marcus. *Algebraic Linguistics; Analytical Models*. Academic Press, N. Y., 1967.

V. Mitrana. Marcus external contextual grammars: From one to many dimensions. *Fundamenta Informaticae*, 54:307–316, 2005.

K. Nakamura and M. Matsumoto. Incremental learning of context-free grammars based on bottom-up parsing and search. *Pattern Recognition*, 38(9):1384–1392, 2005.

T. Oates, T. Armstrong, L. Becerra-Bonache, and M. Atamas. Inferring grammars for mildly context sensitive languages in polynomial-time. In *Proceedings of the International Colloquium on Grammatical Inference (ICGI) 2006*, volume 4201 of *LNCS*, pages 137–147. Springer, 2006.

A. Okhotin. Conjunctive grammars. *J. Autom. Lang. Comb.*, 6(4):519–535, 2001. ISSN 1430-189X.

A. Okhotin. An overview of conjunctive grammars. *Formal Language Theory Column, bulletin of the EATCS*, 79:145–163, 2003.

R. Parekh and V. Honavar. An incremental interactive algorithm for regular grammar inference. In *Grammatical Inference : Learning Syntax from Sentences; 3rd International Colloquium on Grammatical Inference*, pages 238–250. Springer-Verlag, 1996.

L. Pitt. Inductive inference, dfa's, and computational complexity. In *Lecture Notes in Artificial Intelligence*, pages 8–14. Springer-Verlag, 1989.

B. Starkie, F. Coste, and M. Van Zaanen. The omphalos context-free grammar learning competition. In *Grammatical Inference: Algorithms and Applications; 7th International Colloquium on Grammatical Inference*, pages 16–27. Springer, 2004. URL `http://www.irisa.fr/Omphalos/`.

B. Starkie, M. van Zaanen, and D. Estival. The tenjinno machine translation competition. In *Grammatical Inference: Algorithms and Applications, 8th International Colloquium on Grammatical Inference*, volume 4201 of *Lecture Notes in Computer Science*, pages 214–226. Springer-Verlag, 2006.

T. Yokomori. Polynomial-time identification of very simple grammars from positive data. *Theoretical Computer Science*, 298(1):179–206, 2003.

R. Yoshinaka. Learning mildly context-sensitive languages with multidimensional substitutability from positive data. In *Proceedings of the 20th International Conference on Algorithmic Learning Theory (ALT)*, volume 5809 of *LNCS*, pages 278–292. Springer, 2009.

R. Yoshinaka. Identification in the limit of k,l-substitutable context-free languages. In *ICGI '08: Proceedings of the 9th international colloquium on Grammatical Inference*, pages 266–279, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-88008-0.