Identification in the limit of substitutable context-free languages

Alexander Clark[1] and Rémi Eyraud[2]

[1] Department of Computer Science,
Royal Holloway University of London
Egham, Surrey, TW20 0EX
UK
alexc@cs.rhul.ac.uk
http://www.cs.rhul.ac.uk/home/alexc/
[2] EURISE
23, rue du Docteur Paul Michelon
42023 Saint-Étienne Cedex 2
France
remi.eyraud@univ-st-etienne.fr
http://eurise.univ-st-etienne.fr/~eyraud/

Abstract. This paper formalisms 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 grammars. 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. 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.

1 Introduction

Current techniques for grammatical inference have for a long time been focussed 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 [4, 7]. 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 [5]. 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 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 capture the essential language theoretic property that a certain class of algorithms must rely on, and show that this criterion is sufficient to guarantee identification in the limit.

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 nonterminal 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 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.

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. 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 $Sub(L) = \{u \in \Sigma^+ : \exists l, r, lur \in L\}$ (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.

Many classes of languages have been investigated in the literature. In general, the definition of a class $\mathbb{L}$ 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 $G = \langle V, \Sigma, P, S\rangle$ where Σ is a finite alphabet of terminal symbols, V is a 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 $T \to w$.

We will write $uTv \Rightarrow uwv$ when $T \to w \in P$. $\overset{*}{\Rightarrow}$ 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^* i.e.

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

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.

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

Definition 3. *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 $C_L(u) = C_L(v)$.

The weaker idea of substitutability that we will use is defined 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 transitive. 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. *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 set 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.

2.1 Learning

We now define our learning criterion. This is identification in the limit from positive text [6], with polynomial bounds on data and computation, but not on errors of prediction [5].

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 [5].

Definition 6. *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 $\mathcal{L}(R) = \mathcal{L}(R')$.*

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 arc 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 context-free grammar is substitutable, and the sample contains the characteristic set, then SGL will produce a grammar weakly equivalent (ie. that generates the same language) to the target.

Definition 7 (substitution graph).
Given a set of words S, we define the substitution graph $SG(S) = (V, E)$ as follows:
$V = \{u \in \Sigma^+ : \exists l, r \in \Sigma^*, lur \in S\}$
$E = \{(u, v) \in \Sigma^+ \times \Sigma^+ : u \doteq_S v\} = \{(u, v) \in \Sigma^+ \times \Sigma^+ : \exists l, r \in \Sigma^*, lur \in S \wedge lvr \in S\}$

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. Of course 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 subsitutable 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 $[u]_{\cong_S}$ for each element. We will normally omit the subscript where there is no risk of confusion.

3.1 Constructing the grammar

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

We define the set of nonterminals 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 will 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, and we can assume without loss of generality that they occur as substrings in the language, we have a production

$$[a] \to a$$

Note that if we have two letters such that $a \doteq b$, then $[a] = [b]$ and the same nonterminal 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 $u = vw$ add a production $[u] \rightarrow [v][w]$. 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.

We can define the set of productions formally as:

$$\hat{P} = \{[u] \rightarrow [v][w] | u = vw, u \in V \; of \; SG, |v| > 0, |w| > 0\} \cup \{[a] \rightarrow a | a \in \Sigma\}$$

To be explicit about the algorithm, we show it in Algorithm 1, rather than relying on the characteristic set.

Algorithm 1: SGL algorithm

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

3.2 Examples

Example 1: 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 nonterminal $\hat{S}$. The grammar will thus have productions $[aa] \rightarrow [a][a]$ which is $\hat{S} \rightarrow \hat{S}\hat{S}$ and $[a] \rightarrow a$ which is $\hat{S} \rightarrow a$. Thus the learned grammar will generate the language a^+.

Example 2: Consider the language $L = \{a^n c b^n | n \geq 0\}$. Suppose we have a large sample of strings from this language. The substitution graph will have components $C_i \subset \{a^n c b^{n+i} | n \geq 0\}$ for integer values of i, $A_i = \{a^i\}$ and $B_i = \{b^i\}$, for positive values of i, with $\hat{S} = C_0$. The grammar generated from this sample will then have rules of the form (for $i \geq 0$)

$$C_i \rightarrow C_j B_{i-j}$$
$$C_{-i} \rightarrow A_{i-j} C_j$$
$$A_{i+j} \rightarrow A_i A_j, A_1 \rightarrow a$$
$$B_{i+j} \rightarrow B_i B_j, B_1 \rightarrow b$$

Thus, the set of nonterminals can be substantially larger than that of the original grammar.

3.3 Polynomial time

We now show, rather crudely, that SGL runs in a time bounded by a polynomial in the total length 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/nonterminal 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 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

Theorem 1. *SGL polynomially identifies in the limit the class of substitutable (context-free) languages.*

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

Construction of the characteristic sample let $G_* = \langle V, \Sigma, P, S \rangle$ be a target grammar. 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 $\alpha \in (\Sigma \cup V)^+$. For each nonterminal $N \in V$ 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 $S \stackrel{*}{\Rightarrow} lNr$.

We can now define the characteristic set $CS = \{lwr | (N \to \alpha) \in P, (l, r) = c(N), w = w(\alpha)\}$. The cardinality of this set is at most $|P|$ which is clearly polynomially bounded.

Convergence We now must show that for any substitutable context-free grammar G, if $CS(G) \subseteq S \subseteq \mathcal{L}(G)$ then if $\hat{G}$ is the output SGL produces on the sample S, $\mathcal{L}(\hat{G}) = \mathcal{L}(G)$. We will start by showing that the grammar derived from the original substitution graph will define the right grammar.

To do this we will show first the following lemma:

Lemma 1. $\mathcal{L}(G) \subseteq \mathcal{L}(\hat{G})$.

Proof. By the definition of the characteristic set, for each production $N \to \alpha$ in the original grammar, there is a substring $w(N)$ and a substring $w(\alpha)$ in the same component of the substitution graph. If $\alpha = a_1 \dots a_n$ where $a_i \in \Sigma \cup V$, then $w(\alpha) = w(a_1) \dots w(a_n)$[1]. Therefore by the definition of $\hat{P}$, we will have a total of $n-1$ productions, one of the form $[w(\alpha)] \to [w(a_1)][w(a_2) \dots ...w(a_n)]$ and $n-2$ of the form $[w(a_i) \dots w(a_n)] \to [w(a_i)][w(a_{i+1}) \dots w(a_n)]$. These productions suffice to show that $[w(N)] \stackrel{*}{\Rightarrow}_{\hat{G}} [w(a_1)] \dots [w(a_n)]$[2]. Given the definition of the start symbols, and the definition of the non-terminal symbols of the form $[u]$ where $u \in \Sigma$ (i.e. the pre-terminals), this suffices to show that if $S \stackrel{*}{\Rightarrow}_G u$ then $[w(S)] \stackrel{*}{\Rightarrow}_{\hat{G}} u$. Intuitively this is because by the construction of the characteristic set, the set of productions in the hypothesis is going to be a superset of the set of productions in the target, and thus by an induction on the length of the derivation, the defined language will be a superset of the target language. QED.

Next we show that the grammar does not define a language that is too large. Recall that $w(\alpha)$ for some sequence of non terminals and terminals is the smallest (`w.r.t.` $\prec$) string v such that $\alpha \stackrel{*}{\Rightarrow} v$. The basic lemma here is that derivation with respect to $\hat{G}$ maintains syntactic congruence. Note first that by the construction of the grammar: $[u] \stackrel{*}{\Rightarrow}_{\hat{G}} u$.

Lemma 2. *For all $v \in \Sigma^*$, for all $u \in Sub(S)$, $[u] \stackrel{*}{\Rightarrow}_{\hat{G}} v$ implies $u \equiv_L v$*

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

Inductive step: suppose this is true for all derivations of length less than k. Suppose we have a derivation of length $k > 1$. Suppose we have $[u] \Rightarrow [v][w] \stackrel{*}{\Rightarrow}_{\hat{G}} x$. There must be strings l, r such that $x = lr$ and $[v] \stackrel{*}{\Rightarrow}_{\hat{G}} l$ and $[w] \stackrel{*}{\Rightarrow}_{\hat{G}} r$ with derivations of length less than k. Therefore by the inductive hypothesis, $v \equiv_L l$ and $w \equiv_L r$. Since we have a production $[u] \to [v][w]$ 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 $u' \equiv_L u$ and $u \equiv_L v'w'$. Since $\equiv_L$ is a monoid congruence, we have $u \equiv_L v'w' \equiv_L vw' \equiv_L vw \equiv_L lw \equiv_L lr = x$. QED

This lemma suffices to establish that $\mathcal{L}(\hat{G}) \subseteq \mathcal{L}(G)$, since if v is in the sample S, then if $\hat{S} = [v] \stackrel{*}{\Rightarrow}_{\hat{G}} u$, then $u \equiv_L v$ implies $u \in L$. Therefore $\mathcal{L}(\hat{G}) = \mathcal{L}(G)$, and Theorem 1 follows immediately.

[1] This is clear by definition of the partial order $\prec$ above.

[2] This is just the right binarization of the production. We have not made the assumption that the grammar is in Chomsky Normal Form which would remove the need for this step in the proof.

5 Reduction system

As described here the algorithm is not practical, since the number of nonterminals will often become very large. There are a number of algorithms for reducing the number of nonterminals. Clearly one can recursively remove all nonterminals that only have one production by replacing the nonterminal on the left hand side of the production with the right hand side, wherever it occurs. Secondly, one can remove nonterminals, one by one, and test whether the grammar continues to accept all of the sample, and thus arrive at a minimal CFG.

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 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 $u \prec v$ and another that contains the strings lur and lvr, we can reduce the graph by removing the string lvr. This is equivalent to reducing the reduction system associated with the graph.

5.1 Definitions

We will briefly describe semi-Thue systems or reduction systems [3].

Definition 8 (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 $v \prec u$. [3]

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

Definition 9 (Confluent and weakly confluent reduction system).

- *A reduction system T is confluent if and only if for all $w, w1, w2$ such that $w \vdash w1$ and $w \vdash w2$, there exists e such that $w1 \vdash e$ and $w2 \vdash e$.*
- *It is 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 $w2 \vdash^* e$.*

Finally a reduction system is *Noetherian* if 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 set of irreducible strings A, and a reduction system T, we can define a language $L(T, A) = \{v : \exists a \in A \wedge v \vdash_T^* a\}$. These

[3] This differs slightly from the standard definition which requires $|v| < |u|$.

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.

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 to be exactly the set of all pairs $v \vdash u$, where $u \prec v$ and u, v are nodes in the same component of the substitution graph. We can also limit u to be the unique least node (`w.r.t.` $\prec$) 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.

Lemma 3. *If $N \in V$ and $N \stackrel{*}{\Rightarrow} u$ 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 $M \to \beta$ in P. $w(\alpha_i) = w(l)w(M)w(r)$ and $w(\alpha_{i+1}) = w(l)w(\beta)w(r)$ Therefore $w(\alpha_i) \vdash_T^* w(\alpha_{i+1})$. QED.

Lemma 4. *If $v \vdash u$ then $v \in L$ iff $u \in L$*

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

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. 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 an efficient, 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.

6 Substitutable languages

We now give some examples of substitutable CFLs, as well as some simple CFLs that are not substitutable, and discuss 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 [6]. Indeed, the more complex the languages we hope to learn, the smaller the set of finite languages we will we able to learn.

6.1 Examples

- Σ^* is substitutable
- Any language consisting of only one string is substitutable.
- The finite language $\{a, aa\}$ is *not* substitutable. The algorithm presented here would return the hypothesis $\{a^n | n > 0\}$
- $\{a^n | n > 0\}$ is substitutable.
- $\{a^n b^n | n > 0\}$ is *not* substitutable. This is because $a \doteq aab$, but they are clearly not syntactically congruent.
- $\{a^n c b^n | n > 0\}$ is substitutable. Here the addition of a center marker removes the problem.
- $\{wcw^R | w \in (a, b)^*\}$ (the palindrome with center marker) is substitutable.
- Strictly deterministic regular languages [13] are substitutable. Since the automaton is forward and backwards deterministic, and any given string can only be generated by a unique sequence of states, we can see easily that if $u \doteq v$ then the sequence of states that generates u must start and stop in exactly the same state that v starts and stops in.

Recall that very simple grammars [14] consist 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 grammar with productions $S \to bN, S \to aNP, N \to xM, N \to n, P \to rMP, P \to p, M \to m$. This generates the language $bn, bxm, anp, axmp, anrmp, \ldots$ We can see that $x \doteq nr$ but it is not the case that $x \equiv nr$, since bxm is in the language but $bnrm$ is not. Nonetheless we note that the three grammars in [14] Example 2 are all substitutable languages.

We also note the relationship to NTS grammars[11]; which can be seen to be relevant in the next section. NTS grammars have the property that if $N \stackrel{*}{\Rightarrow} v$ and $M \stackrel{*}{\Rightarrow} uvw$ then $M \stackrel{*}{\Rightarrow} uNw$. We conjecture that all substitutable languages are NTS languages.

6.2 Relation to other language classes

Substitutable context-free languages are properly included within the class of congruential languages [3]. They are incomparable with the classes of finite languages, regular languages, and very simple languages. It properly includes the class of strictly deterministic regular languages.

7 Discussion

This work is related to two other strands of work. First work that proves polynomial IIL of other subclasses of context-free grammars. In [14] , Yokomori shows that the class of very simple languages can be polynomially identified in the limit. Unfortunately the 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 [1], who uses a similar heuristic to identify languages. Finally, we can mention the similar work of [12] 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 [8] who show a learnable subclass of CFGs.

We can compare substitutability with reversibility [2, 9]. Recall that a 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 language $\{ab, bb\}$ which is reversible but not substitutable.

We can also compare the substitutability to μ-distinguishability for inference of regular languages [10]. 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 way 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 weakly equivalent, that is to say, 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 [11].

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 is 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.

Acknowledgements

This work has benefitted 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 and Brad Starkie for helpful comments and discussions.

References

1. 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.
2. D. Angluin. Inference of reversible languages. *Communications of the ACM*, 29:741–765, 1982.
3. R. Book and F. Otto. *String rewriting systems*. Springer Verlag, 1993.
4. N. Chomsky. Systems of syntactic analysis. *Journal of Symbolic Logic*, 18(3), 1953.
5. C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27(2):125–138, 1997.
6. E. M. Gold. Language indentification in the limit. *Information and Control*, 10(5):447 – 474, 1967.
7. Z. Harris. Distributional structure. *Word*, 10(2-3):146–62, 1954.
8. 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.

9. E. Mäkinen. On inferring zero-reversible languages. Technical Report A-1998-7, University of Tampere, 1998.
10. D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic finite automata. *Journal of Computer and System Sciences (JCSS)*, 56(2):133–152, 1998.
11. G. Senizergues. The equivalence and inclusion problems for NTS languages. *J. Comput. Syst. Sci.*, 31(3):303–331, 1985.
12. B. Starkie. *Identifying Languages in the Limit using Alignment Based Learning.* PhD thesis, University of Newcastle, Australia, 2004.
13. T. Yokomori. On polynomial-time learnability in the limit of strictly deterministic automata. *Machine Learning*, 19(2):153–179, 1995.
14. T. Yokomori. Polynomial-time identification of very simple grammars from positive data. *Theoretical Computer Science*, 298(1):179–206, 2003.