A note on contextual binary feature grammars

Alexander Clark
Department of Computer Science
Royal Holloway, University of London
alexc@cs.rhul.ac.uk

Rémi Eyraud and **Amaury Habrard**
Laboratoire d'Informatique Fondamentale
de Marseille, CNRS,
Aix-Marseille Université, France
remi.eyraud,amaury.habrard@lif.univ-mrs.fr

Abstract

Contextual Binary Feature Grammars were recently proposed by (Clark et al., 2008) as a learnable representation for richly structured context-free and context sensitive languages. In this paper we examine the representational power of the formalism, its relationship to other standard formalisms and language classes, and its appropriateness for modelling natural language.

1 Introduction

An important issue that concerns both natural language processing and machine learning is the ability to learn suitable structures of a language from a finite sample. There are two major points that have to be taken into account in order to define a learning method useful for the two fields: first the method should rely on intrinsic properties of the language itself, rather than syntactic properties of the representation. Secondly, it must be possible to associate some semantics to the structural elements in a natural way.

Grammatical inference is clearly an important technology for NLP as it will provide a foundation for theoretically well-founded unsupervised learning of syntax, and thus avoid the annotation bottleneck and the limitations of working with small hand-labelled treebanks.

Recent advances in context-free grammatical inference have established that there are large learnable classes of context-free languages. In this paper, we focus on the basic representation used by the recent approach proposed in (Clark et al., 2008). The authors consider a formalism called *Contextual Binary Feature Grammars (CBFG)* which defines a class of grammars using contexts as features instead of classical non terminals. The use of features is interesting from an NLP point of view because we can associate some semantics to them, and because we can represent complex, structured syntactic categories. The notion of contexts is relevant from a grammatical inference standpoint since they are easily observable from a finite sample. In this paper we establish some basic language theoretic results about the class of exact Contextual Binary Feature Grammars (defined in Section 3), in particular their relationship to the Chomsky hierarchy: exact CBFGs are those where the contextual features are associated to all the possible strings that can appear in the corresponding contexts of the language defined by the grammar.

The main results of this paper are proofs that the class of exact CBFGs:

- properly includes the regular languages (Section 5),
- does not include some context-free languages (Section 6),
- and does include some non context-free languages (Section 7).

Thus, this class of exact CBFGs is orthogonal to the classic Chomsky hierarchy but can represent a very large class of languages. Moreover, it has been shown that this class is efficiently learnable. This class is therefore an interesting candidate for modeling natural language and deserves further investigation.

2 Basic Notation

We consider a finite alphabet Σ, and Σ^* the free monoid generated by Σ. λ is the empty string, and a language is a subset of Σ^*. 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 $v = lur$.

Proceedings of the EACL 2009 Workshop on Computational Linguistic Aspects of Grammatical Inference, pages 33–40,
Athens, Greece, 30 March 2009.

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. A context is also known in structuralist linguistics as an *environment*.

The set of contexts, or *distribution*, of a string u of 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. We define the *syntactic congruence* as $u \equiv_L v$ iff $C_L(u) = C_L(v)$. The equivalence classes under this relation are the *congruence* classes of the language. In general we will assume that λ is not a member of any language.

3 Contextual Binary Feature Grammars

Most definitions and lemmas of this section were first introduced in (Clark et al., 2008).

3.1 Definition

Before the presentation of the formalism, we give some results about contexts to help to give an intuition of 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 1. *For any language L and for any strings u, u', v, v' if $C(u) = C(u')$ and $C(v) = C(v')$, then $C(uv) = C(u'v')$.*

We can also consider a slightly stronger result:

Lemma 2. *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 $C(uv) \subseteq C(u'v')$.*

$C(u) \subseteq C(u')$ means that we can replace any occurrence of u in a sentence, 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[1]. Thus given a sentence "I gave him away", we can substitute *it* for *him*, 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 2 we can also say that, if we have some finite set of strings K, where we know the contexts, then:

Corollary 1.

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

This is the basis of the 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). In order to illustrate this concept, we give here a simple example.

Consider the language $\{a^n b^n | n > 0\}$ and the set $K = \{aabb, ab, abb, aab, a, b\}$. Suppose we want to compute the set of contexts of $aaabbb$, Since $C(abb) \subseteq C(aabbb)$, and vacuously $C(a) \subseteq C(a)$, we know that $C(aabb) \subseteq C(aaabbb)$. More generally, the contexts of ab can represent $a^n b^n$, those of aab the strings $a^{n+1} b^n$ and the ones of abb the strings $a^n b^{n+1}$.

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

Definition 1. *A Contextual Binary Feature Grammar (CBFG) G is a tuple $\langle F, P, P_L, \Sigma \rangle$. F is a finite set of contexts, called features, where we write $C = 2^F$ for the power set of F defining the categories of the grammar, $P \subseteq C \times C \times C$ is a finite set of productions that we write $x \to yz$ where $x, y, z \in C$ and $P_L \subseteq C \times \Sigma$ is a set of lexical rules, written $x \to a$.*

Normally P_L contains exactly one production for each letter in the alphabet (the lexicon).

A CBFG G defines recursively a map f_G

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

from $\Sigma^* \to C$ as follows:

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

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

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

We give here more explanation about the map f_G. It defines in fact the analysis of a string by a CBFG. A rule $z \to xy$ is applied to analyse a string w if there is a cut $uv = w$ s.t. $x \subseteq f_G(u)$ and $y \subseteq f_G(v)$, recall that x and y are sets of contexts. Intuitively, the relation given by the production rule is linked with Lemma 2: z is included in the set of features of $w = uv$. From this relationship, for any $(l, r) \in z$ we have $lwr \in L(G)$.

The complete computation of f_G is then justified by Corollary 1: $f_G(w)$ defines all the possible features associated by G to w with all the possible cuts $uv = w$ (*i.e.* all the possible derivations).

Finally, the natural way to define the membership of a string w in $L(G)$ is to have the context $(\lambda, \lambda) \in f_G(w)$ which implies that $\lambda u \lambda = u \in L(G)$.

Definition 2. *The language defined by a CBFG G is the set of all strings that are assigned the empty context: $L(G) = \{u | (\lambda, \lambda) \in f_G(u)\}$.*

As we saw before, 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. From the basic definition of a CBFG, we do not require any specific condition on the features of the grammar, except that a feature is associated to a string if the string appears in the context defined by the feature. However, we can also require that f_G defines exactly all the possible features that can be associated to a given string according to the underlying language.

Definition 3. *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 map $F_L : \Sigma^* \to 2^F$ which is just the map $u \mapsto \{(l, r) \in F | lur \in L\} = C_L(u) \cap F$.*

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

Definition 4. *A CBFG G is* exact *if for all $u \in \Sigma^*$, $f_G(u) = F_{L(G)}(u)$.*

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 4). However, exactness is an important notion because it allows to associate intrinsic components of a language to strings. 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. A basic example of such a learning algorithm is given in (Clark et al., 2008).

3.2 A Parsing Example

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

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

The 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 correspond 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 $(\lambda, \lambda) \in f_G(w) \Leftrightarrow w \in \{a^n b^n : n > 0\}$.

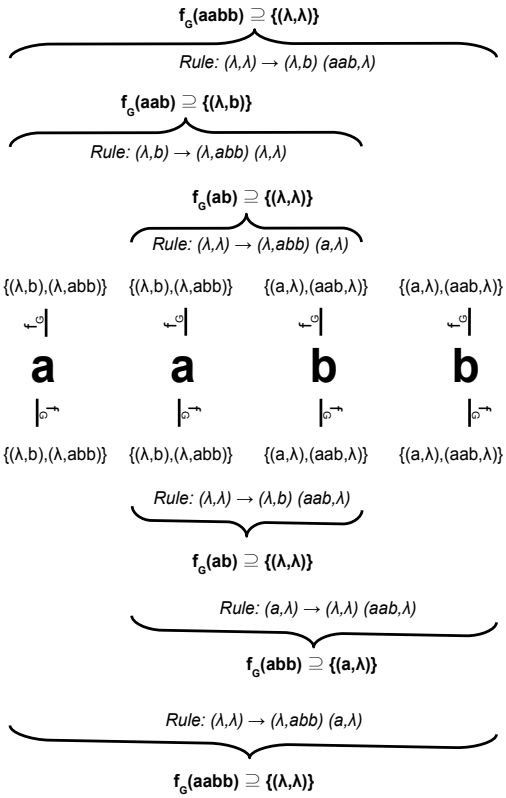

Figure 1: The two derivations to obtain (λ, λ) in $f_G(aabb)$ in the grammar G.

This is a simple example that illustrates the parsing of a string given a CBFG. This example does not characterize the power of CBFG since no right handside part is composed of more than one context. A more interesting, example with a context-sensitive language, will be presented in Section 7.

4 Non exact CBFGs

The aim here is to study the expressive power of CBFG compare to other formalism recently introduced. Though the inference can be done only for exact CBFG, where features are directly linked with observable contexts, it is still worth having a look at the more general characteristics of CBFG. For instance, it is interesting to note that several formalisms introduced with the aim of representing natural languages share strong links with CBFG.

Range Concatenation Grammars

Range Concatenation Grammars are a very powerful formalism (Boullier, 2000), that is a current area of research in NLP.

Lemma 3. *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 $x \rightarrow yz$ in P, for each $f \in x$, where $y = \{g_1, \dots g_i\}$, $z = \{h_1, \dots h_j\}$ add clauses
$f(UV) \rightarrow g_1(U), \dots g_i(U), h_1(V), \dots h_j(V)$.
For each lexical production $\{f_1 \dots f_k\} \rightarrow a$ add clauses $f_i(a) \rightarrow \epsilon$. It is straightforward to verify that $f(w) \vdash \epsilon$ iff $f \in f_G(w)$. □

Conjunctive Grammar

A more exact correspondence is to the class of Conjunctive Grammars (Okhotin, 2001), invented independently of RCGs. 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 $X \rightarrow P_1 Q_1 \& \dots \& P_n Q_n$ 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 $P_i Q_i \stackrel{*}{\Rightarrow}_G u_i$;
- 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

$(l_{P_iQ_i}, r_{P_iQ_i}) \in F_{X,j}$, we add to P' the rules $(l_{P_iQ_i}, r_{P_iQ_i}) \to F_{P_i,k}F_{Q_i,l}$ $(\forall k, l)$.

- We add a new context (w, λ) to F such that $S \stackrel{*}{\Rightarrow}_G w$ and $(w, \lambda) \to \#$ to P_L;
- For all j, we add to P' the rule $(\lambda, \lambda) \to F_{S,j}\{(w, \lambda)\}$.

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

5 Regular Languages

Any regular language can be defined by an exact CBFG. In order to show this we will propose an approach defining a canonical form for representing any regular language.

Suppose we have a regular language L, we consider the left and right residual languages:

$$u^{-1}L = \{w | uw \in L\} \quad (4)$$

$$Lu^{-1} = \{w | wu \in L\} \quad (5)$$

They define two congruencies: if $l, l' \in u^{-1}L$ (resp. $r, r' \in Lu^{-1}$) then for all $w \in \Sigma^*$, $lw \in L$ iff $l'w \in L$ (resp. $wr \in L$ iff $wr' \in L$).

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 Myhil-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:

$$F(L) = L_{min} \times R_{min}. \quad (6)$$

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

If we consider the regular language defined by the deterministic finite automata of Figure 2, 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 (using arguments about congruence classes) that for any strings u, v such that $F_L(u) \supset F_L(v)$, then $C_L(u) \supset C_L(v)$. This means the set of feature F is sufficient to represent context inclusion, we call this property the *fiduciality*.

Note that the number of congruence classes of a regular language is finite. Each congruence class is represented by a set of contexts

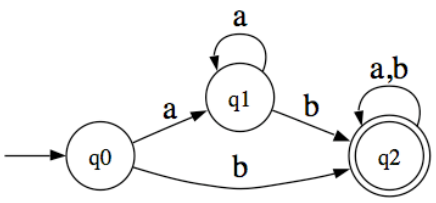

Figure 2: Example of a DFA. The left residuals are defined by $\lambda^{-1}L$, $a^{-1}L$, $b^{-1}L$ and the right ones by $L\lambda^{-1}$, Lb^{-1}, Lab^{-1} (note here that $La^{-1} = L\lambda^{-1}$).

$F_L(u)$. 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

$$F_L(uv) \to F_L(u), F_L(v) \quad (7)$$

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

Lemma 4. *For all $w \in \Sigma^*$, $f_G(w) = F_L(w)$.*

Proof. (Sketch) Proof in two steps: $\forall w \in \Sigma^*, F_L(w) \subseteq f_G(w)$ and $f_G(w) \subseteq F_L(w)$. 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. The key point rely on the fact that when a string w is parsed by a CBFG G, there exists a cut of w in $uv = w$ ($u, v \in \Sigma^*$) and a rule $z \to xy$ in G such that $x \subseteq f_G(u)$ and $y \subseteq f_G(v)$. The rule $z \to xy$ is also obtained from a substring from the set used to build the grammar using the F_L map. By inductive hypothesis you obtain inclusion between f_G and F_L on u and v. □

For the language of Figure 2, the following set is sufficient to build an exact CBGF: $\{a, b, aa, ab, ba, aab, bb, bba\}$ (this corresponds to all the substrings of aab and bba). We have:
$F_L(a) = F(L)\backslash\{(\lambda, \lambda), (a, \lambda)\} \to a$
$F_L(b) = F(L) \to b$
$F_L(aa) = F_L(a) \to F_L(a), F_L(a)$
$F_L(ab) = F(L) \to F_L(a), F_L(b) = F_L(a), F(L)$
$F_L(ba) = F(L) \to F_L(b), F_L(a) = F(L), F_L(a)$
$F_L(bb) = F(L) \to F_L(b), F_L(b) = F(L), F(L)$

$F_L(aab) = F_L(bba) = F_L(ab) = F_L(ba)$

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 $F(L)$: this CBFG will completely model the relation between contexts and substrings.

6 Context-Free Languages

We now consider the relationship between CFGs and CBFGs.

Definition 5. *A context-free grammar (CFG) is a quadruple $G = (\Sigma, V, P, S)$. Σ is a finite alphabet, V is a set of non terminals ($\Sigma \cap V = \emptyset$), $P \subseteq V \times (V \cup \Sigma)^+$ is a finite set of productions, $S \in V$ is the start symbol.*

In the following, we will suppose that a CFG is represented in Chomsky Normal Form, *i.e.* every production is in the form $N \to UW$ with $N, U, W \in V$ or $N \to a$ with $a \in \Sigma$.
We will write $uNv \Rightarrow_G u\alpha v$ if there is a production $N \to \alpha \in P$. $\overset{*}{\Rightarrow}_G$ is the reflexive transitive closure of $\Rightarrow_G$. The language defined by a CFG G is $L(G) = \{w \in \Sigma^* | S \overset{*}{\Rightarrow}_G w\}$.

6.1 A Simple Characterization

A simple approach to try to represent a CFG by a CBFG is to define a bijection between the set of non terminals and the set of context features. Informally we define each non terminal by a single context and rewrite the productions of the grammar in the CBFG form.

To build the set of contexts F, it is sufficient to choose $|V|$ contexts such that a bijection b_C can be defined between V and F with $b_C(N) = (l, r)$ implies that $S \overset{*}{\Rightarrow} lNr$. Note that we fix $b_T(S) = (\lambda, \lambda)$.

Then, we can define a CBFG $\langle F, P', P'_L, \Sigma \rangle$, where $P' = \{b_T(N) \to b_T(U)b_T(W) | N \to UW \in P\}$ and $P'_L = \{b_T(N) \to a | N \to a \in P, a \in \Sigma\}$. A similar proof showing that this construction produces an equivalent CBFG can be found in (Clark et al., 2008).

If this approach allows a simple syntactical convertion of a CFG into a CBFG, it is not relevant from an NLP point of view. Though we associate a non-terminal to a context, this may not correspond to the intrinsic property of the underlying language. A context could be associated with many non-terminals and we choose only one. For example, the context $(He\ is, \lambda)$ allows both noun phrases and adjective phrases. In formal terms, the resulting CBFG is not exact. Then, with the bijection we introduced before, we are not able to characterize the non-terminals by the contexts in which they could appear. This is clearly what we don't want here and we are more interested in the relationship with exact CBFG.

6.2 Not all CFLs have an exact CBFG

We will show here that the class of context-free grammars is not strictly included in the class of exact CBFGs. First, the grammar defined in Section 3.2 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 in the class of CFGs.

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

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

Suppose that there exists an exact CBFG that recognizes it and let N be the length of the biggest feature (*i.e.* the longuest 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 modelling natural languages. Indeed, the example we have given is highly unnatural and such phenomena appear not to occur in attested natural languages.

7 Context-Sensitive Languages

We now show that 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 $M = \{(a, b, c)^*\}$, we consider 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 sublanguages:

- For L_{ab}: (λ, d) and $(\lambda, ad), (\lambda, bd)$.
- For L_{ac}: (λ, e) and $(\lambda, ae), (\lambda, ce)$.
- For L_{abc}: (λ, 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 (λ, 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 (λ, 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:
$\{(\lambda, bd), (\lambda, ce), (a', \lambda)\} \to a$
$\{(\lambda, ad), (b', \lambda)\} \to b$
$\{(\lambda, ad), (\lambda, ae), (c', \lambda)\} \to c$
For the nine other letters, each one is defined with only one context like $\{(\lambda, d')\} \to d$.

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

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 Annex.

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 probably 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 language but also for their rarity.

8 Conclusion

The chart in Figure 3 summarises the different relationship shown in this paper. The substitutable languages (Clark and Eyraud, 2007) and the very simple ones (Yokomori, 2003) form two different learnable class of languages. There is an interesting relationship with Marcus External Contextual Grammars (Mitrana, 2005): if we defined the language of a CBFG to be the set $\{f_G(u) \odot u : u \in \Sigma^*\}$ we would be taking some steps towards contextual grammars.

In this paper we have discussed the weak generative power of Exact Contextual Binary Feature Grammars; we conjecture that the class of natural language stringsets lie in this class. ECBFGs are efficiently learnable (see (Clark et al., 2008) for details) which is a com-

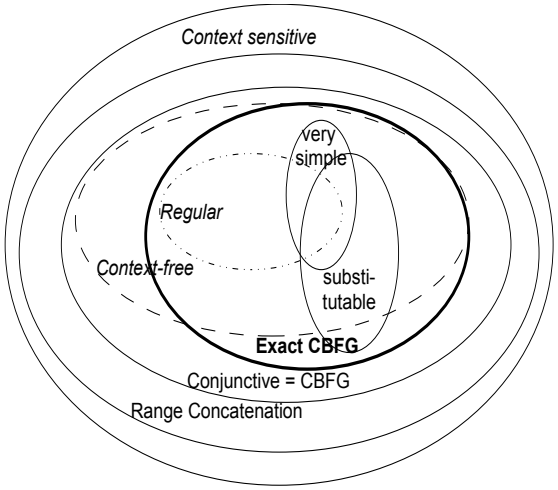

Figure 3: The relationship between CBFG and other classes of languages.

pelling technical advantage of this formalism over other more traditional formalisms such as CFGs or TAGs.

References

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

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

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

Alexander Clark, Rémi Eyraud, and Amaury Habrard. 2008. A polynomial algorithm for the inference of context free languages. In *Proceedings of International Colloquium on Grammatical Inference*, pages 29–42. Springer, September.

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

Alexander Okhotin. 2001. Conjunctive grammars. *J. Autom. Lang. Comb.*, 6(4):519–535.

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

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

Annex

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

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

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

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