

Efficient, correct, unsupervised learning of context-sensitive languages

Alexander Clark

Department of Computer Science
Royal Holloway, University of London
alexcl@cs.rhul.ac.uk

Abstract

A central problem for NLP is grammar induction: the development of unsupervised learning algorithms for syntax. In this paper we present a lattice-theoretic representation for natural language syntax, called Distributional Lattice Grammars. These representations are objective or empiricist, based on a generalisation of distributional learning, and are capable of representing all regular languages, some but not all context-free languages and some non-context-free languages. We present a simple algorithm for learning these grammars together with a complete self-contained proof of the correctness and efficiency of the algorithm.

1 Introduction

Grammar induction, or unsupervised learning of syntax, no longer requires extensive justification and motivation. Both from engineering and cognitive/linguistic angles, it is a central challenge for computational linguistics. However good algorithms for this task are thin on the ground. There are numerous heuristic algorithms, some of which have had significant success in inducing constituent structure (Klein and Manning, 2004). There are algorithms with theoretical guarantees as to their correctness – such as for example Bayesian algorithms for inducing PCFGs (Johnson, 2008), but such algorithms are inefficient: an exponential search algorithm is hidden in the convergence of the MCMC samplers. The efficient algorithms that are actually used are heuristic approximations to the true posteriors. There are algorithms like the Inside-Outside algorithm (Lari and Young, 1990) which are guaranteed to converge efficiently, but not necessarily to the right answer: they converge to a local optimum that

may be, and in practice nearly always is very far from the optimum. There are naive enumerative algorithms that are correct, but involve exhaustively enumerating all representations below a certain size (Horning, 1969). There are no correct *and* efficient algorithms, as there are for parsing, for example.

There is a reason for this: from a formal point of view, the problem is intractably hard for the standard representations in the Chomsky hierarchy. Abe and Warmuth (1992) showed that training stochastic regular grammars is hard; Angluin and Kharitonov (1995) showed that regular grammars cannot be learned even using queries; these results obviously apply also to PCFGs and CFGs as well as to the more complex representations built by extending CFGs, such as TAGs and so on. However, these results do not necessarily apply to other representations. Regular grammars are not learnable, but deterministic finite automata are learnable under various paradigms (Angluin, 1987). Thus it is possible to learn by changing to representations that have better properties: in particular DFAs are learnable because they are “objective”; there is a correspondence between the structure of the language, (the residual languages) and the representational primitives of the formalism (the states) which is expressed by the Myhill-Nerode theorem.

In this paper we study the learnability of a class of representations that we call distributional lattice grammars (DLGs). Lattice-based formalisms were introduced by Clark et al. (2008) and Clark (2009) as context sensitive formalisms that are potentially learnable. Clark et al. (2008) established a similar learnability result for a limited class of context free languages. In Clark (2009), the approach was extended to a significantly larger class but without an explicit learning algorithm. Most of the building blocks are however in place, though we need to make several modifications and ex-

tensions to get a clean result. Most importantly, we need to replace the representation used there, which naively could be exponential, with a lazy, exemplar based model.

In this paper we present a simple algorithm for the inference of these representations and prove its correctness under the following learning paradigm: we assume that as normal there is a supply of positive examples, and additionally that the learner can query whether a string is in the language or not (an oracle for membership queries). We also prove that the algorithm is efficient in the sense that it will use a polynomial amount of computation and makes a polynomial number of queries at each step.

The contributions of this paper are as follows: after some basic discussion of distributional learning in Section 2, we define in Section 3 an exemplar-based grammatical formalism which we call Distributional Lattice Grammars. We then give a learning algorithm under a reasonable learning paradigm, together with a self contained proof in elementary terms (not presupposing any extensive knowledge of lattice theory), of the correctness of this algorithm.

2 Basic definitions

We now define our notation; we have a finite alphabet Σ ; let Σ^* be the set of all strings (the free monoid) over Σ , with λ the empty string. A (formal) language is a subset of Σ^* . We can concatenate two languages A and B to get $AB = \{uv | u \in A, v \in B\}$.

A context or *environment*, as it is called in structuralist linguistics, is just an ordered pair of strings that we write (l, r) where l and r refer to left and right; l and r can be of any length. 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) . There is a special context (λ, λ) : $(\lambda, \lambda) \odot w = w$. We will extend this to sets of contexts and sets of strings in the natural way. We will write $Sub(w) = \{u | \exists (l, r) : lur = w\}$ for the set of substrings of a string, and $Con(w) = \{(l, r) | \exists u \in \Sigma^* : lur = w\}$.

For a given string w we can define the *distribution* of that string to be the set of all contexts that it can appear in: $C_L(w) = \{(l, r) | lur \in L\}$, equivalently $\{f | f \odot w \in L\}$. Clearly $(\lambda, \lambda) \in C_L(w)$ iff $w \in L$.

Distributional learning (Harris, 1954) as a technical term refers to learning techniques which model directly or indirectly properties of the distribution of strings or words in a corpus or a language. There are a number of reasons to take distributional learning seriously: first, historically, CFGs and other PSG formalisms were intended to be learnable by distributional means. Chomsky (2006) says (p. 172, footnote 15):

The concept of “phrase structure grammar” was explicitly designed to express the richest system that could reasonably be expected to result from the application of Harris-type procedures to a corpus.

Second, empirically we know they work well at least for lexical induction, (Schütze, 1993; Curran, 2003) and are a component of some implemented unsupervised learning systems (Klein and Manning, 2001). Linguists use them as one of the key tests for constituent structure (Carnie, 2008), and finally there is some psycholinguistic evidence that children are sensitive to distributional structure, at least in artificial grammar learning tasks (Saffran et al., 1996). These arguments together suggest that distributional learning has a somewhat privileged status.

3 Lattice grammars

Clark (2009) presents the theory of lattice based formalisms starting algebraically from the theory of residuated lattices. Here we will largely ignore this, and start from a straightforward computational treatment. We start by defining the representation.

Definition 1. *Given a non-empty finite alphabet, Σ , a distributional lattice grammar (DLG) is a 3-tuple consisting of $\langle K, D, F \rangle$, where F is a finite subset of $\Sigma^* \times \Sigma^*$, such that $(\lambda, \lambda) \in F$, K is a finite subset of Σ^* which contains λ and Σ , and D is a subset of $(F \odot KK)$.*

K here can be thought of as a finite set of exemplars, which correspond to substrings or fragments of the language. F is a set of contexts or features, that we will use to define the distributional properties of these exemplars; finally D is a set of grammatical strings, the *data*; a finite subset of the language. $F \odot KK$ using the notation above is $\{lur | u, v \in K, (l, r) \in F\}$. This is the finite part of the language that we examine. If the language

we are modeling is L , then $D = L \cap (F \odot KK)$. Since $\lambda \in K, K \subseteq KK$.

We define a concept to be an ordered pair $\langle S, C \rangle$ where $S \subseteq K$ and $C \subseteq F$, which satisfies the following two conditions: first $C \odot S \subseteq D$; that is to say every string in S can be combined with any context in C to give a grammatical string, and secondly they are maximal in that neither K nor F can be increased without violating the first condition.

We define $\mathfrak{B}(K, D, F)$ to be the set of all such concepts. We use the \mathfrak{B} symbol (*Begriff*) to bring out the links to Formal Concept Analysis (Ganter and Wille, 1997; Davey and Priestley, 2002). This lattice may contain exponentially many concepts, but it is clearly finite, as the number of concepts is less than $\min(2^{|F|}, 2^{|K|})$.

There is an obvious partial order defined by $\langle S_1, C_1 \rangle \leq \langle S_2, C_2 \rangle$ iff $S_1 \subseteq S_2$, Note that $S_1 \subseteq S_2$ iff $C_2 \subseteq C_1$.

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 .

$$S' = \{(l, r) \in F : \forall w \in S, lwr \in D\}$$

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

$$C' = \{w \in K : \forall (l, r) \in C, lwr \in D\}$$

The concepts $\langle S, C \rangle$ are just the pairs that satisfy $S' = C$ and $C' = S$; the two maps denoted by $'$ are called the polar maps. For any $S \subseteq K$, $S''' = S'$ and for any $C \subseteq F$, $C''' = C'$. Thus we can form a concept from any set of strings $S \subseteq K$ by taking $\langle S'', S' \rangle$; this is a concept as $S''' = S'$. We will write this as $\mathcal{C}(S)$, and for any $C \subseteq F$, we will write $\mathcal{C}(C) = \langle C', C'' \rangle$.

If $S \subseteq T$ then $T' \subseteq S'$, and $S'' \subseteq T''$. For any set of strings $S \subseteq K, S \subseteq S''$.

One crucial concept here is the concept defined by (λ, λ) or equivalently by the set $K \cap D$ which corresponds to all of the elements in the language. We will denote this concept by $\mathbf{L} = \mathcal{C}(\{(\lambda, \lambda)\}) = \mathcal{C}(K \cap D)$.

We also define a meet operation by

$$\langle S_1, C_1 \rangle \wedge \langle S_2, C_2 \rangle = \langle S_1 \cap S_2, (S_1 \cap S_2)' \rangle$$

This is the greatest lower bound of the two concepts; this is a concept since if $S_1'' = S_1$ and

$S_2'' = S_2$ then $(S_1 \cap S_2)'' = (S_1 \cap S_2)$. Note that this operation is associative and commutative. We can also define a join operation dually; with these operations $\mathfrak{B}(K, D, D)$ is a complete lattice.

So far we have only used strings in $F \odot K$; we now define a concatenation operation as follows.

$$\langle S_1, C_1 \rangle \circ \langle S_2, C_2 \rangle = \langle (S_1 S_2)'', (S_1 S_2)''' \rangle$$

Since S_1 and S_2 are subsets of K , $S_1 S_2$ is a subset of KK , but not necessarily of K . $(S_1 S_2)'$ is the set of contexts shared by all elements of $S_1 S_2$ and $(S_1 S_2)''$ is the subset of K , not KK , that has all of the contexts of $(S_1 S_2)'$. $(S_1 S_2)'''$ might be larger than $(S_1 S_2)'$. We can also write this as $\mathcal{C}((S_1 S_2)')$.

Both \wedge and \circ are monotonic in the sense that if $X \leq Y$ then $X \circ Z \leq Y \circ Z$, $Z \circ X \leq Z \circ Y$ and $X \wedge Z \leq Y \wedge Z$. Note that all of these operations can be computed efficiently; using a perfect hash, and a naive algorithm, we can do the polar maps and \wedge operations in time $\mathcal{O}(|K||F|)$, and the concatenation in time $\mathcal{O}(|K|^2|F|)$.

We now define the notion of derivation in this representation. Given a string w we recursively compute a concept for every substring of w ; this concept will approximate the distribution of the string. We define ϕ_G as a function from Σ^* to $\mathfrak{B}(K, D, F)$; we define it recursively:

- If $|w| \leq 1$, then $\phi_G(w) = \langle \{w\}'', \{w\}' \rangle$
- If $|w| > 1$ then
$$\phi_G(w) = \bigwedge_{u,v \in \Sigma^+ : uv=w} \phi_G(u) \circ \phi_G(v)$$

The first step is well defined because all of the strings of length at most 1 are already in K so we can look them up directly. To clarify the second step, if $w = abc$ then $\phi_G(abc) = \phi_G(a) \circ \phi_G(bc) \wedge \phi_G(ab) \circ \phi_G(c)$; we compute the string from all possible non-trivial splits of the string into a prefix and a suffix. By using a dynamic programming table that stores the values of $\phi(u)$ for all $u \in \text{Sub}(w)$ we can compute this in time $\mathcal{O}(|K|^2|F||w|^3)$; this is just an elementary variant of the CKY algorithm. We define the language defined by the DLG G to be

$$L(G) = \{w | \phi_G(w) \leq \mathcal{C}(\{(\lambda, \lambda)\})\}$$

That is to say, a string is in the language if we predict that a string has the context (λ, λ) . We now consider a trivial example: the Dyck language.

Example 1. Let L be the Dyck language (matched parenthesis language) over $\Sigma = \{a, b\}$, where a corresponds to open bracket, and b to close bracket. Define

- $K = \{\lambda, a, b, ab\}$
- $F = \{(\lambda, \lambda), (\lambda, b), (a, \lambda)\}$.
- $D = \{\lambda, ab, abab, aabb\}$

$G = \langle K, D, F \rangle$ is a DLG. We will now write down the 5 elements of the lattice:

- $\top = \langle K, \emptyset \rangle$
- $\perp = \langle \emptyset, F \rangle$
- $\mathbf{L} = \langle \{\lambda, ab\}, \{(\lambda, \lambda)\} \rangle$
- $A = \langle \{a\}, \{(\lambda, b)\} \rangle$
- $B = \langle \{b\}, \{(a, \lambda)\} \rangle$

To compute the concatenation $A \circ B$ we first compute $\{a\}\{b\} = \{ab\}$; we then compute $\{ab\}'$ which is $\{(\lambda, \lambda)\}$, and $\{(\lambda, \lambda)\}' = \{\lambda, ab\}$, so $A \circ B = \mathbf{L}$. Similarly to compute $\mathbf{L} \circ \mathbf{L}$, we first take $\{\lambda, ab\}\{\lambda, ab\} = \{\lambda, ab, abab\}$. These all have the context (λ, λ) , so the result is the concept \mathbf{L} . If we compute $A \circ A$ we get $\{a\}\{a\}$ which is $\{aa\}$ which has no contexts so the result is \top . We have $\phi_G(\lambda) = L, \phi_G(a) = A, \phi_G(b) = B$. Applying the recursive computation we can verify that $\phi_G(w) = \mathbf{L}$ iff $w \in L$ and so $L(G) = L$. We can also see that $D = L \cap (F \odot KK)$.

4 Search

In order to learn these grammars we need to find a suitable set of contexts F , a suitable set of strings K , and then work out which elements of $F \odot KK$ are grammatical. So given a choice for K and F it is easy to learn these models under a suitable regime: the details of how we collect information about D depend on the learning model.

The question is therefore whether it is easy to find suitable sets, K and F . Because of the way the formalism is designed, it transpires that the search problem is entirely tractable. In order to analyse the search space, we define two maps between the lattices as K and F are increased. We are going to augment our notation slightly; we will write $\mathfrak{B}(K, L, F)$ for $\mathfrak{B}(K, L \cap (F \odot KK), F)$ and similarly $\langle K, L, F \rangle$ for $\langle K, L \cap (F \odot KK), F \rangle$. When we use the two polar maps

(such as C', S'), though we are dealing with more than one lattice, there is no ambiguity as the maps agree; we will when necessary explicitly restrict the output (e.g. $C' \cap J$) to avoid confusion.

Definition 2. For any language L and any set of contexts $F \subseteq G$, and any sets of strings $J \subseteq K \subseteq \Sigma^*$. We define a map g from $\mathfrak{B}(J, L, F)$ to $\mathfrak{B}(K, L, F)$ (from the smaller lattice to the larger lattice) as $g(\langle S, C \rangle) = \langle C', C \rangle$.

We also define a map f from $\mathfrak{B}(K, L, G)$ to $\mathfrak{B}(K, L, F)$, (from larger to smaller) as $f(\langle S, C \rangle) = \langle (C \cap F)', C \cap F \rangle$.

These two maps are defined in opposite directions: this is because of the duality of the lattice. By defining them in this way, as we will see, we can prove that these two maps have very similar properties. We can verify that the outputs of these maps are in fact concepts.

We now need to define two monotonicity lemmas: these lemmas are crucial to the success of the formalism. We show that as we increase K the language defined by the formalism decreases monotonically, and that as we increase F the language increases monotonically. There is some duplication in the proofs of the two lemmas; we could prove them both from more abstract properties of the maps f, g which are what are called residual maps, but we will do it directly.

Lemma 1. Given two lattices $\mathfrak{B}(K, L, F)$ and $\mathfrak{B}(K, L, G)$ where $F \subseteq G$; For all $X, Y \in \mathfrak{B}(K, L, G)$ we have that

1. $f(X) \circ f(Y) \geq f(X \circ Y)$
2. $f(X) \wedge f(Y) \geq f(X \wedge Y)$

Proof. The proof is elementary but difficult to read. We write $X = \langle S_X, C_X \rangle$ and similarly for Y . For part 1 of the lemma: Clearly $(S'_X \cap F) \subseteq S'_X$, so $(S'_X \cap F)' \supseteq S''_X = S_X$ and the same for S_Y . So $(S'_X \cap F)'(S'_Y \cap F)' \supseteq S_X S_Y$ (as subsets of KK). So $((S'_X \cap F)'(S'_Y \cap F)')' \subseteq (S_X S_Y)' \subseteq (S_X S_Y)'''$. Now by definition, $f(X) \circ f(Y)$ is $\mathcal{C}(Z)$ where $Z = ((S'_X \cap F)'(S'_Y \cap F)')' \cap F$ and $f(X \circ Y)$ has the set of contexts $((S_X S_Y)''' \cap F)$. Therefore $f(X \circ Y)$ has a bigger set of contexts than $f(X) \circ f(Y)$ and is thus a smaller concept. For part 2: by definition $f(X \wedge Y) = \langle ((S_X \cap S_Y)' \cap F)', (S_X \cap S_Y)' \cap F \rangle$ and $f(X) \wedge f(Y) = \langle ((S'_X \cap F)' \cap (S'_Y \cap F)'), ((S'_X \cap F)' \cap (S'_Y \cap F)')' \cap F \rangle$. Now $S'_X \cap F \subseteq S'_X$, so (since $S''_X = S_X$) $S_X \subseteq (S'_X \cap F)'$, and so $S_X \cap S_Y \subseteq (S'_X \cap F)' \cap (S'_Y \cap F)'$.

So $(S_X \cap S_Y)' \supseteq ((S'_X \cap F)' \cap (S'_Y \cap F)')$ which gives the result by comparing the context sets of the two sides of the inequality. \square

Lemma 2. For any language L , and two sets of contexts $F \subseteq G$, and any K , if we have two DLGs $\langle K, L, F \rangle$ with map $\phi_F : \Sigma^* \rightarrow \mathfrak{B}(K, L, F)$ and $\langle K, L, G \rangle$ with map $\phi_G : \Sigma^* \rightarrow \mathfrak{B}(K, L, G)$ then for all w , $f(\phi_G(w)) \leq \phi_F(w)$.

Proof. By induction on the length of w ; clearly if $|w| \leq 1$, $f(\phi_G(w)) = \phi_F(w)$. We now take the inductive step; by definition, (suppressing the definition of u, v in the meet)

$$f(\phi_G(w)) = f\left(\bigwedge_{u,v} \phi_G(u) \circ \phi_G(v)\right)$$

By Lemma 1, part 2:

$$f(\phi_G(w)) \leq \bigwedge_{u,v} f(\phi_G(u) \circ \phi_G(v))$$

By Lemma 1, part 1:

$$f(\phi_G(w)) \leq \bigwedge_{u,v} f(\phi_G(u)) \circ f(\phi_G(v))$$

By the inductive hypothesis we have $f(\phi_G(u)) \leq \phi_F(u)$ and similarly for v and so by the monotonicity of \wedge and \circ :

$$f(\phi_G(w)) \leq \bigwedge_{u,v} \phi_F(u) \circ \phi_F(v)$$

Since the right hand side is equal to $\phi_F(w)$, the proof is done. \square

It is then immediate that

Lemma 3. If $F \subseteq G$ then $L(\langle K, L, F \rangle) \subseteq L(\langle K, L, G \rangle)$,

Proof. If $w \in L(\langle K, L, F \rangle)$, then $\phi_F(w) \leq \mathbf{L}$, and so $f(\phi_G(w)) \leq \mathbf{L}$ and so $\phi_G(w)$ has the context (λ, λ) and is thus in $L(\langle K, L, G \rangle)$. \square

We now prove the corresponding facts about g .

Lemma 4. For any $J \subseteq K$ and any concepts X, Y in $\mathfrak{B}(J, L, F)$, we have that

1. $g(X) \circ g(Y) \geq g(X \circ Y)$
2. $g(X) \wedge g(Y) \geq g(X \wedge Y)$

Proof. For the first part: Write $X = \langle S_X, C_X \rangle$ as before. Note that $S_X = C'_X \cap J$. $S_X \subseteq C'_X$, so $S_X S_Y \subseteq C'_X C'_Y$, and so $(S_X S_Y)'' \subseteq (C'_X C'_Y)''$, and $((S_X S_Y)'' \cap J)' \supseteq (C'_X C'_Y)'''$. By calculation $g(X) \circ g(Y) = \langle (C'_X C'_Y)''', (C'_X C'_Y)'''' \rangle$. On the other hand, $g(X \circ Y) = \langle ((S_X S_Y)'' \cap J)'', ((S_X S_Y)'' \cap J) \rangle$ and so $g(X \circ Y)$ is smaller as it has a larger set of contexts.

For the second part: $g(X) \wedge g(Y) = \langle C'_X \cap C'_Y, (C'_X \cap C'_Y)' \rangle$ and $g(X \wedge Y) = \langle (S_X \cap S_Y)'', (S_X \cap S_Y)' \rangle$. Since $S_X = C'_X \cap J$, $S_X \subseteq C'_X$, so $(S_X \cap S_Y) \subseteq C'_X \cap C'_Y$, and therefore $(S_X \cap S_Y)'' \subseteq (C'_X \cap C'_Y)'' = C'_X \cap C'_Y$. \square

We now state and prove the monotonicity lemma for g .

Lemma 5. For all $J \subseteq K \subseteq \Sigma^* \times \Sigma^*$, and for all strings w ; we have that $g(\phi_J(w)) \leq \phi_K(w)$.

Proof. By induction on length of w . Both J and K include the basic elements of Σ and λ . First suppose $|w| \leq 1$, then $\phi_J(w) = \langle (C_L(w) \cap F)' \cap J, C_L(w) \cap F \rangle$, and $g(\phi_J(w)) = \langle (C_L(w) \cap F)'', C_L(w) \cap F \rangle$ which is equal to $\phi_K(w)$.

Now suppose true for all w of length at most k , and take some w of length $k + 1$. By definition of ϕ_J :

$$g(\phi_J(w)) = g\left(\bigwedge_{u,v} \phi_J(u) \circ \phi_J(v)\right)$$

Next by Lemma 4, Part 2

$$g(\phi_J(w)) \leq \bigwedge_{u,v} g(\phi_J(u) \circ \phi_J(v))$$

By Lemma 4, Part 1

$$g(\phi_J(w)) \leq \bigwedge_{u,v} g(\phi_J(u)) \circ g(\phi_J(v))$$

By the inductive hypothesis and monotonicity of \circ and \wedge :

$$g(\phi_J(w)) \leq \bigwedge_{u,v} \phi_K(u) \circ \phi_K(v) = \phi_K(w)$$

\square

Lemma 6. If $J \subseteq K$ then $L(\langle J, L, F \rangle) \supseteq L(\langle K, L, F \rangle)$

Proof. Suppose $w \in L(\langle K, L, F \rangle)$. this means that $\phi_K(w) \leq L_K$. therefore $g(\phi_J(w)) \leq L_k$; which means that (λ, λ) is in the concept $g(\phi_J(w))$, which means it is in the concept $\phi_J(w)$, and therefore $w \in L(\langle J, L, F \rangle)$. \square

Given these two lemmas we can make the following observations. First, if we have a fixed L and F , then as we increase K , the language will decrease until it reaches a limit, which it will attain after a finite limit.

Lemma 7. *For all L , and finite context sets F , there is a finite K such that for all K_2 , $K \subset K_2$, $L(\langle K, L, F \rangle) = L(\langle K_2, L, F \rangle)$.*

Proof. We can define the lattice $\mathfrak{B}(\Sigma^*, L, F)$. Define the following equivalence relation between pairs of strings, where $(u_1, v_1) \sim (u_2, v_2)$ iff $\mathcal{C}(u_1) = \mathcal{C}(u_2)$ and $\mathcal{C}(v_1) = \mathcal{C}(v_2)$ and $\mathcal{C}(u_1 v_1) = \mathcal{C}(u_2 v_2)$. The number of equivalence classes is clearly finite. If K is sufficiently large that there is a pair of strings (u, v) in K for each equivalence class, then clearly the lattice defined by this K will be isomorphic to $\mathfrak{B}(\Sigma^*, L, F)$. Any superset of K will not change this lattice. \square

Moreover this language is unique for each L, F . We will call this the limit language of L, F , and we will write it as $L(\langle \Sigma^*, L, F \rangle)$.

If $F \subseteq G$, then $L(\langle \Sigma^*, L, F \rangle) \subseteq L(\langle \Sigma^*, L, G \rangle)$. Finally, we will show that the limit languages never overgeneralise.

Lemma 8. *For any L , and for any F , $L(\langle \Sigma^*, L, F \rangle) \subseteq L$.*

Proof. Recall that $\mathcal{C}(w) = \langle \{w\}'', \{w\}' \rangle$ is the real concept. If G is a limit grammar, we can show that we always have $\phi_G(w) > \mathcal{C}(w)$, which will give us the result immediately. First note that $\mathcal{C}(u) \circ \mathcal{C}(v) \geq \mathcal{C}(uv)$, which is immediate by the definition of \circ . We proceed, again, by induction on the length of w . For $|w| \leq 1$, $\phi_G(w) = \mathcal{C}(w)$. For the inductive step we have $\phi_G(w) = \bigwedge_{u,v} \phi_G(u) \circ \phi_G(v)$; by inductive hypothesis we have that this must be more than $\bigwedge_{u,v} \mathcal{C}(u) \circ \mathcal{C}(v) > \bigwedge_{u,v} \mathcal{C}(uv) = \mathcal{C}(w)$ \square

5 Weak generative power

First we make the following observation: if we consider an infinite variant of this, where we set $K = \Sigma^*$ and $F = \Sigma^* \times \Sigma^*$ and $D = L$, we can prove easily that, allowing infinite ‘‘representations’’, for any L , $L(\langle K, D, F \rangle) = L$. In this infinite data limit, \circ becomes associative, and the structure of $\mathfrak{B}(K, D, F)$ becomes a residuated lattice, called the syntactic concept lattice of the language L , $\mathfrak{B}(L)$. This lattice is finite iff the language is regular. The fact that this lattice now has

residuation operations suggest interesting links to the theory of categorial grammar. It is the finite case that interests us.

We will use \mathcal{L}_{DLG} to refer to the class of languages that are limit languages in the sense defined above.

$$\mathcal{L}_{\text{DLG}} = \{L \mid \exists F, L(\langle \Sigma^*, L, F \rangle) = L\}$$

Our focus in this paper is not on the language theory: we present the following propositions. First \mathcal{L}_{DLG} properly contains the class of regular languages. Secondly \mathcal{L}_{DLG} contains some non-context-free languages (Clark, 2009). Thirdly it does not contain all context-free languages.

A natural question to ask is how to convert a CFG into a DLG. This is in our view the wrong question, as we are not interested in modeling CFGs but modeling natural languages, but given the status of CFGs as a default model for syntactic structure, it will help to give a few examples, and a general mechanism. Consider a non-terminal N in a CFG with start symbol S . We can define $C(N) = \{(l, r) \mid S \xrightarrow{*} lNr\}$ and the yield $Y(N) = \{w \mid N \xrightarrow{*} w\}$. Clearly $C(N) \odot Y(N) \subseteq L$, but these are not necessarily maximal, and thus $\langle C(N), Y(N) \rangle$ is not necessarily a concept. Nonetheless in most cases, we can construct a grammar where the non-terminals will correspond to concepts, in this way.

The basic approach is this: for each non-terminal, we identify a finite set of contexts that will pick out only the set of strings generated from that non-terminal: we find some set of contexts F_N typically a subset of $C(N)$ such that $Y(N) = \{w \mid \forall (l, r) \in F_N, lwr \in L\}$. We say that we can *contextually define* this non-terminal if there is such a finite set of contexts F_N . If a CFG in Chomsky normal form is such that every non-terminal can be contextually defined then the language defined by that grammar is in \mathcal{L}_{DLG} . If we can do that, then the rest is trivial. We take any set of features F that includes all of these F_N ; probably just $F = \bigcup_N F_N$; we then pick a set of strings K that is sufficiently large to rule out all incorrect generalisations, and then define D to be $L \cap (F \odot KK)$.

Consider the language $L = \{a^n b^n c^m \mid n, m \geq 0\} \cup \{a^m b^n c^n \mid n, m \geq 0\}$. L is a classic example of an inherently ambiguous and thus non-deterministic language.

The natural CFG in CNF for L has non-terminals that generate the following

sets: $\{a^n b^n | n \geq 0\}$, $\{a^{n+1} b^n | n \geq 0\}$, $\{b^n c^n | n \geq 0\}$, $\{b^n c^{n+1} | n \geq 0\}$, $\{a^*\}$ and $\{c^*\}$. We note that the six contexts (aa, bbc) , $(aa, bbcc)$, $(abb, cc)(abbb, cc)$, (λ, a) and (c, λ) will define exactly these sets, in the sense that the set of strings that occur in each context will be exactly the corresponding set. We can also pick out λ, a, b, c with individual contexts. Let $F = \{(\lambda, \lambda), (aaabb, bccc), (aaabbc, \lambda), (\lambda, abbccc), (aaab, bccc), (aa, bbc), (aa, bbcc), (abb, cc), (abbb, cc), (\lambda, a), (c, \lambda)\}$. If we take a sufficiently large set K , say $\lambda, a, b, c, ab, aab, bc, bcc, abc$, and set $D = L \cap F \odot KK$, then we will have a DLG for the language L . In this example, it is sufficient to have one context per non-terminal. This is not in general the case.

Consider $L = \{a^n b^n | n \geq 0\} \cup \{a^n b^{2n} | n \geq 0\}$. Here we clearly need to identify sets of strings corresponding to the two parts of this language, but it is easy to see that no one context will suffice. However, note that the first part is defined by the two contexts $(\lambda, \lambda), (a, b)$ and the second by the two contexts $(\lambda, \lambda), (a, bb)$. Thus it is sufficient to have a set F that includes these four contexts, as well as similar pairs for the other non-terminals in the grammar, and some contexts to define a and b .

We can see that we will not always be able to do this for every CFG. One fixable problem is if the CFG has two separate non-terminals, M, N such that $C(M) \supseteq C(N)$. If this is the case, then we must have that $Y(N) \supseteq Y(M)$. If we pick a set of contexts to define $Y(N)$, then clearly any string in $Y(M)$ will also be picked out by the same contexts. If this is not the case, then we can clearly try to rectify it by adding a rule $N \rightarrow M$ which will not change the language defined.

However, we cannot always pick out the non-terminals with a *finite* set of contexts. Consider the language $L = \{a^n b | n > 0\} \cup \{a^n c^m | m > n > 0\}$ defined in Clark et al. (2008). Suppose wlog that F contains no context (l, r) such that $|l| + |r| \geq k$. Then it is clear that we will not be able to pick out b without also picking out c^{k+1} , since $C_L(c^{k+1}) \cap F \supseteq C_L(b) \cap F$. Thus L , which is clearly context-free, is not in \mathcal{L}_{DLG} . Luckily, this example is highly artificial and does not correspond to any phenomena we are aware of in linguistics.

In terms of representing natural languages, we clearly will in many cases need more than one

context to pick out syntactically relevant groups of strings. Using a very simplified example from English, if we want to identify say singular noun phrases, a context like (that is, λ) will not be sufficient since as well as noun phrases we will also have some adjective phrases. However if we include multiple contexts such as $(\lambda, \text{is over there})$ and so on, eventually we will be able to pick out exactly the relevant set of strings. One of the reasons we need to use a context sensitive representation, is so that we can consider every possible combination of contexts simultaneously: this would require an exponentially large context free grammar.

6 Learning Model

In order to prove correctness of the learning algorithm we will use a variant of Gold-style inductive inference (Gold, 1967). Our choice of this rather old-fashioned model requires justification. There are two problems with learning – the information theoretic problems studied under VC-dimension etc., and the computational complexity issues of constructing a hypothesis from the data. In our view, the latter problems are the key ones. Accordingly, we focus entirely on the efficiency issue, and allow ourself a slightly unrealistic model; see (Clark and Lappin, 2009) for arguments that this is a plausible model.

We assume that we have a sequence of positive examples, and that we can query examples for membership. Given a language L a presentation for L is an infinite sequence of strings w_1, w_2, \dots such that $\{w_i | i \in \mathbb{N}\} = L$. 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 – polynomial in the total size of the presentation. 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 (Pitt, 1989).

7 Learning Algorithm

We now define a simple learning algorithm, that establishes learnability under this paradigm.

There is one minor technical detail we need to deal with. We need to be able to tell when adding a string to a lazy DLG will leave the grammar unchanged. We use a slightly weaker test. Given $G_1 = \langle K, D, F \rangle$ we define as before the equivalence relation between pairs of strings of K , where $(u_1, v_1) \sim_{G_1} (u_2, v_2)$ iff $C_D(u_1) = C_D(u_2)$ and $C_D(v_1) = C_D(v_2)$ and $C_D(u_1v_1) = C_D(u_2v_2)$. Note that $C_D(u) = \{(l, r) | lur \in D\}$.

Given two grammars $G_1 = \langle K, D, F \rangle$ and $G_2 = \langle K_2, D_2, F \rangle$ where $K \subseteq K_2$ and $D \subseteq D_2$ but F is unchanged, we say that these two are *indistinguishable* iff the number of equivalence classes of $K \times K$ under \sim_{G_1} is equal to the number of equivalence classes of $K_2 \times K_2$ under \sim_{G_2} . This can clearly be computed efficiently using a union-find algorithm, in time polynomial in $|K|$ and $|F|$. If they are indistinguishable then they define the same language.

7.1 Algorithm

Algorithm 1 presents the basic algorithm. At various points we compute sets of strings like $(F \odot KK) \cap L$; these can be computed using the membership oracle.

First we prove that the program is efficient in the sense that it runs in polynomial update time.

Lemma 9. *There is a polynomial p , such that Algorithm 1, for each w_n , runs in time bounded by $p(n, l)$ where l is the maximum length of a string in w_1, \dots, w_n .*

Proof. First we note that K, K_2 and F are always subsets of $Sub(E) \cup \Sigma$ and $Con(E)$, and thus both $|K|$ and $|F|$ are bounded by $nl(l+1)/2 + |\Sigma| + 1$. Computing D is efficient as $|F \odot KK|$ is bounded by $|K|^2|F|$. We can compute ϕ_G as mentioned above in time $|K|^2|F|l^3$; distinguishability is as observed earlier also polynomial. \square

Before we prove the correctness of the algorithm we make some informal points. First, we are learning under a rather pessimistic model – the positive examples may be chosen to confuse us, so we cannot make any assumptions. Accordingly we have to very crudely add all substrings and all

Algorithm 1: DLG learning algorithm

Data: Input strings $S = \{w_1, w_2, \dots\}$, membership oracle \mathcal{O}

Result: A sequence of DLGs G_1, G_2, \dots

$K \leftarrow \Sigma \cup \{\lambda\}, K_2 = K;$

$F \leftarrow \{(\lambda, \lambda)\}, E = \{ \};$

$D = (F \odot KK) \cap L;$

$G = \langle K, D, F \rangle;$

for w_i **do**

$E \leftarrow E \cup \{w_i\};$

$K_2 \leftarrow K_2 \cup Sub(w_i);$

if *there is some $w \in E$ that is not in*

$L(G)$ **then**

$F \leftarrow Con(E);$

$K \leftarrow K_2;$

$D = (F \odot KK) \cap L;$

$G = \langle K, D, F \rangle;$

end

else

$D_2 \leftarrow (F \odot K_2K_2) \cap L;$

if $\langle K_2, D_2, F \rangle$ *not indistinguishable*

from $\langle K, D, F \rangle$ **then**

$K \leftarrow K_2;$

$D = (F \odot KK) \cap L;$

$G = \langle K, D, F \rangle;$

end

end

 Output G ;

end

contexts, rather than using sensible heuristics to select frequent or likely ones.

Intuitively the algorithm works as follows: if we observe a string not in our current hypothesis, then we increase the set of contexts which will increase the language defined. Since we only see positive examples, we will never explicitly find out that our hypothesis overgenerates, accordingly we always add strings to a tester set K_2 and see if this gives us a more refined model. If this seems like it might give a tighter hypothesis, then we increase K .

In what follows we will say that the hypothesis at step n , $G_n = \langle K_n, D_n, F_n \rangle$, and the language defined is L_n . We will assume that the target language is some $L \in \mathcal{L}_{DLG}$ and w_1, \dots is a presentation of L .

Lemma 10. *Then there is a point n , and a finite set of contexts F such that for all $N > n$, $F_N = F$, and $L(\langle \Sigma^*, L, F \rangle) = L$.*

Proof. Since $L \in \mathcal{L}_{DLG}$ there is some set of con-

texts $G \subset \text{Con}(L)$, such that $L = L(\langle \Sigma^*, L, G \rangle)$. Any superset of G will define the correct limit language. Let n be the smallest n such that G is a subset of $\text{Con}(\{w_1, \dots, w_n\})$. Consider F_n . If F_n defines the correct limit language, then we will never change F as the hypothesis will be a superset of the target. Otherwise it must define a subset of the correct language. Then either there is some $N > n$ at which it has converged to the limit language which will cause the first condition in the loop to be satisfied and F will be increased to a superset of G , or F will be increased before it converges, and thus the result holds. \square

Lemma 11. *After F converges according to the previous lemma, there is some n , such that for all $N > n$, $K_N = K_n$ and $L(\langle K_n, L, F_n \rangle) = L$.*

Proof. let n_0 be the convergence point of F ; for all $n > n_0$ the hypothesis will be a superset of the target language; therefore the only change that can happen is that K will increase. By definition of the limit language, it must converge after a finite number of examples. \square

Theorem 1. *For every language $L \in \mathcal{L}_{\text{DLG}}$, and every presentation of L , Algorithm 1 will converge to a grammar G such that $L(G) = L$.*

This result is immediate by the two preceding lemmas.

8 Conclusion

We have presented an efficient, correct learning algorithm for an interesting class of languages; this is the first such learning result for a class of languages that is potentially large enough to describe natural language.

The results presented here lack a couple of technical details to be completely convincing. In particular we would like to show that given a representation of size n , we can learn once we have seen a set of examples that is polynomially bounded by n . This will be challenging, as the size of the K we need to converge can be exponentially large in F . We can construct DFAs where the number of congruence classes of the language is an exponential function of the number of states. In order to learn languages like this, we will need to use a more efficient algorithm that can learn even with “insufficient” K : that is to say when the lattice $\mathfrak{B}(K, L, F)$ has fewer elements than $\mathfrak{B}(KK, L, F)$.

This algorithm can be implemented directly and functions as expected on synthetic examples, but would need modification to run efficiently on natural languages. In particular rather than considering whole contexts of the form (l, r) it would be natural to restrict them just to a narrow window of one or two words or tags on each side. Rather than using a membership oracle, we could probabilistically cluster the data in the table of counts of strings in $F \odot K$. In practice we will have a limited amount of data to work with and we can control over-fitting in a principled way by controlling the relative size of K and F .

This formalism represents a process of analogy from stored examples, based on distributional learning – this is very plausible in terms of what we know about cognitive processes, and is compatible with much non-Chomskyan theorizing in linguistics (Blevins and Blevins, 2009). The class of languages is a good fit to the class of natural languages; it contains, as far as we can tell, all standard examples of context free grammars, and includes non-deterministic and inherently ambiguous grammars. It is hard to say whether the class is in fact large enough to represent natural languages; but then we don’t know that about any formalism, context-free or context-sensitive. All we can say is that there are no phenomena that we are aware of that don’t fit. Only large scale empirical work can answer this question.

Ideologically these models are empiricist – the structure of the representation is based on the structure of the data: this has to be a good thing for computational modeling. By minimizing the amount of hidden, unobservable structure, we can improve learnability. Languages are enormously complex, and it would be simplistic to try to reduce their acquisition to a few pages of mathematics; nonetheless, we feel that the representations and grammar induction algorithms presented in this paper could be a significant piece of the puzzle.

References

- N. Abe and M. K. Warmuth. 1992. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, 9:205–260.
- D. Angluin and M. Kharitonov. 1995. When won't membership queries help? *J. Comput. Syst. Sci.*, 50:336–355.
- D. Angluin. 1987. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106.
- James P. Blevins and Juliette Blevins. 2009. *Analogy in grammar: Form and acquisition*. Oxford University Press.
- A. Carnie. 2008. *Constituent structure*. Oxford University Press, USA.
- Noam Chomsky. 2006. *Language and mind*. Cambridge University Press, 3rd edition.
- Alexander Clark and Shalom Lappin. 2009. Another look at indirect negative evidence. In *Proceedings of the EACL Workshop on Cognitive Aspects of Computational Language Acquisition*, Athens, March.
- 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.
- Alexander Clark. 2009. A learnable representation for syntax using residuated lattices. In *Proceedings of the 14th Conference on Formal Grammar*, Bordeaux, France.
- J.R. Curran. 2003. *From distributional to semantic similarity*. Ph.D. thesis, University of Edinburgh.
- B. A. Davey and H. A. Priestley. 2002. *Introduction to Lattices and Order*. Cambridge University Press.
- B. Ganter and R. Wille. 1997. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag.
- E. M. Gold. 1967. Language identification in the limit. *Information and control*, 10(5):447 – 474.
- Zellig Harris. 1954. Distributional structure. *Word*, 10(2-3):146–62.
- J. J. Horning. 1969. *A Study of Grammatical Inference*. Ph.D. thesis, Stanford University, Computer Science Department, California.
- M. Johnson. 2008. Using adaptor grammars to identify synergies in the unsupervised acquisition of linguistic structure. In *46th Annual Meeting of the ACL*, pages 398–406.
- Dan Klein and Chris Manning. 2001. Distributional phrase structure induction. In *Proceedings of CoNLL 2001*, pages 113–121.
- Dan Klein and Chris Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting of the ACL*.
- K. Lari and S. J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56.
- L. Pitt. 1989. Inductive inference, dfa's, and computational complexity. In K. P. Jantke, editor, *Analogical and Inductive Inference*, number 397 in LNAI, pages 18–44. Springer-Verlag.
- J. R. Saffran, R. N. Aslin, and E. L. Newport. 1996. Statistical learning by eight month old infants. *Science*, 274:1926–1928.
- Hinrich Schütze. 1993. Part of speech induction from scratch. In *Proceedings of the 31st annual meeting of the Association for Computational Linguistics*, pages 251–258.