Journal of Machine Learning Research 8 (2007) 1725-1745 Submitted 3/07; Revised 5/07; Published 8/07

Polynomial Identification in the Limit of Substitutable Context-free Languages

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

Rémi Eyraud REMI.EYRAUD@LIF.UNIV-MRS.FR
Laboratoire d'Informatique Fondamentale
Centre de Mathmatiques et Informatique
39 rue Joliot-Curie
Marseille Cedex 13
FRANCE

Editor: Rocco Servedio

Abstract

This paper formalises the idea of substitutability introduced by Zellig Harris in the 1950s and makes it the basis for a learning algorithm from positive data only for a subclass of context-free languages. We show that there is a polynomial characteristic set, and thus prove polynomial identification in the limit of this class. We discuss the relationship of this class of languages to other common classes discussed in grammatical inference. It transpires that it is not necessary to identify constituents in order to learn a context-free language—it is sufficient to identify the syntactic congruence, and the operations of the syntactic monoid can be converted into a context-free grammar. We also discuss modifications to the algorithm that produces a reduction system rather than a context-free grammar, that will be much more compact. We discuss the relationship to Angluin's notion of reversibility for regular languages. We also demonstrate that an implementation of this algorithm is capable of learning a classic example of structure dependent syntax in English: this constitutes a refutation of an argument that has been used in support of nativist theories of language.

Keywords: grammatical inference, context-free languages, positive data only, reduction system, natural languages

1. Introduction

Current techniques for grammatical inference have for a long time been focused to a great extent on learnable subclasses of regular languages. For many application domains though, there are structural dependencies in the data that are more naturally modelled by context-free grammars of various types. One of the oldest ideas for a grammatical inference algorithm, and one geared towards context-free inference, is Harris's use of substitutability (Chomsky, 1953; Harris, 1954). Though this has formed the intuitive motivation for a number of grammatical inference algorithms before, it has never been adequately formalized. In this paper we present an explicit mathematical formalization of this idea of substitutability and use it to define a subclass of context-free languages that we call the *substitutable* languages, that can be learned according to the polynomial identification in the limit paradigm (de la Higuera, 1997). These languages are not comparable to the very simple languages, but seem better suited to be the basis for algorithms that can learn natural languages.

In this paper we use a polynomial variant of Gold's identification in the limit (IIL) paradigm, working from positive data only. We hope in the future to be able to extend this to a more practical PAC-learning result, but in the meantime work in this paradigm allows some foundational issues to be addressed. The contribution of the work presented in this paper lies in two main directions. First we demonstrate that it is not necessary to learn constituent structure in order to learn context-free grammars. Indeed, it is sufficient to be able to learn the syntactic congruence: the syntactic monoid can be converted into a context-free grammar in Chomsky normal form in a very natural way. Secondly, we define a simple, purely language theoretic criterion, which allows the syntactic congruence to be identified very easily. This criterion is sufficient to guarantee polynomial identification in the limit.

Additionally, we point out the relationship to NTS grammars, which in our opinion are an unarticulated assumption underlying many algorithms in GI and unsupervised learning of natural language (see for instance the work of Adriaans et al., 2000; van Zaanen, 2002; Solan et al., 2004).

The key to the Harris approach for learning a language L, is to look at pairs of strings u and v and to see whether they occur in the same contexts; that is to say, to look for pairs of strings of the form lur and lvr that are both in L. This can be taken as evidence that there is a non-terminal symbol that generates both strings. In the informal descriptions of this, there is an ambiguity between two ideas. The first is that they should appear in *all* the same contexts; and the second is that they should appear in *some* of the same contexts. We can write the first criterion as follows: (we define our notation more formally in the next section, but we hope the reader will bear with us for the moment)

$$\forall l, r \; lur \in L \text{ if and only if } lvr \in L. \tag{1}$$

The second, weaker, criterion is

$$\exists l, r \; lur \in L \text{ and } lvr \in L. \tag{2}$$

The problem is then that to draw conclusions about the structure of the language, one needs the former; but all one can hope for by observation of given data is the latter. In general, the class of context-free grammars will be unlearnable: certainly according to the Gold style approach we take in this paper since it is a superfinite class. Therefore to obtain learnability results we must define subclasses of the languages that sufficiently restrict the class so that learning can take place. The restriction we consider here is that whenever two strings have one context in common, then they have all contexts in common: Equation 2 implies Equation 1. We call these the *substitutable* languages.

Our main formal result is that this simple, but powerful constraint on languages—and note that it is expressed in purely language theoretic terms—sufficiently restricts the class of context-free languages to the extent that it can be learned using a simple polynomial algorithm. In this case, we can learn according to the IIL criterion, and the algorithm will be polynomial in the amount of data it needs (the characteristic set) and in computation.

More generally, this work shows how one can move from the syntactic congruence of a context-free language to a grammar for that language under certain assumptions. This can be done through a remarkably simple construction, and finally provides a solid theoretical grounding for the numerous empirical algorithms based on different heuristics that have relied on learning the syntactic congruence.

2. Definitions

We start by defining some standard notation.

An *alphabet* Σ is a finite nonempty set of symbols called *letters*. A *string* w over Σ is a finite sequence $w = a_1 a_2 \dots a_n$ of letters. Let $|w|$ denote the length of w. In the following, letters will be indicated by $a, b, c, \dots$, strings by $u, v, \dots, z$, and the empty string by λ. We shall write $|u|_a$ for the number of occurrences of the letter a in the string u. Let Σ^* be the set of all strings, the free monoid generated by Σ. By a language we mean any subset $L \subseteq \Sigma^*$. The set of all substrings of a language L is denoted $Sub(L) = \{u \in \Sigma^+ : \exists l, r \in \Sigma^* \text{ such that } lur \in L\}$ (notice that the empty word does not belong to $Sub(L)$). We shall assume an order $\prec$ or $\preceq$ on Σ which we shall extend to Σ^* in the normal way by saying that $u \prec v$ if $|u| < |v|$ or $|u| = |v|$ and u is lexicographically before v.

In general, and though it is not the case of the main class studied in this paper, the definition of a class of languages $\mathbb{L}$ relies on a class $\mathbb{R}$ of abstract machines, here called *representations*, together with a function $\mathcal{L}$ from representations to languages, that characterize all and only the languages of $\mathbb{L}$: (1) $\forall R \in \mathbb{R}, \mathcal{L}(R) \in \mathbb{L}$ and (2) $\forall L \in \mathbb{L}, \exists R \in \mathbb{R}$ such that $\mathcal{L}(R) = L$. Two representations R_1 and R_2 are *equivalent iff* $\mathcal{L}(R_1) = \mathcal{L}(R_2)$.

Definition 1 (Grammar) *A grammar is a quadruple $G = \langle \Sigma, V, P, S \rangle$ where Σ is a finite alphabet of terminal symbols, V is a (distinct) finite alphabet of variables or non-terminals, P is a finite set of production rules, and $S \in V$ is a start symbol.*

If $P \subseteq V \times (\Sigma \cup V)^+$ then the grammar is said to be context-free (CF), and we will write the productions as $T \to \alpha$.

Note that we do not allow empty right hand sides to the productions and thus these grammars will not generate the empty string.

We will write $uTv \Rightarrow u\alpha v$ when $T \to \alpha \in P$. $\overset{*}{\Rightarrow}$ is the reflexive and transitive closure of $\Rightarrow$.

We denote by $\mathcal{L}(G) = \{w \in \Sigma^* : S \overset{*}{\Rightarrow}_G w\}$ the language defined by the grammar. Since we do not allow rules with an empty right hand side this language cannot contain λ.

Definition 2 (Syntactic congruence) *We say that two words u and v are syntactically congruent* `w.r.t.` *a language L, written $u \equiv_L v$, if and only if $\forall l, r \in \Sigma^*$ $lur \in L$ iff $lvr \in L$.*

We can think of this syntactic congruence as the strong notion of substitutability. Note two things: first this is clearly an equivalence relation, and secondly, it is a congruence of the monoid Σ^*, that is,

$$u \equiv_L v \text{ implies } \forall l, r \; lur \equiv_L lvr.$$

The syntactic monoid of the language L is just the quotient of Σ^* by this relation. It is a standard result that this will be finite if and only if L is regular.

We shall write $[u]_L$ for the equivalence class of the string u under $\equiv_L$.

Example 1 *Consider the language $L = \{u \in \{a, b\}^* : |u|_a = |u|_b\}$. We can see that $u \equiv_L v$ iff $(|u|_a - |u|_b) = (|v|_a - |v|_b)$; the congruence classes will thus correspond to particular values of $(|u|_a - |u|_b)$ and the syntactic monoid will be isomorphic to $\mathbb{Z}$.*

Another way of looking at this relation is to define the set of contexts of a string:

Definition 3 (Set of contexts) *The set of contexts of a string u in a language L is written $C_L(u)$ and defined as $C_L(u) = \{(l,r) : lur \in L\}$.*

Using this definition we can say that $u \equiv_L v$ if and only if $C_L(u) = C_L(v)$.

We define the weaker idea of substitutability that we will use in the following way.

Definition 4 (Weak substitutability) *Given a language L, we say that two words u and v are weakly substitutable* `w.r.t.` *L, written $u \doteq_L v$, if there exist $l,r \in \Sigma^*$ such that $lur \in L$ and $lvr \in L$.*

Note that this is in general not a congruence or even a transitive relation. Normally we will have a finite sample S of the language L: clearly $u \doteq_S v$ implies $u \doteq_L v$.

We now can define the class of languages that we are concerned with:

Definition 5 (Substitutable language) *A language L is substitutable if and only if for every pair of strings u,v, $u \doteq_L v$ implies $u \equiv_L v$.*

In terms of contexts we can say that a language is substitutable, if whenever the sets of contexts of two strings have non-empty intersection, they are identical. The substitutable context-free languages are just those languages that are both substitutable and context-free. A number of examples can be found in Section 6.

Lemma 6 *There are languages which are substitutable but not context-free.*

Proof Let $\Sigma = \{a,b,c,d\}$ be the alphabet. The language $L = \{u \in \Sigma^* : \ |u|_a = |u|_b \wedge |u|_c = |u|_d\}$ is substitutable, as can easily be verified: the syntactic monoid here is isomorphic to $\mathbb{Z} \times \mathbb{Z}$. The intersection of L with the regular language $\{a^*c^*b^*d^*\}$ gives the language $\{a^n c^m b^n d^m : n,m > 0\}$ which is not context-free; therefore L is not context-free. ■

2.1 Learning

We now define our learning criterion. This is identification in the limit from positive text (Gold, 1967), with polynomial bounds on data and computation (de la Higuera, 1997), but not on errors of prediction (Pitt, 1989).

A learning algorithm A for a class of representations $\mathbb{R}$, is an algorithm that computes a function from a finite sequence of strings $s_1,\ldots,s_n$ to $\mathbb{R}$. We define a presentation of a language L to be an infinite sequence of elements of L such that every element of L occurs at least once. Given a presentation, we can consider the sequence of hypotheses that the algorithm produces, writing $R_n = A(s_1,\ldots s_n)$ for the nth such hypothesis.

The algorithm A is said to identify the class $\mathbb{R}$ in the limit if for every $R \in \mathbb{R}$, for every presentation of $\mathcal{L}(R)$, there is an N such that for all $n > N$, $R_n = R_N$ and $\mathcal{L}(R) = \mathcal{L}(R_N)$.

We further require that the algorithm needs only polynomially bounded amounts of data and computation. We use the slightly weaker notion defined by de la Higuera (1997); note that the size of a set of strings S is defined as $\sum_{w\in S} |w|$.

Definition 7 (Polynomial identification in the limit) *A representation class $\mathbb{R}$ is identifiable in the limit from positive data with polynomial time and data iff there exist two polynomials $p(), q()$ and an algorithm A such that*

1. *Given a positive sample S of size m A returns a representation $R \in \mathbb{R}$ in time $p(m)$*
2. *For each representation R of size n there exists a characteristic set CS of size less than $q(n)$ such that if $CS \subseteq S$, A returns a representation R' such that $\mathcal{L}(R) = \mathcal{L}(R')$.*

However, this definition initially designed for the learning of regular languages, is somehow unsuitable as a model for learning context-free grammars.

Example 2 *A context-free grammar G_n with the one letter alphabet $\{a\}$ is defined as follows: it contains a set of non-terminals $N_1, \ldots N_n$. The productions consist of $N_i \rightarrow N_{i-1}N_{i-1}$ for $i = 2, \ldots n$ and $N_1 \rightarrow aa$. The sentence symbol is N_n. It can easily be seen that $L(G_n)$ consists of the single string a^{2^n}, yet the size of the representation of G_n is linear in n.*

According to the de la Higuera definition, no non-trivial language class that contains the grammars $G_1, G_2 \ldots$ can be learnable, since any characteristic set for G_n must contain the string a^{2^n}, and will therefore have size exponential in the size of the representation. Yet it seems absurd to insist on a polynomial size characteristic set, when reading a single example requires exponential time.

There is not at present a consensus on the most appropriate modification of this criterion for the learning of context-free grammars. Clearly, relaxing the constraint for polynomial *size* of the characteristic set, to merely requiring polynomial *cardinality*, is unsatisfactory, since it would allow algorithms to use exponential amounts of computation, by specifying an exponentially long string in the characteristic set. Several ideas have been formulated to tackle this problem, such as to focus on shallow languages (Adriaans, 2002) or to require a characteristic sample that is polynomial in a parameter other than the size of the target (Wakatsuki and Tomita, 1993), but none of them represents a consensus.[1] Nonetheless, it is beyond the scope of this paper to attempt to resolve this difficulty, and we shall thus adopt this approach in this paper.

3. Algorithm

We now define an algorithm *SGL* (Substitution Graph Learner), that will learn a context-free grammar from a sample of positive strings of a language.

The primary data structure of our algorithm can be conceived of as a graph, where each node of the graph corresponds to a substring of a string in the sample, and where there is an edge between any two nodes corresponding to substrings u,v if and only if $u \doteq_S v$ where S is the set of positive examples.

In the next section we will define a characteristic set of examples for a context-free grammar, and show that whenever the corresponding context-free language is substitutable, and the sample contains the characteristic set, then *SGL* will produce a grammar that generates the same language as the target.

Definition 8 (Substitution graph) *Given a finite set of words S, we define the substitution graph $SG(S) = (V, E)$ as follow:*

$V = \{u \in \Sigma^+ : \exists l, r \in \Sigma^*, lur \in S\}$,

$E = \{(u, v) \in \Sigma^+ \times \Sigma^+ : u \doteq_S v\}$.

1. One can check that our main result holds in both of the frameworks cited above.

This graph will consist of a number of components, in the usual graph theoretic sense. If the language is substitutable, then every member of the same component will be syntactically congruent, and can thus be freely swapped with each other without altering language membership. In general, there may be more than one component corresponding to the same congruence class, since we are deriving the graph from a small finite sample.

First, note that since syntactic congruence is transitive, and we are interested in substitutable languages, we can compute the transitive closure of the graph, by adding any edges (u,w) when we have edges $(u,v),(v,w)$. We will write $\cong_S$ for the transitive closure of $\doteq_S$. If S is a subset of a substitutable language L then $u \cong_S v$ implies $u \equiv_L v$.

We can write $SG/\cong_S$ for the set of components of the substitution graph and $\lceil u \rceil_S$ for the component that contains the string u. We will normally omit the subscript where there is no risk of confusion. Recall that we write $[u]_L$ for the congruence class of u with respect to the syntactic congruence of L.

3.1 Constructing the Grammar

Algorithm 1: Algorithm to generate a grammar from a substitution graph.

```
Data: A substitution graph $SG = (V,E)$
Result: A context-free grammar $\hat{G}$
Let $\Sigma$ be the set of all the letters used in the nodes of $SG$ ;
Compute $\hat{V}$ the set of components of $SG$ ;
Store map $V \to \hat{V}$ defined by $u \mapsto \lceil u \rceil$ ;
Let $\hat{S}$ be the unique element of $V$ corresponding to the context $(\lambda,\lambda)$. ;
$\hat{P} = \{\}$ ;
for $u \in V$ do
    if $|u| > 1$ then
        for $v,w$ such that $u = vw$ do
            $\hat{P} \leftarrow \hat{P} \cup (\lceil u \rceil \to \lceil v \rceil \lceil w \rceil)$ ;
        end
    else
        $\hat{P} \leftarrow \hat{P} \cup (\lceil u \rceil \to u)$
    end
end
output $\hat{G} = \langle \Sigma, \hat{V}, \hat{S}, \hat{P} \rangle$ ;
```

Given the SG we now construct a grammar $\hat{G} = \langle \Sigma, \hat{V}, \hat{P}, \hat{S} \rangle$.

We define the set of non-terminals to be the set of components of the substitution graph, $\hat{V} = SG/\cong_S$. First note that there will be precisely one component of the substitution graph that will contain all the strings in the sample S. This is because they will all appear in the empty context (λ,λ). This component is defined to be $\hat{S}$.

We now define the set of productions for the grammar. These consist of two types. First for every letter in the alphabet, that occurs as a substring in the language, we have a production.

$$\lceil a \rceil \to a$$

Note that if we have two letters such that $a \doteq b$, then $\lceil a \rceil = \lceil b \rceil$ and the same non-terminal will have two productions rewriting it.

The second set of productions is defined for every substring of length greater than 1. For every node in the substitution graph u, if $|u| > 1$, for every pair of non-empty strings v, w such that $u = vw$, we add a production $\lceil u \rceil \to \lceil v \rceil \lceil w \rceil$. Again note that if the component has more than one node in it, then all of the productions will have the same left hand side.

This is the most important step in the algorithm. It is worth pausing here to consider this construction informally: we shall prove these results formally below. Here we are taking an operation in the syntactic monoid; and constructing a production directly from it. Given some substrings such that $[u] = [v][w]$, we can consider this as saying, that any element of the congruence class $[v]$ concatenated with any element of the congruence class $[w]$ will give an element of the congruence class $[u]$. Phrased like this, it is clear that this can be directly considered as a production rule: if we wish to generate an element of $[u]$ one way of doing it is to generate a string from $[v]$ and then a string from $[w]$.

We can define the set of productions formally as:

$$\hat{P} = \{\lceil u \rceil \to \lceil v \rceil \lceil w \rceil : u = vw, u \in V\ of\ SG, |v| > 0, |w| > 0\} \cup \{\lceil a \rceil \to a : a \in \Sigma\}.$$

Algorithm 1 defines the process of generating a grammar from a substitution graph. To be fully explicit about the global algorithm, we show it in Algorithm 2, rather than relying on the characteristic set.

Algorithm 2: SGL algorithm

```
Data: A sequence of strings $s_1, s_2, \ldots$
Result: A sequence of CFGs $G_1, G_2, \ldots$
$G$ = Grammar generating the empty language ;
while true do
    read next string $s_n$;
    if $s_n \notin \mathcal{L}(G)$ then
        set $SG$ to be the substitution graph generated from $\{s_1, \ldots s_n\}$;
        set $G$ to be the grammar generated from $SG$;
    end
    output $G$;
end
```

3.2 Examples

To clarify the behaviour of the algorithm, we shall now give two examples of its execution. The first one is a step-by-step description on a (very) small sample. The second one allows a discussion about the induced grammar for a particular language.

Example 3 *Suppose the sample consists of the two strings $S = \{a, aa\}$. $Sub(S) = \{a, aa\}$. It is clear that $a \doteq_S aa$. Therefore there is only one component in the substitution graph, associated with the non-terminal $\hat{S}$. The grammar will thus have productions $\lceil aa \rceil \to \lceil a \rceil \lceil a \rceil$ which is $\hat{S} \to \hat{S}\hat{S}$ and $\lceil a \rceil \to a$ which is $\hat{S} \to a$. Thus the learned grammar will be $\langle \{a\}, \{\hat{S}\}, \{\hat{S} \to \hat{S}\hat{S}, \hat{S} \to a\}, \hat{S} \rangle$ which generates the language a^+.*

Example 4 *Consider the language $L = \{a^n cb^n : n \geq 0\}$ that can be represented, for instance, by the set of productions $\{S \to aSb, S \to c\}$. Suppose we have a large sample of strings from this language. The substitution graph will have components $C_i \subset \{a^m cb^{m+i} : m, m+i \geq 0\}$ for both positive and negative values of i, with $C_0 = \hat{S} \subset \{a^m cb^m : m \geq 0\}$ being the sentence symbol. We also have components made of a unique string: $A_i = \{a^i\}$ and $B_i = \{b^i\}$, for positive values of i. The grammar generated from this sample will then have rules of the form*

$$\begin{aligned} C_{j+k} &\to C_j B_k \text{ for all } k > 0,\ j \in \mathbb{Z}\ , \\ C_{j-k} &\to A_k C_j \text{ for all } k > 0,\ j \in \mathbb{Z}, \\ C_0 &\to c, \\ A_{i+j} &\to A_i A_j \text{ for all } i, j > 0, \\ A_1 &\to a, \\ B_{i+j} &\to B_i B_j \text{ for all } i, j > 0, \\ B_1 &\to b. \end{aligned}$$

This grammar defines the language L, but as can be seen the set of non-terminals can be substantially larger than that of the original grammar.

3.3 Polynomial Time

We now show that SGL runs in a time bounded by a polynomial in the total size of the sample. Suppose the sample is $S = \{w_1, \ldots, w_n\}$. We can define $N = \sum |w_i|$, and $L = \max |w_i|$. Clearly $L \leq N$, and $n \leq N$. The total number of substrings, and thus nodes in the graph, is less than N^2. The cost of computing, for a given pair of strings u,v, all of the substrings u', v' such that $u' \doteq_S v'$ can be done in time less than L^2, and thus assuming a constant time map from substrings to nodes in the graph, we can compute all the edges in the graph in time less than $L^2 n^2$. Computing the transitive closure of $\doteq$ or equivalently identifying the components of the substitution graph, can be done in time linear in the sum of the number of nodes and edges which are both polynomially bounded. When constructing the grammar, the number of rules defined by each component/non-terminal is clearly bounded by the number of different ways of splitting the strings in the component, and thus the total number of rules must be bounded by LN^2, and each rule can be constructed in constant time.

There are clearly much more efficient algorithms that could be used: hashing from contexts to components and using a union-find algorithm to identify the components, for example.

4. Proof

Our main result is as follows:

Theorem 9 *SGL polynomially identifies in the limit the class of substitutable (context-free) languages.*

We shall proceed in two steps. First, we shall show that for any sample, the language defined by the grammar inferred from the sample will be a subset of the target language; another way of saying this is that the learner will never make a false positive error. The second step is to show that there is a characteristic set; so that whenever the sample contains this characteristic set, the hypothesis will be correct. This set will have a cardinality which is linear in the size of the representation; however the size of the characteristic set (the sum of the lengths of all the strings in the set) will not necessarily be polynomially bounded. Indeed this is inevitable as the problem class contains examples where *every string* is exponentially large.

4.1 Proof that Pypothesis is Not Too Large

First of all we shall show that, for any finite sample of a substitutable language, the grammar derived from a finite sample does not define a language that is too large.

We start by proving a basic lemma, which is intuitively clear from the definitions.

Lemma 10 *For any sample C, not necessarily including a characteristic set, if $w \in Sub(C)$ then $\lceil w \rceil \overset{*}{\Rightarrow}_{\hat{G}} w$.*

Proof The proof can be done by induction on the length of the string. If $|w| = 1$, then by the construction of the grammar there is a unary rule $\lceil w \rceil \to w$. If $|w| = k+1$, we can write $w = ua$, where $|u| = k, a \in \Sigma$. By the inductive hypothesis, $\lceil u \rceil \overset{*}{\Rightarrow}_{\hat{G}} u$. By the construction of the grammar, there will be a production $\lceil w \rceil \to \lceil u \rceil \lceil a \rceil$, which establishes the lemma. Therefore the generated grammar will always include the training sample. ■

The next lemma states that derivation with respect to $\hat{G}$ maintains syntactic congruence.

Lemma 11 *For any sample C, for all $x \in \Sigma^*$, for all $u \in Sub(C)$, $\lceil u \rceil \overset{*}{\Rightarrow}_{\hat{G}} x$ implies $u \equiv_L x$*

Proof As $u \equiv_L u$ is trivial, we can restrict ourselves to the case $u \neq x$.

By induction on the length of the derivations k. Base step: $k = 1$. This means the derivation must be a single production of the form $\lceil u \rceil \to x$. This will only be the case if $|x| = 1$ and x is in the same component as u; therefore $u \equiv_L x$.

Inductive step: suppose this is true for all derivations of length less than k. Suppose we have a derivation of length $k > 1$. By construction of the grammar, there exist $\lceil v \rceil$ and $\lceil w \rceil$ such that $\lceil u \rceil \Rightarrow \lceil v \rceil \lceil w \rceil \Rightarrow^{k-1} x$. There must be strings x_1, x_2 such that $x = x_1 x_2$ and $\lceil v \rceil \overset{*}{\Rightarrow}_{\hat{G}} x_1$ and $\lceil w \rceil \overset{*}{\Rightarrow}_{\hat{G}} x_2$ with derivations of length less than k. Therefore by the inductive hypothesis, $v \equiv_L x_1$ and $w \equiv_L x_2$. Since we have a production $\lceil u \rceil \to \lceil v \rceil \lceil w \rceil$ in $\hat{P}$, there must be strings v', w' such that $v'w'$ is a string in the same component as u, and $v' \equiv_L v$ and $w' \equiv_L w$ and $u \equiv_L v'w'$. Since $\equiv_L$ is a monoid congruence, we have $u \equiv_L v'w' \equiv_L vw' \equiv_L vw \equiv_L x_1 w \equiv_L x_1 x_2 = x$. ■

Theorem 12 *For any positive sample of a substitutable context-free language L, if $\hat{G}$ is the result of applying the algorithm to it then $\mathcal{L}(\hat{G}) \subseteq L$.*

Proof Let $C \subset L$ be the sample. If $u \in L(\hat{G})$ then there must be some $v \in C$, such that $\hat{S} = \lceil v \rceil \overset{*}{\Rightarrow}_{\hat{G}} u$. Therefore $u \equiv_L v$, which implies $u \in L$. ■

4.2 Proof that Hypothesis is Large Enough

To prove that the hypothesis is large enough, we first need to define a characteristic set, that is to say a subset of a target language L_* which will ensure that the algorithm will output a grammar G such that $\mathcal{L}(G) = L_*$.

4.2.1 CONSTRUCTION OF THE CHARACTERISTIC SAMPLE

Let $G_* = \langle V, \Sigma, P, S \rangle$ be a target grammar. We will assume without loss of generality, that G_* is reduced, that is to say for every non-terminal $N \in V$, there are strings $l, u, r \in \Sigma^*$ such that $S \overset{*}{\Rightarrow} lNr \overset{*}{\Rightarrow} lur$. We are going to define a set CS of words of L_*, such that the algorithm SGL will identify L_* from any superset of CS.

We define $w(\alpha) \in \Sigma^+$ to be the smallest word, according to $\prec$, generated by $\alpha \in (\Sigma \cup V)^+$. Thus in particular for any word $u \in \Sigma^+$, $w(u) = u$. For each non-terminal $N \in V$ define $c(N)$ to be the smallest pair of terminal strings (l, r) (extending $\prec$ from Σ^* to $\Sigma^* \times \Sigma^*$, in some way), such that $S \overset{*}{\Rightarrow} lNr$.

We can now define the characteristic set $CS(G_*) = \{lwr : (N \rightarrow \alpha) \in P, (l,r) = c(N), w = w(\alpha)\}$. The cardinality of this set is at most $|P|$ which is clearly polynomially bounded. In general the cardinality of the set will not polynomially bound the size of the sample, for reasons discussed above in Section 2.1. However, notice that if there exists a polynomial-sized *structurally complete sample* (that is to say a sample where there is at least a string that uses each production rule Dupont et al., 1994) then the size of that characteristic set is polynomial.

Example 5 *Let $G = \langle \{a,b,c\}, \{S\}, \{S \rightarrow aSb, S \rightarrow c\}, S \rangle$ be the target grammar. It generates the language $\{a^n cb^n : n \geq 0\}$. For the rule $S \rightarrow c$, the construction gives $C(S) = (\lambda, \lambda)$ and $w(c) = c$, so the string c belongs to the characteristic set. Concerning the rule $S \rightarrow aSb$, we have $C(S) = (\lambda, \lambda)$ and $w(aSb) = acb$. The characteristic set is then $CS(G) = \{c, acb\}$.*

4.2.2 CONVERGENCE

We now must show that for any substitutable context-free grammar G, if the sample C contains the characteristic set $CS(G)$, that is, $CS(G) \subseteq C \subseteq \mathcal{L}(G)$, and if $\hat{G}$ is the output SGL produces on the sample C, then $\mathcal{L}(\hat{G}) = \mathcal{L}(G)$.

Lemma 13 *For any sample C containing the characteristic set, if $(N \rightarrow \alpha) \in P$, where $\alpha = V_1 \ldots V_l$, then for any $0 < i < j \leq l$, $\lceil w(V_i) \ldots w(V_j) \rceil \overset{*}{\Rightarrow}_{\hat{G}} \lceil w(V_i) \rceil \ldots \lceil w(V_j) \rceil$.*

Proof By construction of the characteristic set, if $(N \rightarrow \alpha) \in P$ then $w(\alpha) = w(V_1) \ldots w(V_l)$ is going to appear as a substring in the characteristic sample. Each production in the derivation $\lceil w(V_i) \ldots w(V_j) \rceil \Rightarrow_{\hat{G}} \lceil w(V_i) \ldots w(V_{j-1}) \rceil \lceil w(V_j) \rceil \cdots \Rightarrow_{\hat{G}} \lceil w(V_i) \rceil \ldots \lceil w(V_j) \rceil$ will be in the set of productions by the construction of the grammar. ■

Lemma 14 *For all $N \in V$ if $N \overset{*}{\Rightarrow}_G u$ then $\lceil w(N) \rceil \overset{*}{\Rightarrow}_{\hat{G}} u$.*

Proof By induction on the length of the derivation. Suppose the derivation is of length 1. Then $N \Rightarrow_G u$ and we must have $N \rightarrow u \in P$. Therefore by the construction of the characteristic set,

$w(N) \in Sub(C)$, $u \in Sub(C)$, and $\lceil u \rceil = \lceil w(N) \rceil$. By Lemma 10, $\lceil w(N) \rceil = \lceil u \rceil \overset{*}{\Rightarrow}_{\hat{G}} u$. This establishes the base case. Suppose this is true for all derivations of length at most k. Suppose $N \Rightarrow_G^{k+1} u$. So we take the first production $N \Rightarrow_G \alpha \Rightarrow_G^k u$. Suppose $|\alpha| = l$. Then we write $\alpha = V_1 \dots V_l \overset{*}{\Rightarrow}_G u_1 \dots u_l = u$, where $V_i \in V \cup \Sigma$ and $V_i \overset{*}{\Rightarrow}_G u_i$. if $V_i \in V$, then by the induction hypothesis $\lceil w(V_i) \rceil \overset{*}{\Rightarrow}_{\hat{G}} u_i$. If $V_i \in \Sigma$, then we have $\lceil w(V_i) \rceil \overset{*}{\Rightarrow}_{\hat{G}} u_i$ thanks to Lemma 10. By the construction of the characteristic set, we have $\lceil w(N) \rceil = \lceil w(\alpha) \rceil$. By Lemma 13 we have $\lceil w(N) \rceil \overset{*}{\Rightarrow}_{\hat{G}} \lceil w(V_1) \rceil \dots \lceil w(V_l) \rceil \overset{*}{\Rightarrow}_{\hat{G}} u_1 \dots u_l = u$. ■

Theorem 15 *For any substitutable CFG G, when the sample contains the characteristic set,* $L \subseteq \mathcal{L}(\hat{G})$.

This follows immediately by applying the previous theorem to S: if $S \overset{*}{\Rightarrow}_G u$ then $\lceil w(S) \rceil \overset{*}{\Rightarrow}_{\hat{G}} u$. Combining this with Theorem 12 establishes that $\mathcal{L}(\hat{G}) = \mathcal{L}(G)$, and therefore Theorem 9 holds.

5. Reduction System

As described up to now, our focus has been on producing an algorithm for which it is easy to prove some correctness results. However, the algorithm is not practical, since the number of non-terminals will often become very large, since in the worst case the algorithm will have one non-terminal for each substring in the training data. Many of these substrings will of course be redundant, and can be removed without reducing the language. In this section we will consider some algorithms for doing this; the languages defined will be unchanged, but the algorithms will be more efficient.

There are a number of algorithms for reducing the number of non-terminals. Clearly one can recursively remove all non-terminals that only have one production by replacing the non-terminal on the left hand side of the production with the right hand side, wherever it occurs. Secondly, one can remove non-terminals, one by one, and test whether the grammar continues to accept all of the sample, and thus arrive at a minimal CFG. Although the cost of that last operation is considerable, it obviously is polynomial.

In this section we describe a variant algorithm that is efficient and practical for large data sets, but that produces a reduction system, rather than a grammar. The idea is that if two substrings appear in the same component of the substitution graph, then we can rewrite every occurrence of the bigger one into the smaller.

The key point here is to reduce the substitution graph, by removing strings that are potentially redundant. In particular if we have one component that contains the strings u and v, where $v \prec u$, and another that contains the strings lur and lvr, we can reduce the graph by removing the string lur. This is equivalent to reducing the reduction system associated with the graph. Indeed, if we know we can rewrite u into v, then there is no need to add a rule that rewrites lur into lvr.

5.1 Definitions

We will briefly describe semi-Thue systems or reduction systems (Book and Otto, 1993).

Definition 16 (Reduction system) *A reduction system T, over an alphabet Σ is a finite set of pairs of strings $T \subset \Sigma^* \times \Sigma^*$, where each pair (u,v) is normally written $u \vdash_T v$, is called a reduction rule and satisfies $v \prec u$.*[2]

By extension, we will denote $lur \vdash lvr$ when $u \vdash v \in T$. $\vdash^*$ is the reflexive and transitive closure of $\vdash$.

Definition 17 (Confluent and weakly confluent reduction system) *A reduction system T is*

- *confluent if and only if for all $w, w1, w2 \in \Sigma^*$ such that $w \vdash w1$ and $w \vdash w2$, there exists $e \in \Sigma^*$ such that $w1 \vdash e$ and $w2 \vdash e$.*
- *weakly confluent on a set S if and only if for all $w, w1, w2 \in S$ such that $w \vdash w1$ and $w \vdash w2$, there exists $e \in S$ such that $w1 \vdash^* e$ and $w2 \vdash^* e$.*

Note that as defined these reduction systems are *Noetherian* which means that there is no infinite sequence of reductions. This defines a congruence relation where u and v are congruent if and only if they can be reduced to the same element. Being confluent and Noetherian means that there is a simple algorithm to determine this congruence: each string belong to only one congruence class. If we have the strict requirement that the reductions must be length reducing ($|v| < |u|$), then the maximum number of reductions is the length of the string you start with. Since we have a looser definition ($v \prec u$), this number can be exponential.

Given a reduction system one can define a language as the union of finitely many congruence classes. Thus given a reduction system T and a set of irreducible strings A on an alphabet Σ, we can define a language $L(T,A) = \{v \in \Sigma^* : \exists a \in A \wedge v \vdash_T^* a\}$. These are the congruential languages. In some cases, this is a more natural way of defining the structure of a language than systems from the traditional Chomsky hierarchy.

For example consider the reduction system $T = \{(aca, c), (bcb, c)\}$, and the axiom c (i.e., we are looking at the congruence class of c). The language defined by $L(T, \{c\})$ is exactly the palindrome language over a, b with center marker c.

The next subsection deals with the modifications of the algorithm SGL allowed by the use of reduction systems instead of grammars.

5.2 Reduction of a Substitution Graph

Given a substitution graph $SG = \langle V, E\rangle$, we say that SG reduces to $SG' = \langle V', E'\rangle$ if and only if there exists $(u,v) \in E : v \prec u$, and (l,r), $|l| + |r| > 0$, such that $lur \in V$, $V' = (V \setminus \{lur\}) \cup \{lvr\}$, $E' = \{(x,y) \in V' \times V' : (x,y) \in E \vee ((lur, y) \in E \wedge x = lvr)\}$.

We say that a substitution graph SG is *irreducible* if there exists no other substitution graph SG' such that SG reduces to SG'.

Given this reduced graph, we define a reduction system directly from the graph.

In this case we will define the set of reductions T to be exactly the set of all pairs $u \vdash v$, where $v \prec u$ and u, v are nodes in the same component of the substitution graph. We can also limit v to be the unique least node (with respect to $\prec$) in each component.

Assuming that we have a set of examples generated from a substitutable CFG that contains the characteristic set, it is easy to prove the following lemmas.

2. This differs slightly from the standard definition which requires $|v| < |u|$.

Lemma 18 *If $N \in V$ and $N \overset{*}{\Rightarrow} u$ for $u \in \Sigma^*$, then $u \vdash^* w(N)$.*

Proof Suppose $N = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_n = u$ is a derivation of u. Map this to a sequence $(w(N), w(\alpha_1), \ldots, w(\alpha_n), u)$ of strings from Σ^*. Consider a single step $\alpha_i = lMr$ and $\alpha_{i+1} = l\beta r$ and there is a production $M \rightarrow \beta$ in P. We have $w(\alpha_i) = w(l)w(M)w(r)$ and $w(\alpha_{i+1}) = w(l)w(\beta)w(r)$, which implies $w(\alpha_{i+1}) \vdash_T^* w(\alpha_i)$ by construction. Therefore $u \vdash^* w(N)$. ■

Lemma 19 *If $u \vdash v$ then $u \in L$ iff $v \in L$.*

Proof $u \vdash v$ implies $\exists (x,y) \in T$ and $l, r \in \Sigma^*$ such that $u = lxr$ and $v = lyr$. $x \doteq_S y$ implies $x \doteq_L y$ implies $x \equiv_L y$ implies $lxr \in L$ iff $lyr \in L$. ■

The reduction system will be weakly confluent on L, and it is Noetherian, since the number of strings smaller (`w.r.t.` $\prec$) than a given string is clearly finite. Unfortunately in general we will not be able to compute an irreducible string for any given word u in a polynomial (in the size of u) number of reductions (see Lohrey, 2000, for a formal proof). Thus though the reduction system itself may be much smaller, in some cases the "parsing" algorithm, determining whether a word is in the language, may be exponential. Subject to this caveat, we can define a small reduction system that represents the same language, namely the set of all strings that reduces to the least string $w(S)$ (`w.r.t.` $\prec$) in the language, but without the large number of redundant rules that the simple grammar construction produces. Without positing further restrictions on the sample set, it is not possible to give precise bounds on the relative sizes of the reduction system and the grammar.

6. Substitutable Languages

This section contains some examples of substitutable CFLs, as well as some simple CFLs that are not substitutable, and a discussion on the relationship of this class of languages to other standard classes. This is without a doubt a restricted class of languages but contains some interesting examples. They are not closed under any standard operation except reversal.

Since we are learning under a Gold style paradigm, we cannot hope to learn all finite languages (Gold, 1967). Indeed, the more complex the languages we hope to learn, the smaller the set of finite languages we will be able to learn.

6.1 Examples

We will now give some examples of some simple languages that are substitutable as well as some simple languages that are not. In general it will not be possible to decide for any given context-free language if it is substitutable. However, it can be checked "by hand" on particular examples.

Given an alphabet Σ, the following languages are substitutable:

- The set Σ^* of all the strings on the alphabet is substitutable. Indeed, all substrings appear in all possible contexts.

- Any language consisting of only one string (namely a singleton language) is substitutable, since in that case $u \doteq v$ implies $u = v$.

- The languages $\{a^n : n > 0\}$, for all $a \in \Sigma$, are substitutable.
- A more complex language, $\{wcw^R : w \in (a,b)^*\}$ (the language of palindromes with center marker) is substitutable.

We now turn our attention to simple languages that are not substitutable.

First, the finite language $L = \{a, aa\}$ is *not* substitutable. Indeed, $a \doteq_L aa$, since they appear in the context (λ,λ), but they are not congruent since (a,λ) is a context of a but not aa. This (counter) example shows that the class is not superfinite and is thus part of the explanation of the positive result proven above. The algorithm presented here would return the hypothesis $\{a^n : n > 0\}$.

The well-known context-free language $\{a^n b^n : n > 0\}$ is also *not* substitutable. Indeed, $a \doteq aab$, because they share the context (a, bb), but they are clearly not syntactically congruent, since $(a, abbb)$ is a context of a but not of aab. However, the language $\{a^n cb^n : n > 0\}$ is substitutable. Here the addition of a center marker removes the problem.

6.2 Relation to Other Language Classes

The state of the art concerning polynomial identification of context-free languages from positive example only is quite limited. In addition, as all studied subclasses are defined with respect to a particular class of representations, the comparison with our purely syntactically defined class is a hard task.

The fact that substitutable context-free languages can be represented by reduction systems proves that they are properly included within the class of congruential languages (Book and Otto, 1993). The examples previously presented show that they are incomparable with the classes of finite languages and regular languages.

The most important subclass of context-free languages that is polynomially identifiable is that of very simple grammars (Yokomori, 2003). It consists of CFGs in Greibach normal form such that no terminal symbol is used in more than one production. Some very simple grammars are not substitutable: an example is the regular grammar with productions $S \to aNP, S \to bN, N \to n, P \to rP, P \to p$. This generates the language $anr^*p \cup bn = \{bn, anp, anrp, anrrp, \ldots\}$. We can see that $n \doteq nr$ but it is not the case that $n \equiv nr$, since bn is in the language but bnr is not.

On the other hand, some substitutable languages, as for instance $\{wcw^R : w \in (a,b)^*\}$, are not very simple.

Finally, we also note the relationship to NTS grammars (Sénizergues, 1985); which can be seen to be relevant in Section 8. NTS grammars have the property that if $N \overset{*}{\Rightarrow} v$ and $M \overset{*}{\Rightarrow} uvw$ then $M \overset{*}{\Rightarrow} uNw$. We conjecture that all substitutable context free languages are NTS languages.

7. Practical Experiment

We will now present a simple experiment that demonstrates the applicability of this algorithm to the learning of natural languages.

For some years, a particular set of examples has been used to provide support for nativist theories of first language acquisition (FLA). These examples, which hinge around auxiliary inversion in the formation of questions in English, have been considered to provide a strong argument in favour of the nativist claim: that FLA proceeds primarily through innately specified domain specific mechanisms or knowledge, rather than through the operation of general-purpose cognitive mechanisms.

A key point of empirical debate is the frequency of occurrence of the forms in question. If these are vanishingly rare, or non-existent in the primary linguistic data, and yet children acquire the construction in question, then the hypothesis that they have innate knowledge would be supported. But this rests on the assumption that examples of that specific construction are necessary for learning to proceed. In this paper we show that this assumption is false: that this particular construction can be learned without the learner being exposed to any examples of that particular type. Our demonstration is primarily mathematical/computational: we present a simple experiment that demonstrates the applicability of this approach to this particular problem neatly, but the data we use is not intended to be a realistic representation of the primary linguistic data, nor is the particular algorithm we use suitable for large scale grammar induction. We will present the dispute in traditional terms, though later we shall analyse some of the assumptions implicit in this description. In English, polar interrogatives (yes/no questions) are formed by fronting an auxiliary, and adding a dummy auxiliary "do" if the main verb is not an auxiliary. For example,

Example 1a The man is hungry.

Example 1b Is the man hungry?

When the subject NP has a relative clause that also contains an auxiliary, the auxiliary that is moved is not the auxiliary in the relative clause, but the one in the main (matrix) clause.

Example 2a The man who is eating is hungry.

Example 2b Is the man who is eating hungry?

An alternative rule would be to move the first occurring auxiliary, that is, the one in the relative clause, which would produce the form

Example 2c Is the man who eating is hungry?

In some sense, there is no reason that children should favour the correct rule, rather than the incorrect one, since they are both of similar complexity and so on. Yet children do in fact, when provided with the appropriate context, produce sentences of the form of Example 2b, and rarely if ever produce errors of the form Example 2c (Crain and Nakayama, 1987). The problem is how to account for this phenomenon.

Chomsky claimed first, that sentences of the type in Example 2b are vanishingly rare in the linguistic environment that children are exposed to, yet when tested they unfailingly produce the correct form rather than the incorrect Example 2c. This is put forward as strong evidence in favour of innately specified language specific knowledge: we shall refer to this view as linguistic nativism.

In a special volume of the Linguistic Review, Pullum and Scholz (2002) showed that in fact sentences of this type are not rare at all. Much discussion ensued on this *empirical* question and the consequences of this in the context of arguments for linguistic nativism. These debates revolved around both the methodology employed in the study, and also the consequences of such claims for nativist theories. It is fair to say that in spite of the strength of Pullum and Scholz's arguments, nativists remained completely unconvinced by the overall argument.

Reali and Christiansen (2004) present a possible solution to this problem. They claim that local statistics, effectively *n*-grams, can be sufficient to indicate to the learner which alternative should be preferred. However this argument has been carefully rebutted by Kam et al. (2005), who show that this argument relies purely on a phonological coincidence in English. This is unsurprising since it is implausible that a flat, finite-state model should be powerful enough to model a phenomenon that is clearly structure dependent in this way.

7.1 Implementation

We have implemented the algorithm described above. There are a number of algorithmic issues that were addressed. First, in order to find which pairs of strings are substitutable, the naive approach would be to compare strings pairwise which would be quadratic in the number of sentences. A more efficient approach maintains a hashtable mapping from contexts to congruence classes. Caching hashcodes, and using a union-find algorithm for merging classes allows an algorithm that is effectively linear in the number of sentences. However, since it is necessary to consider every substring in the sentence, there appears to be a quadratic dependence on sentence length that seems to be hard to remove. Nonetheless this compares favourably to the complexity of context-free parsing.

In order to handle large data sets with thousands of sentences, it was necessary to modify the algorithm in various ways which slightly altered its formal properties. Primarily these involved pruning non-terminals that appeared to be trivial: since in the worst case the number of non-terminals approaches the number of distinct substrings in the corpus this is an enormous improvement. However for the experiments reported here we used a version which performs exactly in line with the mathematical description above.

7.2 Data

For clarity of exposition, we have used extremely small artificial data-sets, consisting only of sentences of types that would indubitably occur in the linguistic experience of a child. We do not attempt in this paper to demonstrate that, with naturally occurring distributions of data, this algorithm will correctly learn this distinction.

Our first experiments were intended to determine whether the algorithm could determine the correct form of a polar question when the noun phrase had a relative clause, even when the algorithm was not exposed to any examples of that sort of sentence. We accordingly prepared a small data set shown in Table 1. Above the line is the training data that the algorithm was trained on. It was then tested on all of the sentences, including the ones below the line. By construction the algorithm would generate all sentences it has already seen, so it scores correctly on those. The learned grammar also correctly generated the correct form and did not generate the final form.

We can see how this happens quite easily since the simple nature of the algorithm allows a straightforward analysis. We can see that in the learned grammar "the man" will be congruent to "the man who is hungry", since there is a pair of sentences which differ only by this. Similarly, "hungry" will be congruent to "ordering dinner". Thus the sentence "is the man hungry ?" which is in the language, will be congruent to the correct sentence.

One of the derivations for this sentence would be: [is the man hungry ?] $\rightarrow$ [is the man hungry] [?] $\rightarrow$ [is the man] [hungry] [?] $\rightarrow$ [is] [the man] [hungry] [?] $\rightarrow$ [is] [the man] [who is hungry] [hungry] [?] $\rightarrow$ [is] [the man] [who is hungry] [ordering dinner] [?].

Our second data set is shown in Table 2, and is a fragment of the English auxiliary system. This has also been claimed to be evidence in favour of nativism. This was discussed in some detail by Pilato and Berwick (1985). Again the algorithm correctly learns: the learned language includes the correct form but not the incorrect form.

the man who is hungry died .
the man ordered dinner .
the man died .
the man is hungry .
is the man hungry ?
the man is ordering dinner .
is the man who is hungry ordering dinner ?
∗is the man who hungry is ordering dinner ?

Table 1: Auxiliary fronting data set. Examples above the line were presented to the algorithm during the training phase, and it was tested on examples below the line.

it rains.
it may rain.
it may have rained.
it may be raining.
it has rained.
it has been raining.
it is raining.
it may have been raining.
∗it may have been rained.
∗it may been have rain.
∗it may have been rain.

Table 2: English auxiliary data. Training data above the line, and testing data below.

7.3 Conclusion of the Experiment

Chomsky was among the first to point out the limitations of Harris's approach, and it is certainly true that the grammars produced from these toy examples overgenerate radically. On more realistic language samples SGL would eventually start to generate even the incorrect forms of polar questions.

Given the solution we propose it is worth looking again and examining why nativists have felt that auxiliary fronting was such an important issue. It appears that there are several different areas. First, the debate has always focused on how to construct the interrogative from the declarative form. The problem has been cast as finding which auxiliary should be "moved". Implicit in this is the assumption that the interrogative structure must be defined with reference to the declarative, one of the central assumptions of traditional transformational grammar. Now, of course, given our knowledge of many different formalisms which can correctly generate these forms without movement we can see that this assumption is false. There is of course a relation between these two sentences, a semantic one, but this does not imply that there need be any particular syntactic relation, and certainly not a "generative" relation.

Secondly, the view of learning algorithms is very narrow. It is considered that only sentences of that exact type could be relevant. We have demonstrated, if nothing else, that that view is false. The distinction can be learned from a set of data that does not include any example of the exact piece of data required: as long as the various parts can be learned separately, the combination will function in the natural way.

8. Discussion

The important step in the algorithm is the realisation that it is not necessary to learn constituent structure in order to learn context-free languages. The rule $\lceil uv \rceil \rightarrow \lceil u \rceil \lceil v \rceil$ appears at first sight vacuous, yet it is sufficient to define correct languages for an interesting class of grammars.

8.1 Related Work

This work is related to two other strands of work. First work that proves polynomial IIL of other subclasses of context-free grammars. Yokomori (2003) shows that the class of very simple languages can be polynomially identified in the limit. Unfortunately the time complexity is $N^{|\Sigma|+1}$ and the alphabet size is equal to the number of productions in a very simple grammar, so this algorithm is not practical for large scale problems. Secondly, we can relate it to the work of Adriaans et al. (2000), who uses a similar heuristic to identify languages. Finally, we can mention the similar work of Starkie (2004) who shows an identification in the limit result of a class of grammars called "left-aligned R grammars". This work defines a rather complicated family of grammars, and shows how constituents can be identified. We also note Laxminarayana and Nagaraja (2003) who show a learnable subclass of CFGs.

We can compare substitutability with reversibility (Angluin, 1982; Makinen, 2000). Recall that a regular language is reversible if whenever uw and vw are in the language then ux is in the language if and only if vx is in the language. Thus reversibility is the exact analogue of substitutability for regular languages. Note that reversibility is a weaker criterion than substitutability. Substitutability implies reversibility, but not vice versa, as can be seen from the finite language $\{ab, bb\}$ which is reversible but not substitutable. Nonetheless, following the work on reversible regular languages, one can think of a definition of k-substitutable languages and may obtain positive learning results on that hierarchy of languages.

We can also compare the substitutability to μ-distinguishability for inference of regular languages (Ron et al., 1998). Ron uses a measure of similarity of residual languages, rather than of contexts as we use here. Considered in this way, our measure is very crude, and brittle—contexts are equal if they have non empty intersection. Nonetheless the techniques of Ron et al., suggest a way that this technique could be extended to a PAC-learning result, using a bound on a statistical property of the distribution. There are some technical problems to be overcome, since the number of syntactic congruence classes will be infinite for non regular languages, and thus the distinguishability will not in general be bounded from below. A more serious problem is that the worst case sample complexity, if the data is drawn randomly, is clearly exponential, since the chance of getting two strings that differ only in a single point is in general exponential in the derivational entropy of the grammar.

Algorithms for learning regular languages focus on identifying the states of a deterministic automaton. When trying to move to learning context-free languages, the obvious way is to try to identify configurations (i.e., pairs of states and strings of stack symbols) of a deterministic push down automaton. A problem here is that the structure of this set depends on the representation, the automaton. One way of viewing the work presented in this paper is to say that a better approach is to try to identify the elements of the syntactic monoid. This monoid represents in the barest form the combinatorial structure of the language. From a learnability point of view this is interesting because it is purely syntactic—it is not semantic as it does not depend on the representation of the language but only on the language itself. Since we are interested in algorithms that learn from unstructured data—strings from the language that are not annotated with structural information—this seems a more natural approach. Importantly, our algorithm does not rely on identifying constituents: that is to say on identifying which substrings have been generated by the non-terminals of the target *grammar*. This has up to now been considered the central problem in context-free grammatical inference, though it is in some sense an ill-posed problem since there may be many different grammars with different constituent structure that are nonetheless equivalent, in the sense that they define the same language.

One of the weaknesses in the work is the fact that we do not yet have a grammatical characterisation of substitutability, nor an algorithm for determining whether a grammar defines a substitutable language. It is clear from standard results in the field that this property will be undecidable in general, but it might be possible to decide it for NTS grammars (Sénizergues, 1985).

Looking at our approach more generally, it is based on identifying syntactically congruent substrings. Substitutable languages have a property that allows a trivial procedure for determining when two substrings are congruent, but it is easy to think of much more robust methods of determining this that rely on more complex properties of the context distributions. Thus in principle we can use any property of the sample from the context distribution: average length, substring counts, marginal distributions at certain offsets and so on.

To conclude, we have shown how a simple formalization of Harris's substitutability criterion can be used to polynomially learn an interesting subclass of context-free languages.

Acknowledgments

This work has benefited from the support of the EU funded PASCAL Network of Excellence on Pattern Analysis, Statistical Modelling and Computational Learning. We would like to thank Colin de la Higuera, Jean-Christophe Janodet, Géraud Sénizergues, Brad Starkie and the two anonymous reviewers for helpful comments and discussions.

References

P. Adriaans. Learning shallow context-free languages under simple distributions. In K. Vermeulen and A. Copestake, editors, *Algebras, Diagrams and Decisions in Language, Logic and Computation*, volume 127. CSLI Publications, 2002.

P. Adriaans, M. Trautwein, and M. Vervoort. Towards high speed grammar induction on large text corpora. In *SOFSEM 2000*, pages 173–186. Springer Verlag, 2000.

D. Angluin. Inference of reversible languages. *Communications of the ACM*, 29:741–765, 1982.

R. Book and F. Otto. *String Rewriting Systems*. Springer Verlag, 1993.

N. Chomsky. Systems of syntactic analysis. *Journal of Symbolic Logic*, 18(3):242–256, 1953.

S. Crain and M. Nakayama. Structure dependence in grammar formation. *Language*, 63(3):522–543, 1987.

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

P. Dupont, L. Miclet, and E. Vidal. What is the search space of the regular inference? In R. C. Carrasco and J. Oncina, editors, *Grammatical Inference and Applications, Second International Colloquium, ICGI-94*, pages 25–37, 1994.

E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447 – 474, 1967.

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

X. N. C. Kam, I. Stoyneshka, L. Tornyova, J. D. Fodor, and W. G. Sakas. Non-robustness of syntax acquisition from n-grams: A cross-linguistic perspective. In *The 18th Annual CUNY Sentence Processing Conference*, April 2005.

J. A. Laxminarayana and G. Nagaraja. Inference of a subclass of context free grammars using positive samples. In *Proceedings of the Workshop on Learning Context-Free Grammars at ECML/PKDD 2003*, 2003.

M. Lohrey. Word problems and confluence problems for restricted semi-thue systems. In *RTA*, volume 1833 of *LNCS*, pages 172–186. Springer, 2000.

E. Makinen. On inferring zero-reversible languages. *Acta Cybernetica*, 14:479–484, 2000.

S. Pilato and R. Berwick. Reversible automata and induction of the english auxiliary system. In *Proceedings of the ACL*, pages 70–75, 1985.

L. Pitt. Inductive inference, DFA's, and computational complexity. In *AII '89: Proceedings of the International Workshop on Analogical and Inductive Inference*, pages 18–44. Springer-Verlag, 1989.

G. Pullum and B. Scholz. Empirical assessment of stimulus poverty arguments. *The Linguistic Review*, 19(1-2):9–50, 2002.

Florencia Reali and Morten H. Christiansen. Structure dependence in language acquisition: Uncovering the statistical richness of the stimulus. In *Proceedings of the 26th Annual Conference of the Cognitive Science Society*, Mahwah, NJ, 2004. Lawrence Erlbaum.

D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic finite automata. *J. Comput. Syst. Sci.*, 56(2):133–152, 1998.

G. Sénizergues. The equivalence and inclusion problems for NTS languages. *J. Comput. Syst. Sci.*, 31(3):303–331, 1985.

Z. Solan, D. Horn, E. Ruppin, and S. Edelman. Unsupervised context sensitive language acquisition from a large corpus. In *Proceedings of NIPS 2003*, 2004.

B. Starkie. *Identifying Languages in the Limit using Alignment Based Learning*. PhD thesis, University of Newcastle, Australia, 2004.

M. van Zaanen. Implementing alignment-based learning. In *Grammatical Inference: Algorithms and Applications, ICGI*, volume 2484 of *LNCS*, pages 312–314. Springer, 2002.

M. Wakatsuki and E. Tomita. A fast algorithm for checking the inclusion for very simple deterministic pushdown automata. *IEICE Trans. on Information and Systems*, E76-D(10):1224–1233, 1993.

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