Learning Context Free Grammars with the Syntactic Concept Lattice

Alexander Clark

Department of Computer Science
Royal Holloway, University of London
Egham, TW20 0EX
alexc@cs.rhul.ac.uk

Abstract. The Syntactic Concept Lattice is a residuated lattice based on the distributional structure of a language; the natural representation based on this is a context sensitive formalism. Here we examine the possibility of basing a context free grammar (CFG) on the structure of this lattice; in particular by choosing non-terminals to correspond to concepts in this lattice. We present a learning algorithm for context free grammars which uses positive data and membership queries, and prove its correctness under the identification in the limit paradigm. Since the lattice itself may be infinite, we consider only a polynomially bounded subset of the set of concepts, in order to get an efficient algorithm. We compare this on the one hand to learning algorithms for context free grammars, where the non-terminals correspond to congruence classes, and on the other hand to the use of context sensitive techniques such as Binary Feature Grammars and Distributional Lattice Grammars. The class of CFGs that can be learned in this way includes inherently ambiguous and thus non-deterministic languages; this approach therefore breaks through an important barrier in CFG inference.

1 Introduction

In recent years, grammatical inference has started to moved from the learnability of regular languages onto the study of the inference of context free languages. The approach developed by Clark and Eyraud [1] is one active research direction: they consider defining context free grammars where the non-terminals correspond to the congruence classes of the language and are able to demonstrate a learnability result for the class of substitutable languages. Though this class is small, the result is significant and has already lead to a number of extensions [2–4]. Given a richer source of data, including membership queries, it is possible to increase the class of languages learned, while maintaining this basic representational assumption (see [5] in this volume).

The limitations of the pure congruence approach are however quite strict. Though it includes many standard languages, such as the Dyck language and so on, there are many simple languages which are not in the class. Consider the following language [6]

$$L_2 = \{a^n b^n | n \geq 0\} \cup \{a^n b^{2n} | n \geq 0\}$$

This is clearly a context free language. It is easy to show, using a pumping argument, that any CFG for this language must have a non-terminal that generates an infinite set of strings of the form $\{a^{p+qn}b^{qn}|n \geq 0 \wedge (p+qn) \geq 0\}$ for some positive q and some integer value p. None of these strings are congruent to each other. Moreover, since the language itself is a union of infinitely many congruence classes, it is immediate that it is not in the class of languages that we can learn using a pure congruential class. Similarly, the language of all odd and even length palindromes over $\{a, b\}$ has the property that no two distinct strings are congruent, and thus a congruence based approach will also fail.

In related work, [7] extends this work by switching to a more powerful context sensitive representation; [8] bases a more powerful approach in the theory of residuated lattices. That work is motivated by the problems of linguistics – context free languages are well known to be inadequate for natural language syntax. However there are other non-linguistic areas where grammatical inference is relevant, and in those areas, it may be worthwhile restricting the representations to context free grammars for external reasons – for example one might have prior knowledge the representations are in fact CFGs. Moreover, there are problems involved with generating from these context-sensitive representations, and with making stochastic variants of them, whereas these problems are well understood in the field of CFGs [9]. Therefore, even though CFGs may not be the right representation for natural languages, it is still worth studying their learnability under various paradigms.

The question is then how can we increase the class of context free languages that we can learn using the same family of distributional techniques, while still keeping a CFG as a representation. Alternatively, we can ask whether we really need to use the context-sensitive grammar formalisms if all we are really interested in is languages which are in fact context free. The goal of this paper is to take the lattice-based techniques of [8] and use them to push CFG inference techniques as far as we can.

The basic strategy is to define various sets of strings that we will correspond to the non-terminals, or more precisely to the set of strings generated by a non-terminal. We consider a finite set of possibly infinite sets of strings, $\mathcal{C}$. If we have that $PQ \subseteq N$ for three sets in $\mathcal{C}$, then we can add a rule $N \rightarrow PQ$; see for example [10]. Similarly, if $w \in N$ we can add a rule $N \rightarrow w$; in particular we will need rules of the form $N \rightarrow a$ and $N \rightarrow \lambda$ where $a \in \Sigma$. Thus we can add all rules of the first type, and only a finite number of the second type to get a context free grammar, and this context free grammar will clearly have the property that $N \stackrel{*}{\Rightarrow} w$ will imply that $w \in N$. Obviously this leaves many questions unanswered: how to define these sets, and how to compute whether one is a subset of the other.

In [1], the class $\mathcal{C}$ consists of a finite set of congruence classes; these are non-overlapping sets, which form a congruence: that is to say for any two strings u,v, we have that $[u][v] \subseteq [uv]$. This makes the inference process quite straightforward. Here we will look at using a much larger class $\mathcal{C}$ which includes many overlapping sets and has the structure of a lattice. In particular the primitive el-

ements are defined dually: in terms of finite intersections of sets of strings which are defined by contexts, not strings. Given a context, or pair of strings, (l, r), the elementary sets we consider are of the form:

$$\{w | lwr \in L\}$$

Whereas the congruence classes are the smallest and most fine-grained sets that can be distributionally defined, since they are the sets of strings that have identical distributions, the elementary classes that we define here are in some sense the largest possible classes that we can define. These are sets of strings that have only one context in their common distribution, as opposed to sets that have all their contexts in their common distribution. The languages that can be defined using these classes are thus rather different in character from those that are defined using the congruence based approaches.

2 Distributional Lattice

The theoretical base of our approach is the syntactic concept lattice [8], which is a rich algebraic object which can be thought of as a lattice-theoretic generalisation of the syntactic monoid.

2.1 Notation

Given a finite non-empty alphabet Σ, we use Σ^* to refer to the set of all strings and λ to refer to the empty string. As usual a language L is any subset of Σ^*.

A context is just an ordered pair of strings that we write (l, r) – l and r refer to left and right. We can combine a context (l, r) with a string u with a wrapping operation that we write $\odot$: so $(l, r) \odot u$ is defined to be lur. We will sometimes write f for a context (l, r). We will extend this notation to sets of contexts and strings in the natural way.

Given a formal language L and a given string w we can define the *distribution* of that string to be the set of all contexts that it can appear in: $C_L(w) = \{(l, r) | lwr \in L\}$, equivalently $\{f | f \odot w \in L\}$. There is a special context (λ, λ): clearly $(\lambda, \lambda) \in C_L(w)$ iff $w \in L$.

There is a natural equivalence relation on strings defined by equality of distribution: $u \equiv_L v$ iff $C_L(u) = C_L(v)$; this is called the syntactic congruence. We write $[u]$ for the congruence class of u.

2.2 Lattice

Distributional learning is learning that exploits or models the distribution of strings in the language. A sophisticated approach can be based on the Galois connection between sets of strings and sets of contexts.

For a given language L we can define two polar maps from sets of strings to sets of contexts and vice versa. Given a set of strings S we can define a set of contexts S' to be the set of contexts that appear with every element of S.

$$S' = \{(l, r) \in \Sigma^* \times \Sigma^* : \forall w \in S \;\; lwr \in L\} \tag{1}$$

Dually we can define for a set of contexts C the set of strings C' that occur with all of the elements of C

$$C' = \{w \in \Sigma^* : \forall (l, r) \in C \;\; lwr \in L\} \tag{2}$$

We define a syntactic concept to be an ordered pair of a set of strings S and a set of contexts C, written $\langle S, C\rangle$, such that $S' = C$ and $C' = S$. An alternative way of looking at this is that the concepts are the pairs $\langle S, C\rangle$ such that $C \odot S \subseteq L$ and such that these are maximal. For any set of strings S we can define a concept $\mathcal{C}(S) = \langle S'', S'\rangle$. S'' is called the closure of the set S; this is a closure operator as $S''' = S'$, for any set of strings. We can also form a concept from a set of contexts C as $\mathcal{C}(C) = \langle C', C''\rangle$.

Importantly, there will be a finite number of concepts in the lattice, if and only if the language is regular. Each concept represents a natural set of strings in the language; these form an overlapping hierarchy of all sets of strings that can be distributionally defined.

We can define a partial order on these concepts where:

$$\langle S_1, C_1\rangle \leq \langle S_2, C_2\rangle \text{ iff } S_1 \subseteq S_2.$$

Note that $S_1 \subseteq S_2$ iff $C_1 \supseteq C_2$.

We can see that $\mathcal{C}(L) = \mathcal{C}(\{(\lambda, \lambda)\})$, and clearly $w \in L$ iff $\mathcal{C}(\{w\}) \leq \mathcal{C}(\{(\lambda, \lambda)\})$. We will drop brackets from time to time to improve legibility. This poset in fact forms a complete lattice with a top element $\top$ which will normally be $\langle \Sigma^*, \emptyset\rangle$, though there may be some contexts shared by every string, for example in the language Σ^*, and similarly a bottom element $\bot$, $\langle \emptyset, \Sigma^* \times \Sigma^*\rangle$, though again there may be some strings in the bottom element. Indeed if $L = \Sigma^*$ then the lattice has only one element and $\top = \bot = \mathcal{C}(L) = \langle \Sigma^*, \Sigma^* \times \Sigma^*\rangle$. If $L = \Sigma^* a \Sigma^*$, then the lattice has two elements: the top element is $\langle \Sigma^*, L \times \Sigma^* \cup \Sigma^* \times L\rangle$, and the bottom element is $\langle L, \Sigma^* \times \Sigma^*\rangle$.

The relation to the synactic monoid is crucial; $\mathcal{C}(\{u\}) = \mathcal{C}([u])$ but there may be other strings in the concept that are not congruent to u. In particularly the set of strings in the concept of u will be the union of all of the congruence classes whose distribution contains the distribution of u. i.e. if $C_L(u) \subseteq C_L(v)$ then v and indeed $[v]$ will be in the concept $\mathcal{C}(u)$. More formally $\mathcal{C}(\{u\})$ will have the set of strings $\{v | C_L(v) \supseteq C_L(u)\}$ whereas $[u] = \{v | C_L(v) = C_L(u)\}$.

We define a concatenation operation on these as follows; somewhat similar to the concatenation in the syntactic monoid.

Definition 1. $\langle S_x, C_x\rangle \circ \langle S_y, C_y\rangle = \langle (S_x S_y)'', (S_x S_y)'\rangle$

We refer the reader to [8] for a more detailed derivation of the properties of this lattice; and a proof that it is a residuated lattice.

3 Grammar

We will use CFGs, which we define standardly as a tuple $\langle \Sigma, V, P, S \rangle$; where Σ is a non-empty finite set, the alphabet; V is a finite set of non-terminals disjoint from Σ, S is a distinguished element of V, the start symbol, and P is a finite set of productions of the form $V \times (\Sigma \cup V)^*$; we will write these as $V \to \alpha$. We will consider CFGs in Chomsky Normal Form, where all of the productions are either of the form $N \to a$, $N \to \lambda$ or $N \to RS$. We write the standard derivation as $\beta N \gamma \Rightarrow_G \beta \alpha \gamma$, when $N \to \alpha \in P$ and the transitive reflexive closure as $\stackrel{*}{\Rightarrow}_G$.

Given a lattice $\mathfrak{B}(L)$ and a finite set of concepts $V \subseteq \mathfrak{B}(L)$, which includes the concept $\mathcal{C}(L)$, we can define a CFG as follows. The set of non-terminals will be equal to this set of concepts; either a set of symbols that are in bijection with the concepts or alternatively the concepts themselves. The start symbol will be the concept $\mathcal{C}(L)$. We define the set of productions P as follows: If $N = \langle S, C \rangle$ and $a \in \Sigma \cup \{\lambda\}$ and $a \in S$, then we add a rule $N \to a$. If $A \circ B \leq N$ then we add a rule $N \to AB$. We will call this grammar $G(L, V)$.

Lemma 1. *For all L and for all $V \subseteq \mathfrak{B}(L)$, if $N = \langle S_N, C_N \rangle$ and $N \stackrel{*}{\Rightarrow}_G w$ then $w \in S_N$.*

Proof. By induction on the length of the derivation. Suppose $N \to w$ is the complete derivation; then by construction $w \in S_N$. Suppose it is true for all derivations of length at most k, and let $N \stackrel{*}{\Rightarrow}_G w$ be a derivation of length $k+1$. Suppose the first step of the derivation is $N \to PQ \stackrel{*}{\Rightarrow}_G uv = w$, and $P \stackrel{*}{\Rightarrow}_G u$ and $Q \stackrel{*}{\Rightarrow}_G v$; by the inductive hypothesis $u \in S_P$ and $v \in S_Q$; (using the obvious notation for the sets of strings in the concepts P and Q). By the definition of $P \circ Q$ we have that $uv \in S_{P \circ Q}$, and since $P \circ Q$ is less than N we have that $uv \in S_N$.

Of course this result assumes that we can correctly identify both the concepts and the concatenation operation; we now address this point.

3.1 Partial lattice

Given a finite set of strings K, and a finite set of contexts F we can produce a partial lattice. We consider a set D which is a sufficiently large subset of L; in particular we take $D = L \cap (F \odot KK)$. We then define the lattice $\mathfrak{B}(K, D, F)$ to be the set of all ordered pairs $\langle S, C \rangle$ where $S \subseteq K$ and $C \subseteq F$ and $C = S' \cap F$, and $S = C' \cap K$. We can compute this lattice using only the finite sets K, D and F. We use two subroutines defined as **GetK** and **GetF**; if C is a set of contexts, **GetK**(C) will return C', and **GetF**(S) will return S'.

We therefore define the following algorithm which uses a membership oracle, and takes as input a finite set of strings K and a finite set of contexts F. We assume that $(\lambda, \lambda) \in F$. Algorithm 1 will generate a CFG in Chomsky normal form given the finite sets K, D and F, and an integer bound f which is at least 1. We discuss f further below.

The important point of this algorithm is the schema we use for generating the branching productions of the grammar. If we have three non-terminals N, P, Q which correspond to $\langle S_N, C_N \rangle$, $\langle S_P, C_P \rangle$, $\langle S_Q, C_Q \rangle$, then we have rules of the form $N \to PQ$ if and only if $((S_P S_Q)' \cap F)' \cap K \subseteq S_N$.

Let us consider this condition for a moment. $S_P S_Q$ is a subset of KK; this may not contain any elements of K at all; but we take the set of all strings of K that have all of the contexts that are shared by all of the elements of $S_P S_Q$, and compare this to S_N.

It could be that $((S_P S_Q)' \cap F)' \cap K$ is empty; then for every non-terminal N we will have a rule $N \to PQ$; or it could be that $N = \top$ in which case for every non-terminals P, Q, we will have $N \to PQ$. We can remove these excessive productions and redundant non-terminals using standard techniques. An alternative condition would be $((S_P S_Q)' \cap F) \supseteq C_N$; but this does not have quite the right properties.

Example 1. Consider the Dyck language over a, b, that consists of strings like $\{\lambda, ab, abab, aabb, \dots\}$. Let $K = \{\lambda, a, b, ab\}$ and $F = \{(\lambda, \lambda), (a, \lambda), (b, \lambda)\}$. $D = L \cap (F \odot KK) = \{\lambda, ab, aabb, abab\}$. There are therefore 5 concepts in the lattice $\mathfrak{B}(K, D, F)$

- The top element $T = \langle K, \emptyset \rangle$
- The bottom element $N = \langle \emptyset, F \rangle$
- $S = \langle \{\lambda, ab\}, \{(\lambda, \lambda)\} \rangle$
- $A = \langle \{a\}, \{(\lambda, b)\} \rangle$
- $B = \langle \{b\}, \{(a, \lambda)\} \rangle$

The grammar therefore has 5 non-terminals, labelled S, A, B, T and N. We will have lexical rules; $T \to a, A \to a$ and $T \to b$ and $B \to b$. We will have λ-rules $S \to \lambda$ and $T \to \lambda$. Since there are 5 concepts, there are 125 possible branching rules. We have a large number of vacuous rules with N on the right hand side —there are 45 such rules $N \to NN, A \to BN$ and so on. We then have a large number of vacuous rules with T on the left — there are 25 of these: $T \to AA, T \to AN$ etc. 16 of these are not duplicated with the 45 previous rules.

Stripping out all of these we are left with the following rules $S \to SS, S \to AB$, $A \to AS, A \to SA$, $B \to BS, B \to SB$ and $S \to \lambda$, $A \to a$, $B \to b$. This grammar clearly generates the Dyck language.

For a fixed value of f, Algorithm 1 will produce a polynomial sized grammar and will always run in polynomial time. Note that if we did not bound the set of concepts in some way, and used the whole lattice $\mathfrak{B}(K, D, F)$ we might have exponentially large grammars.

Example 2. Suppose $\Sigma = \{a_1, \dots a_n\}$ and $L = \{xy | x, y \in \Sigma, x \neq y\}$. If $K = \Sigma$ and $F = \{(\lambda, x) | x \in \Sigma\}$ then $\mathfrak{B}(K, L, F)$ will have 2^n elements;

Algorithm 1: MakeGrammar

Data: A finite set of strings K, a finite set of contexts F, a finite set of strings D, a non-empty finite set Σ, a bound f
Result: A context-free grammar G
$S = \mathcal{C}(\{(\lambda, \lambda)\})$;
for $A \subseteq F$, $|A| \leq f$ **do**
  $V \leftarrow V \cup \{\mathcal{C}(A)\}$;
end
$P \leftarrow \emptyset$;
for *each* $a \in \Sigma \cup \{\lambda\}$ **do**
  for *each* $N \in V$, $N = \langle S_N, C_N \rangle$ **do**
    if $a \in S_N$ **then**
      $P \leftarrow P \cup \{N \rightarrow a\}$;
    end
  end
end
for *each* $A, B \in V$ **do**
  $J = S_A S_B$;
  $S_X \leftarrow$ `GetK(GetF(`J`))` ;
  for *each* $N \in V$ **do**
    if $S_X \subseteq S_N$ **then**
      $P \leftarrow P \cup \{N \rightarrow AB\}$;
    end
  end
end
return $G = \langle \Sigma, V, P, S \rangle$;

4 Inference

Given a suitable source of information we can then consider the search process for a suitable set of strings K and set of contexts F. We will now establish two monotonicity lemmas just as in [8]. We first change our notation slightly. We assume that L is fixed and that $D = L \cap (F \odot KK)$. We therefore will write $\mathfrak{B}(K, L, F)$ as a shorthand for $\mathfrak{B}(K, L \cap (F \odot KK), F)$, and similarly $G(K, L, F)$ for the grammar generated from this. In what follows we will consider the bounds k and f to be fixed.

The first lemma states that if we increase the set of contexts we use, then the language will increase. We take a language L, and two sets of contexts $F_1 \subseteq F_2$ and a set of strings K. Let G_1 be the grammar formed from K, L, F_1 and $G_2 = G(K, L, F_2)$. For a concept $\langle S, C \rangle$ in $\mathfrak{B}(K, L, F_1)$ note that there is a corresponding concept $\langle S, S' \cap F_2 \rangle$ in $\mathfrak{B}(K, L, F_2)$. We will write $f^* : \mathfrak{B}(K, L, F_1) \rightarrow \mathfrak{B}(K, L, F_2)$ for the function $f^*(\langle S, C \rangle) = \langle S, S' \cap F_2 \rangle$.

Lemma 2. *If $N \rightarrow PQ$ is a production in G_1 then $f^*(N) \rightarrow f^*(P) f^*(Q)$ is a production in G_2.*

Proof. First of all, since $N \to PQ$ is a production in G_1 it follows that $((S_P S_Q)' \cap F)' \cap K \subseteq S_N$. Clearly, since $G \supseteq F$, $((S_P S_Q)' \cap G) \supseteq ((S_P S_Q)' \cap F)$ and so $((S_P S_Q)' \cap G)' \subseteq ((S_P S_Q)' \cap F)'$, therefore $((S_P S_Q)' \cap G)' \cap K \subseteq S_N$; which means that there is a rule $f^*(N) \to f^*(P) f^*(Q)$ in G_2. Note that if N, P, Q lie in the bounded subset of concepts defined by f, $f^*(N), f^*(P)$ and $f^*(Q)$ will also lie in the bounded set in the larger lattice.

Lemma 3. *If* $\langle S, C \rangle \stackrel{*}{\Rightarrow}_{G_1} w$ *then* $\langle S, S' \rangle \stackrel{*}{\Rightarrow}_{G_2} w$.

Proof. Suppose $\langle S, C \rangle = N \stackrel{*}{\Rightarrow}_{G_1} w$. We proceed by induction on length of the derivation. It is obviously true for a derivation of length 1 by construction. Suppose true for all derivations of length at least k; We must have $N \to PQ \stackrel{*}{\Rightarrow}_{G_1} uv = w$; where $P \stackrel{*}{\Rightarrow}_{G_1} u$ and $Q \stackrel{*}{\Rightarrow}_{G_1} v$. By the inductive hypothesis we have that $f^*(P) \stackrel{*}{\Rightarrow}_{G_2} u$ and $f^*(Q) \stackrel{*}{\Rightarrow}_{G_2} v$. Since $f^*(N) \to f^*(P) f^*(Q)$ will be a production in G_2, by the previous lemma, the result holds by induction.

Clearly this means that as we increase the set of contexts the language defined can only increase. Conversely we have that if we increase the set of strings in the kernel, the language will decrease. Suppose $J \subseteq K$; and define the map between $\mathfrak{B}(K, L, F)$ (which defines a grammar G_1) and $\mathfrak{B}(J, L, F)$ which defines the grammar G_2 as $g(\langle S, C \rangle) = \langle S \cap J, (S \cap J)' \rangle$.

Lemma 4. *If* $\langle S, C \rangle \stackrel{*}{\Rightarrow}_{G_1} w$ *then* $g(\langle S, C \rangle) \stackrel{*}{\Rightarrow}_{G_2} w$

Proof. If we have a rule $N \to PQ$ in G_1; then this means that $((S_P S_Q)' \cap F)' \cap K \subseteq S_N$. Now $(S_P \cap J)(S_Q \cap J) \subseteq (S_P S_Q)$ And so $(((S_P \cap J)(S_Q \cap J)' \cap F)' \cap K \subseteq S_N$ And so $(((S_P \cap J)(S_Q \cap J)' \cap F)' \cap J \subseteq S_N \cap J$ which means there is a rule in G_2 $g(N) \to g(P) g(Q)$, and the result follows by induction as before.

Lemma 5. *For any language L, and set of contexts F, there is a set of strings K, such that $L(G(K, L, F)) \subseteq L$*

Proof. As we increase K the number of concepts in $\mathfrak{B}(K, L, F)$ may increase, but is obviously bounded by $2^{|F|}$. We start by assuming that K is large enough that we have a maximal number of concepts. Given two concepts X, Y, define $D = (C'_x C'_y)' \cap F$. This is the set of contexts shared in the infinite data limit. For each element $(l, r) \in F \setminus D$ we take a pair of strings $u \in C'_X$ and $v \in C'_Y$ such $luvr \notin L$; if have all such pairs in K, then we can easily see that $\langle S, C \rangle \stackrel{*}{\Rightarrow} w$ implies that $w \in C'$.

This means that for any set of contexts, as we increase K the language will decrease until finally it will be a subset of the target language.

For a given L and F define the limit language as $\bigcap_{K \subset \Sigma^*} L(G(K, L, F))$, where the intersection is limited to all finite K. This limit will be attained for some finite K.

Definition 2. *Given a language L, a finite set of contexts F is* adequate*, iff for every finite set of strings K that includes $\Sigma \cup \{\lambda\}$, $L(G(K, L, F)) \supseteq L$.*

Clearly by the previous definitions, any superset of an adequate set of contexts is also adequate. If a language has an adequate finite set of contexts F, then for sufficiently large K, the grammar will define the right language.

We say that a CFG in CNF has the finite context property if every non-terminal can be defined by a finite set of contexts. Defining $L(G, N) = \{w | N \stackrel{*}{\Rightarrow}_G w\}$, this property requires that for all non-terminals N, there is a finite set of contexts F_N such that $L(G, N) = F'_N$.

For a given CFG with the FCP we can define $f(G)$ to be the maximum cardinality of F_N over the non-terminals in G, and we will define $f(L)$ to be the minimum of $k(G)$ over all grammars G such that $L(G) = L$. We will now show that the algorithm will learn all context free languages with a bound on $f(L)$.

5 Learning Model

Before we present an algorithm we will describe the learning model that we will use. We use the same approach as in [7] and other papers. We assume that we have a sequence of positive examples, and that we can query examples for membership. See [11] for arguments that this is a plausible model.

In other words, we have two oracles, one which will generate positive examples from the language, and the other which will allow us to test whether a given string in the language. After every time the algorithm receives a positive example, the learner must use a polynomial amount of computation, and produce a hypothesis.

Given a language L a presentation for L is an infinite sequence of strings $w_1, w_2, \ldots$ such that $\{w_i | i > 0\} = L$. An algorithm receives a sequence T and an oracle, and must produce a hypothesis H at every step, using only a polynomial number of queries to the membership oracle. It identifies in the limit the language L iff for every presentation T of L there is a N such that for all $n > N$ $H_n = H_N$, and $L(H_N) = L$. We say it identifies in the limit a class of languages $\mathcal{L}$ iff it identifies in the limit all L in $\mathcal{L}$. We say that it identifies the class in polynomial update time iff there is a polynomial p, such that at each step the model uses an amount of computation (and thus also a number of queries) that is less than $p(n, l)$, where n is the number of strings and l is the maximum length of a string in the observed data. We note that this is slightly too weak. It is possible to produce vacuous enumerative algorithms that can learn anything by only processing a logarithmically small prefix of the string [12].

6 Algorithm

Algorithm 2 is the learning algorithm we use. We will present the basic ideas informally before proving its correctness.

We initialise K and F to the most basic sets we can: so F will just consist of the empty context (λ, λ) and K will be λ. We generate a grammar, and then we repeat the following process. We draw a positive example and if the positive

example is not in our current hypothesis, then we add additional contexts to F. We want to keep F to be very limited and only increase it when we are forced to.

We maintain a large set of all of the substrings that we have seen so far; this is stored in the variable K_2; we compare the grammar formed with K_2 to the one formed with K. If they are different then that means that K_2 is more accurate, in that it will eliminate some incorrect rules, and that we have not yet attained the limit language, and as a result we might overgeneralise, and so we will increase K to K_2. In general we will add to K as much as we want; it can only make the grammar more accurate.

We just need to check whether two grammars with the same set of contexts are identical. This merely requires us to verify that there are the same number of concepts; and that for every concept in one there is a concept in the other with the same set of contexts, and that for every triple of concepts $X \geq Y \circ Z$ the same inequality holds between the corresponding elements in the larger lattice.

Algorithm 2: CFG learning algorithm

Data: Input alphabet Σ, bounds k, f
Result: A sequence of CFGs $G_1, G_2, \ldots$
$K \leftarrow \Sigma \cup \{\lambda\}$, $K_2 = K$;
$F \leftarrow \{(\lambda, \lambda)\}$, $E = \{\}$;
$D = (F \odot KK) \cap L$;
$G =$ `Make(` K, D, F,f`)` ;
repeat
  $w =$ `GetPositiveExample`; **if** *there is some* $w \in E$ *that is not in* $L(G)$ **then**
    $F \leftarrow Con(E)$;
    $K \leftarrow K_2$;
    $D = (F \odot KK) \cap L$;
    $G =$ `Make(`K, D, F,f`)` ;
  end
  else
    $D_2 \leftarrow (F \odot K_2K_2) \cap L$;
    if $\langle K_2, D_2, F\rangle$ *not isomorphic to* $\langle K, D, F\rangle$ **then**
      $K \leftarrow K_2$;
      $D = (F \odot KK) \cap L$;
      $G =$ `Make(`K,D,F,f`)`;
    end
  end
  Output G;
until ;

6.1 Proof

We will now show that this algorithm is correct for a certain class of languages. We have a parameter f which we assume is fixed; for each value we have a

learnable class. For a fixed value of f we can learn all context-free languages L such that $f(L) \leq f$. That is to say, all context-free languages that can be defined by a CFG where each non-terminal can be contextually defined by at most f contexts. As is now standard in context-free grammatical inference we cannot define a decidable syntactic property, but rely on defining a class of languages by reference to the algorithm. We will later try to clarify the class of languages that lie in each class.

Theorem 1. *Algorithm 2 identifies in the limit, from positive data and membership queries the class* $\mathcal{L}_{\text{FCP}}(f)$.

This theorem is an immediate consequence of the following two lemmas.

Lemma 6. *There is a point* N *at which* F_N *is adequate for* L*; and for all* $n > N$, $F_n = F_N$.

Proof. First, once F is adequate the language defined will always include the target and thus F will never be increased again. Suppose $L \in \mathcal{L}_{\text{FCP}}(f)$ and let F be an adequate set of contexts. Let n be the first time that $Con(E_n)$ contains F. Let F_n be the set of contexts at that point. If F_n is adequate we are done; assume it is not. Therefore there is some set of strings K such that $L(G(K, D, F_N))$ is not a subset of L; i.e. there is some $w \in L \setminus L(G(K, D, F_N))$ Let n_2 be the first time that K_2 contains K and E contains w; at this point, or some earlier point, we will increase F and it will be adequate.

Lemma 7. *If* F_n *is adequate then there is some* $n_2 > n$ *at which point* $L(G(K_n, L, F_n)) = L$.

Proof. Let K_0 be a set of strings such that $G(K, L, F_n)$ defines exactly the right language. Furthermore $K_0 \subseteq Sub(L)$. Let T be a set of strings of L such that $K_0 \subset Sub(T)$ and let m be a point such that $T \subseteq E_m$ and $m > n$. Either $G(K_m, L, F_n)$ is correct, in which case we are done, or it is not, in which case it will differ from $G(Sub(E), L, F_n)$ and thus K will be increased to include K_0, which means that it will correct at this point.

We will now give some simple examples of languages for varying values of f. We define $\mathcal{L}_{\text{FCP}}(f)$ to be the set of languages learnable for a value of f. The simplest class is the class $\mathcal{L}_{\text{FCP}}(1)$. First, we note that the regular languages lie within this class. Given a regular language L consider a deterministic regular grammar for L, with no unreachable non-terminals. For a given non-terminal N in this grammar let w_N be a string such that $S \stackrel{*}{\Rightarrow} wN$. Since the grammar is deterministic, we know that if $S \stackrel{*}{\Rightarrow} wM$ then $N = M$. The context (w, λ) therefore contextually defines the non-terminal N.

Consider the language $L_{nd} = \{a^n b^n c^m | n, m \geq 0\} \cup \{a^m b^n c^n | n, m \geq 0\}$. L_{nd} is a classic example of an inherently ambiguous and thus non-deterministic language; moreover it is a union of infinitely many congruence classes. This language also lies within the class $\mathcal{L}_{\text{FCP}}(1)$. In Table 1, we give on the left the sets of strings generated by a natural CFG for this language, and on the right a

context that defines that set of strings. Thus even the very simplest element of this class contains inherently ambiguous languages.

If we consider the class $\mathcal{L}_{\text{FCP}}(2)$, then we note that the palindrome languages and the language L_2 defined in the introduction lie in this class. The palindrome languages require two contexts to define the elementary letters of the alphabet – so for example $\{a\}$ can be defined by the two contexts (λ, ba) and (λ, aa).

$L(G,N)$	F_N
λ	$(aaabb, bccc)$
a	$(\lambda, abbccc)$
b	$(aaab, bccc)$
c	$(aaabbc, \lambda)$
c^*	(c, λ)
a^*	(λ, a)
$\{a^n b^n \vert n \geq 0\}$	(a, b)
$\{a^n b^{n+1} \vert n \geq 0\}$	(aa, b)
$\{b^n c^n \vert n \geq 0\}$	(b, c)
$\{b^n c^{n+1} \vert n \geq 0\}$	(bb, c)
L_{nd}	(λ, λ)

Table 1. Contextually defined grammar for the language $L_{nd} = \{a^n b^n c^m \vert n, m \geq 0\} \cup \{a^m b^n c^n \vert n, m \geq 0\}$

7 Discussion

There is an important point that we will discuss at length: the switch from context free representations to context sensitive representations. Backing off to CFGs makes it clear how important the use of Distributional Lattice Grammars (DLGs) are. With a DLG, we can compactly represent the potentially exponentially large set of concepts and perform the parsing directly. The difference is this: In the CFG we have to treat each derivation separately. If we have one concept that contains the context f and another that contains the context g and both of these can generate the same string w; this does not mean that there is a concept containing f and g that will generate that string, since the two contexts could come from different sub-derivations. In a DLG however, we can aggregate information from different derivations – far from making things more difficult, this actually makes the derivation process simpler since we only need to keep one concept for each span in the parse table. Thus in this case using a context-sensitive representation is a solution to a problem, rather than creating new problems itself.

Suppose we are given the sets of strings generated by each non-terminal of a CFG in CNF. Then it is easy to write down the rules: for any triple of sets/non-terminals X, Y, Z we have a rule $X \rightarrow YZ$ iff $X \supseteq YZ$. We also have rules

of the form $X \to a$ iff $a \in X$. Thus the general strategy for CF inference that we propose is to define a suitable collection of sets of strings, and then to construct all valid rules. The congruence based approach uses the congruence classes: since $[u][v] \subseteq [uv]$ we have the rule schema $[uv] \to [u][v]$. Here we have the same approach with concepts: $S_x S_y \subseteq (S_x S_y)''$, and so we have a rule $\mathcal{C}(X) \circ \mathcal{C}(Y) \to \mathcal{C}(X), \mathcal{C}(Y)$.

The current approach is very heavily influenced by the class of Binary Feature Grammars BFGs, but they are also very different. In particular, this model is in some sense a dual representation. In the BFG formalism the primitive elements are determined by strings, and the contexts are used to restrict the generated rules. Here, dually, the primitive elements are defined by the contexts and the strings are used to restrict the generated rules. As a result the the monotonicity lemmas for BFGs go in the opposite direction: for a BFG, if we increase K we increase the language, and if we increase F we decrease the language. In the current construction, the opposite is true.

In general there are two ways of proceeding – we define the sets of non-terminals in terms of strings; the set of all strings congruent to a given string, or alternatively in terms of contexts: the set of all strings that can occur in a given set of contexts. These two approaches give rise to two different sets of algorithms. Clearly this does not exhaust the approach, as we could combine the two. For example, if we define $[l, r]$ to be the set of all strings that have the context (l, r), then the rule schemas $[l, r] \to [l, xr][x]$ and $[l, r] \to [y][ly, r]$ are also valid.

Since the concepts form a lattice, we suggest as a direction for future research that a variety of heuristic algorithms could be used, since a lattice is a good search space, as is known for regular grammatical inference [13], but we do not consider such heuristic algorithms here.

8 Conclusion

We have presented an approach to context-free grammatical inference where the non-terminals correspond to elements of a distributional lattice. Using this representational assumption we have presented an efficient algorithm for the inference of a very large set of context free languages, subject to the setting of certain bounds on the number of non-terminals defined.

Acknowledgments

I am very grateful to the reviewers for identifying several confusing points in this paper; the paper has been much improved as a result of their comments.

References

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

2. Clark, A.: PAC-learning unambiguous NTS languages. In: Proceedings of the 8th International Colloquium on Grammatical Inference (ICGI). (2006) 59–71
3. Yoshinaka, R.: Identification in the Limit of k-l-Substitutable Context-Free Languages. In: Proceedings of the International Colloquium on Grammatical Inference, Springer (2008) 266–279
4. Yoshinaka, R.: Learning mildly context-sensitive languages with multidimensional substitutability from positive data. In Gavaldà, R., Lugosi, G., Zeugmann, T., Zilles, S., eds.: ALT. Volume 5809 of Lecture Notes in Computer Science., Springer (2009) 278–292
5. Clark, A.: Distributional learning of some context-free languages with a minimally adequate teacher. In: Proceedings of the ICGI, Valencia, Spain (September 2010)
6. Asveld, P., Nijholt, A.: The inclusion problem for some subclasses of context-free languages. Theoretical computer science **230**(1-2) (2000) 247–256
7. Clark, A., Eyraud, R., Habrard, A.: A polynomial algorithm for the inference of context free languages. In: Proceedings of the International Colloquium on Grammatical Inference, Springer (September 2008) 29–42
8. Clark, A.: A learnable representation for syntax using residuated lattices. In: Proceedings of the 14th Conference on Formal Grammar, Bordeaux, France (2009)
9. Chi, Z., Geman, S.: Estimation of probabilistic context-free grammars. Computational Linguistics **24**(2) (1998) 299–305
10. Martinek, P.: On a Construction of Context-free Grammars. Fundamenta Informaticae **44**(3) (2000) 245–264
11. Clark, A., Lappin, S.: Another look at indirect negative evidence. In: Proceedings of the EACL Workshop on Cognitive Aspects of Computational Language Acquisition, Athens (March 2009)
12. Pitt, L.: Inductive inference, DFAs, and computational complexity. In Jantke, K.P., ed.: Analogical and Inductive Inference. Number 397 in LNAI. Springer-Verglag (1989) 18–44
13. Dupont, P., Miclet, L., Vidal, E.: What is the search space of the regular inference? In: ICGI: Grammatical Inference and Applications, Springer (1994) 25–37