Journal of Machine Learning Research 1 (2013) 0-0 Submitted 4/00; Published 10/00

Learning Trees from Strings: a Strong Learning Algorithm for some Context-Free Grammars

Alexander Clark ALEXANDER.CLARK@KCL.AC.UK,ALEXC@CS.RHUL.AC.UK
Department of Philosophy,
King's College London,
The Strand,
London WC2R 2LS

Editor: Mehryar Mohri

Abstract

Standard models of language learning are concerned with weak learning: the learner, receiving as input only information about the strings in the language, must learn to generalise and to generate the correct, potentially infinite, set of strings generated by some target grammar. Here we define the corresponding notion of strong learning: the learner, again only receiving strings as input, must learn a grammar that generates the correct set of structures or parse trees. We formalise this using a modification of Gold's identification in the limit model, requiring convergence to a grammar that is isomorphic to the target grammar. We take as our starting point a simple learning algorithm for substitutable context-free languages, based on principles of distributional learning, and modify it so that it will converge to a canonical grammar for each language. We prove a corresponding strong learning result for a subclass of context-free grammars.

Keywords: context-free grammars; grammatical inference; identification in the limit; structure learning

1. Introduction

We present an algorithm for inducing a context-free grammar from a set of strings; this algorithm comes with a strong theoretical guarantee: it works in polynomial time, and for any grammar in a certain class it will converge to a grammar which is isomorphic/strongly equivalent to the target grammar. Moreover the convergence is rapid in a technical sense. This very strong guarantee comes of course at a price: the class of grammars is small. In the first part of the paper we explain the learning model we use which is an extension of the Gold identification in the limit model; and in the second part we present an algorithm which learns a class of languages with respect to this model. We have implemented this algorithm and we present some examples at the end which illustrate the properties of this algorithm, testing on some simple example languages. As far as we are aware this is the first nontrivial algorithm for learning trees from strings which has any sort of theoretical guarantee of its convergence and correctness.

Our ultimate domain of application of these techniques is primarily in linguistics, where the strings will be sequences of words in a natural language, but the techniques can be

applied more broadly to artificial languages, bioinformatics and other fields where the input data consists of strings which have some hierarchical structure.

We can contrast the approach here with the task of unsupervised parsing in computational linguistics as exemplified by Cohn et al. (2010). Unsupervised parsers use a variety of heuristic approaches to extract a single tree for each sentence, taking as input a large natural language corpus, and being evaluated against some linguistically annotated corpus. Here we are interested not in finding the most likely parse, but in finding the set of allowable parses in a theoretically well-founded way.

1.1 Linguistics

The notions of weak and strong generation are fundamental in the fields of mathematical and theoretical linguistics. A formal grammar weakly generates a set of strings, and strongly generates a set of structures (Miller, 1999). We do not have the space for a full discussion of the rather subtle methodological and indeed philosophical issues involved with which model is appropriate for studying linguistics, which questions depend on what the subject matter of linguistics is taken to be; we merely note that while mathematical attention has largely focused on the issues of weak generation, many linguists are more concerned with the issues of strong generation and as a result take the weak results to be largely irrelevant (Berwick et al., 2011). Indeed, taking a grammar as a model of human linguistic competence, we are primarily interested in the set of structures generated. Unfortunately, we have little or no direct evidence about the nature of these structures, notwithstanding recent advances in neuroimaging and psycholinguistics, and our sources of information are essentially only about the set of strings that are weakly generated by the grammar, since these can be observed, and our intuitions about the associated meanings.

We can define corresponding notions of weak and strong learning.[1] Weak learning involves merely learning a grammar that generates the right set of strings; strong learning involves learning a grammar that generates the right set of structures (Wexler and Culicover, 1980, p. 58). Some sentences are ambiguous and will require a grammar that generates more than one structure for a particular sentence. We do not consider in this paper the problem of learning when the input to the learner are *trees*; see for example Sakakibara (1990, 1992); Drewes and Högberg (2003); López et al. (2004). We consider only the problem where the learner has access to the flat strings alone, but must infer an appropriate set of trees for each string in the language. Rather than observing the derivation trees themselves, we observe only the yields of the trees.

Weak learning of context-free grammars and richer formalisms has made significant progress in recent years (Clark and Eyraud, 2007; Yoshinaka, 2011; Yoshinaka and Kanazawa, 2011; Yoshinaka, 2012) but strong learning has received less attention. For CFGs this means that the hypothesis needs to be isomorphic (assuming it is trim) or better yet, *identical* to the target grammar. We define these notions of equivalence and the associated learning models in Section 3. Strong learning is obviously impossible for the full class of context-free grammars since there are an infinite number of structurally different context-free grammars that generate a given context-free language.

1. Note that this has nothing to do with strong and weak learners as those terms are used in the boosting literature in machine learning (Schapire, 1999).

In this paper we work in a categorical model which assumes, unrealistically, a partition of the strings into grammatical and ungrammatical, but probabilistically the situation is not better; given a distribution defined by a probabilistic CFG (PCFG) there are infinitely many structurally different CFGs that define the same set of distributions; in other words PCFGs are not identifiable from strings (Hsu et al., 2013). This is in contrast to discrete HMMs which are (Petrie, 1969).

The contributions of this paper are as follows. We first define an appropriate notion of strong learning from strings, restricting ourselves to the case of CFGs for simplicity. We then show that existing learning algorithms for regular languages (Angluin, 1982) can be viewed as also being strong learning algorithms, in a trivial sense. We then present a strong learning algorithm for some CFGs, based on combining the polynomial algorithm for substitutable context-free languages defined in Clark and Eyraud (2007), which we recall in Section 4, with a recent proposal for a formal notion of syntactic structure (Clark, 2011) that we interpret as a form of canonical grammars. We specify the canonical grammars we target in Section 5, present an algorithm in Section 6, and prove its correctness and efficiency in Section 7. Section 8 contains some examples, including one with an ambiguous grammar. An appendix contains some detailed proofs of various technical lemmas regarding the properties of the languages we consider in this paper.

2. Notation

Let Σ be a finite non-empty set of atomic symbols. Σ^* is the set of all finite strings over Σ. We denote the empty string by λ. The set of non-empty strings is $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. We write $|u|$ for the length of a string u, and for a finite set of strings X we define the size as $\|X\| = \sum_{w \in X} |w|$.

A language L is any subset of Σ^*. Given two languages $M, N \subseteq \Sigma^*$ we write $M \cdot N$ or sometimes just MN for the set $\{uv \mid u \in M, v \in N\}$. Note that this is just the normal concatenation of sets of strings.

Given a language D, we define $\mathrm{Sub}(D)$ to be the set of non-empty substrings of elements of D.

$$\mathrm{Sub}(D) = \{u \in \Sigma^+ \mid \exists (l, r) \in \Sigma^* \times \Sigma^*, lur \in D\}$$

Given a non-zero sequence of languages $\alpha = \langle X_1, \ldots, X_n \rangle$ we write $\bar{\alpha}$ for the concatenation i.e. $\bar{\alpha} = X_1 \cdot \cdots \cdot X_n$. We shall assume an order $\prec$ or $\preceq$ on Σ which we shall extend to the length-lexicographic order on Σ^*.

We define a context (l, r) to be an ordered pair of strings, an element of $\Sigma^* \times \Sigma^*$. The distribution of a string $u \in \Sigma^*$ with respect to a language L is defined to be

$$D_L(u) = \{(l, r) \in \Sigma^* \times \Sigma^* \mid lur \in L\}.$$

We say that $u \equiv_L v$ iff $D_L(u) = D_L(v)$. This is the syntactic congruence, which is equivalent to complete mutual substitutability of u and v.

We write $[u]_L$ for $\{v \in \Sigma^* \mid D_L(u) = D_L(v)\}$. If we have a set of strings, X, that are all congruent then we write $[X]$ for the congruence class containing them. Note that for any strings u, v, $[uv] \supseteq [u][v]$ so if X, Y are congruence classes we can write $[XY]$ and the result is well defined.

The unique congruence class $[\lambda]$ is called the unit congruence class. The set $\{u \mid D_L(u) = \emptyset\}$ if it is non-empty is a congruence class, which is called the zero congruence class. A congruence class in a language L is non-zero iff it is a subset of $\mathrm{Sub}(L)$. We are mostly concerned with non-zero non-unit congruence classes in this paper.

Definition 1 *We will be considering sequences of congruence classes: so if α is a sequence $X_1, \ldots, X_n$ where each of the X_i is a congruence class, then we write $\bar{\alpha}$ for the set of strings formed by concatenating all of the X_i. We write $|\alpha|$ for the length of the sequence, n in this case. Note that all of the elements of $\bar{\alpha}$ will be congruent: if $u, v \in \bar{\alpha}$ then $u \equiv_L v$. We can therefore write without ambiguity $[\bar{\alpha}]$ for the congruence class of the strings in $\bar{\alpha}$.*

We say that $u \doteq_L v$ if there is some (l, r) such that $lur \in L$ and $lvr \in L$. This is partial or weak substitutability; u and v can be substituted for each other in the context (l, r). If $u \equiv_L v$ and u, v have a non-empty distribution then $u \doteq_L v$, but the converse is clearly not true.

Definition 2 *A language L is* substitutable *if for all $u, v \in \Sigma^+$, $u \doteq_L v$ implies $u \equiv_L v$.*

In other words, for any two non-empty strings u, v if $D_L(u) \cap D_L(v) \neq \emptyset$ then $D_L(u) = D_L(v)$. This language theoretic closure property allows us to define algorithms that generalise correctly, even under pessimistic learning conditions.

2.1 Context-free grammars

A context-free grammar (CFG) G is a tuple $G = \langle \Sigma, V, I, P \rangle$ where V is a finite non-empty set of nonterminals disjoint from Σ, $I \subseteq V$ is a set of distinguished start symbols and P is a subset of $V \times (V \cup \Sigma)^+$ called the set of productions. We write this as $N \to \alpha$. We do not allow productions with a right hand side of length 0, and as a result the languages we consider will not contain the empty string. We use $\mathcal{G}_{\mathrm{CFG}}$ for the class of all context-free grammars.

We define the standard notion of single-step derivation as $\Rightarrow$ and define $\overset{*}{\Rightarrow}$ as the reflexive transitive closure of $\Rightarrow$; for all $N \in V$, $\mathcal{L}(G, N) = \{w \in \Sigma^* \mid N \overset{*}{\Rightarrow} w\}$; and $\mathcal{L}(G) = \bigcup_{S \in I} \mathcal{L}(G, S)$. Using a set of start symbols rather than a single start symbol does not change the generative capacity of the formalism.

We say that a CFG is trim if for every nonterminal N there is a context (l, r) such that $S \overset{*}{\Rightarrow} lNr$ for some $S \in I$ and a string u such that $N \overset{*}{\Rightarrow} u$: in other words every nonterminal can be used in the derivation of some string.

We say that two CFGs, G and G' are weakly equivalent if $\mathcal{L}(G) = \mathcal{L}(G')$.

Proposition 3 *(Ginsburg, 1966) Given two CFGs, G and G', it is undecidable whether $\mathcal{L}(G) = \mathcal{L}(G')$.*

Two CFGs are isomorphic if there is a bijection between the two sets of nonterminals which extends to a bijection between the productions. In other words they are identical up to a relabeling of nonterminals. We denote this by $G \cong G'$. Clearly if two grammars are isomorphic then they are weakly equivalent.

Proposition 4 *Given two* CFG*s, G and G', it is decidable whether $G \cong G'$.*

There is a trivial exponential time algorithm that involves searching through all possible bijections. This problem is GI-complete: as hard as the problem of graph isomorphism (Zemlyachenko et al., 1985; Read and Corneil, 1977). We may not be able to do this efficiently for general CFGs.

3. Learning models

We start by reviewing the basic theory of learnability using the Gold identification in the limit paradigm (Gold, 1967). We consider only the model of given text — where the learner is provided with positive data only. We assume a class of CFGs, $\mathcal{G} \subseteq \mathcal{G}_{\text{CFG}}$.

A presentation of a language L is an infinite sequence of elements of Σ^*, $w_1, w_2, \ldots$ such that $L = \{w_i \mid i > 0\}$. Given a presentation $T = \langle w_1, \ldots \rangle$, we write T_n for the finite subsequence consisting of the first n elements. A polynomial learning algorithm is a polynomially computable function from finite sequences of positive examples to $\mathcal{G}_{\text{CFG}}$.

Given a presentation T of some language L, we can apply A to the various prefixes of T, which produces an infinite sequence of elements of $\mathcal{G}_{\text{CFG}}$, $A(T_1), A(T_2), \ldots$. These are hypothesis grammars; we will use G_i to refer to $A(T_i)$, the ith hypothesis output by the learning algorithm.

Consider a target grammar $G_* \in \mathcal{G}$, and a sequence of hypothesized grammars $G_1, G_2, \ldots$ produced by a learning algorithm on a presentation T for $\mathcal{L}(G_*)$. There are various notions of convergence of which we outline four, which vary on two dimensions: one dimension concerns whether we are interested in weak or strong learning, and the other whether we are interested in controlling the number of internal changes as well, or are only interested in the external behaviour.

Weak behaviorally correct learning (WBC)
There is an N such that for all $n > N$, $\mathcal{L}(G_n) = \mathcal{L}(G_*)$.

Weak Gold learning (GOLD)
There is an N such that for all $n > N$, $\mathcal{L}(G_n) = \mathcal{L}(G_*)$ and $G_n = G_N$.

Strong behaviorally correct learning (SBC)
There is an N such that for all $n > N$, $G_n \cong G_*$.

Strong Gold learning (SGOLD)
There is an N such that for all $n > N$, $G_n \cong G_*$ and $G_n = G_N$.

For each of these four notions of convergence, we have a corresponding notion of learnability. We say that a learner, A, WBC/GOLD/SBC/SGOLD learns a grammar G_*, iff for every presentation of $\mathcal{L}(G_*)$, it WBC/GOLD/SBC/SGOLD converges on that presentation. Given a class of CFGs, $\mathcal{G}$, we say that A WBC/GOLD/SBC/SGOLD learns the class, iff for every G_* in $\mathcal{G}$ the learner WBC/GOLD/SBC/SGOLD learns G_*.

In the case of GOLD learning, this coincides precisely with the standard model of Gold identification in the limit from given text (positive data only) (Gold, 1967). WBC-learning is the standard model of behaviorally correct learning (Case and Lynes, 1982). We cannot

in general turn a WBC-learner into a GOLD-learner: see discussion in Osherson et al. (1986). The property of *order-independence* as defined by Blum and Blum (1975), can be thought of as an even stronger version of SGOLD learning.

However, a SBC-learner can be changed into a SGOLD-learner, if we can test whether two hypotheses are isomorphic. There does not seem to be a theoretically interesting difference between SBC-learning and SGOLD-learning: the only difference, in the case of CFGs, is that the SBC learner may occasionally pick different labels for the nonterminals after convergence, whereas the SGOLD learner may not.

We can ask how can a GOLD learner differ from a SGOLD learner: how can a weak learner fail to be a strong learner? The difference is that on different presentations of the same language, a weak Gold learner may converge to different answers. That is to say we might have a learner which on presentation T' of grammar G produces a grammar G' and on presentation T'' of the same grammar, produces a grammar G'', where G' and G'' are weakly equivalent but not isomorphic.

Definition 5 *We say that a class of context-free grammars is redundant if it contains two grammars G_1, G_2 such that $G_1 \ncong G_2$ and $\mathcal{L}(G_1) = \mathcal{L}(G_2)$.*

Proposition 6 *Suppose that A is an algorithm which* SGOLD*-learns a class of grammars $\mathcal{G}$. Then $\mathcal{G}$ is not redundant.*

The proof is immediate — any presentation for G_1 is also a presentation for G_2. In other words if $\mathcal{G}$ contains two non-isomorphic grammars for the same language then it is not strongly learnable. A simple corollary is then that the class of CFGs is not strongly identifiable in the limit even from informed data, that is to say from labelled positive and negative examples, since there are an infinite number of non-isomorphic grammars for every non-empty language.

One can therefore try to convert a weak learner to a strong learner by defining a canonical form. If we can restrict the class so that there is only one structurally distinct grammar for each language, and we can compute that, then we could find a strong learning algorithm. We formalise this as follows. Suppose A is some learning algorithm that outputs grammars in a hypothesis class $\mathcal{H}_\mathcal{A} \subseteq \mathcal{G}_{\text{CFG}}$, and suppose that it can GOLD-learn the class of grammars $\mathcal{G} \subseteq \mathcal{H}_\mathcal{A}$. Suppose we have some 'canonicalisation' function f from $\mathcal{H}_\mathcal{A} \to \mathcal{G}_{\text{CFG}}$ such that for each $G \in \mathcal{G}$, $\mathcal{L}(G) = \mathcal{L}(f(G))$ and such that $f(\mathcal{G})$ is not redundant. Then we can construct a learner A' which outputs $A'(T_i) = f(A(T_i))$, which will then be a SGOLD learner for $\mathcal{G}$. Moreover, if A and f are both polynomially computable then so will A' be.

For example, suppose $\mathcal{D}$ is the class of all DFAs and f is the standard function for minimizing a deterministic finite-state automaton (DFA), which can be done in polynomial time. Since all minimal DFAs for a given regular language are isomorphic, $f(\mathcal{D})$ is not redundant. Therefore any learner for regular languages that outputs DFAs, such as the one in Angluin (1982), can be converted into a strong learner using this technique. From the point of view of structural learning such results are trivial in two important respects. The first is that each string in the language has exactly one labelled structure, and the other is that every structure is uniformly right branching, whereas we are interested in learning grammars which may assign more than one different structure to a given string.

Moreover, it is easy to see that any SGOLD-learner for a class of grammars $\mathcal{G}$ will implicitly define such a canonicalisation function for $\mathcal{G}$. We can enumerate the strings in the language and apply the learner to them, and the limit of the hypothesis grammars will then satisfy the conditions given above, though this function may not be computable. There is therefore a close relationship between canonicalisers and strong learners. There is much more that could be said about the learning models here, and further refinements of them, but this is enough for our purposes.

4. Weak Learning of Substitutable Languages

We recall the Clark and Eyraud (2007) result, using a simplified version of the algorithm (Yoshinaka, 2008), and explain why it is only a weak rather than a strong result.

Given a finite non-empty set of strings $D = \{w_1, \ldots, w_n\}$ the learner constructs a grammar as shown in Algorithm 1. We create a set of symbols in bijection with the elements of $\mathrm{Sub}(D)$ where we write $[[u]]$ for the symbol corresponding to the substring u: that is to say we have one nonterminal for each substring of the observed data. The grammar $\hat{G}(D)$ is the CFG $\langle \Sigma, V, I, P_L \cup P_B \cup P_U \rangle$ as shown in the pseudocode in Algorithm 1. The sets P_L, P_B and P_U are the sets of lexical, branching and unary productions respectively.

Algorithm 1 Grammar construction procedure

Data: A finite set of strings $w_1, w_2, \ldots, w_n$
Result: A CFG G
let $V := \mathrm{Sub}(D)$;
$I = \{[[u]] \mid u \in D\}$
$P_L = \{[[a]] \to a \mid a \in \Sigma \cap V\}$
$P_B = \{[[uv]] \to [[u]][[v]] \mid u, v, uv \in V\}$
$P_U = \{[[u]] \to [[v]] \mid \exists (l, r) \wedge lur \in D \wedge lvr \in D\}$
output $G = \langle \Sigma, V, I, P_L \cup P_B \cup P_U \rangle$

Example 1 *Given a set* $D = \{c, acb\}$*, we have* $\mathrm{Sub}(D) = \{a, b, c, ac, cb, acb\}$*, and corresponding sets:* $V = \{[[a]], [[b]], [[c]], [[ac]], [[cb]], [[acb]]\}$*,* $I = \{[[c]], [[acb]]\}$*,* $P_L = \{[[a]] \to a, [[b]] \to b, [[c]] \to c\}$*,* $P_B = \{[[ac]] \to [[a]][[c]], [[cb]] \to [[c]][[b]], [[acb]] \to [[ac]][[b]], [[acb]] \to [[a]][[cb]]\}$ *and* $P_U = \{[[c]] \to [[acb]], [[acb]] \to [[c]]\}$*. As can be verified this* CFG *defines the language* $\{a^n c b^n \mid n \geq 0\}$*.*

There are two natural ways to turn this grammar construction procedure into a learning algorithm. One is simply to apply this procedure to all of the available data. This will give a WBC-learner for the class of substitutable CFGs.

Alternatively since we can parse with the grammars, we can convert this into a GOLD learner, by only changing the hypothesis when the hypothesis is demonstrably too small. This means that once we have a weakly correct hypothesis the learner will no longer change its output. This simple modification gives a variant of the learner in Clark and Eyraud (2007). However this does not mean that this is a strong learner, since it may converge to a different hypothesis for different presentations of the same language. For example if a presentation of the language from Example 1 starts $\{a, acb, \ldots\}$ then the learner will

converge in two steps to the grammar shown in Example 1. If on the other hand, the presentation starts $\{acb, aacbb, \dots\}$ then it will also converge in two steps, but to a different, larger, grammar that includes nonterminals such as $[[aa]]$ and has a larger set of productions. This grammar is weakly equivalent to the former grammar, but it is not isomorphic or structurally equivalent, as it will assign a larger set of parses to strings like *aacbb*. It is more ambiguous. Indeed it is easy to see that this grammar will assign every possible binary branching structure to any string that is part of the set that the grammar is constructed from. And of course, the presentation could start with an arbitrarily long string — in which case the first grammar which it generates could be arbitrarily large.

5. The syntactic structure of substitutable languages

In this section we use a modification of Clark (2011) as the basis for our canonical grammars; in the case of substitutable languages the theory is quite simple so we will not present it in all its generality. Each nonterminal/syntactic category will correspond to a congruence class. With substitutable languages, we can show that the language itself, considered as a set of strings, has a simple intrinsic structure that can be used to define a particular finite grammar.

We start with the following definition:

Definition 7 *A congruence class X is* prime *if it is non-zero and non-unit and for any two congruence classes Y, Z such that $X = Y \cdot Z$ then either Y or Z is the unit. If a non-zero non-unit congruence class is not prime then we say it is* composite.

In other words a class is not prime if it can be decomposed into the concatenation of two other congruence classes. The stipulation that the unit and zero congruence classes are not prime is analogous to the stipulation that 1 is not a prime number. We will not give a detailed exposition of why the concept of a prime congruence class is important, but one intuitive reason is this. If we have nonterminals that correspond to congruence classes, and a congruence class N is composite, then that means that we can decompose N into two classes P, Q such that $N = PQ$. In that case we can replace every occurrence of N on the right hand side of a rule by the sequence $\langle P, Q\rangle$; assuming that P and Q can be represented adequately, nothing will be lost. Thus non-prime congruence classes can always be replaced by a sequence of prime congruence classes, and we can limit our attention to the primes which informally are those where "the whole is greater than the sum of the parts". More algebraically, we can think of the primes as representing the points where the concatenation operations in the free monoid and the syntactic monoid differ in interesting ways.

Example 2 *Consider the language $L = \{a^n c b^n \mid n \geq 0\}$. This language is not regular and therefore has an infinite number of congruence classes of which three are prime. The congruence classes are as follows:*

- *$\{\lambda\}$ is a congruence class with just one element; this is the unit congruence class which is not prime.*
- *The zero congruence class which consists of all strings that have empty distribution.*

- *L is a congruence class which is prime.*
- *$[a] = \{a\}$ is prime as is $[b] = \{b\}$.*
- *We also have an infinite number of congruence classes of the form $\{a^i\}$ for any $i > 1$. These are all composite as they can be represented as $[a] \cdot [a^{i-1}]$; similarly for $\{b^i\}$.*
- *Similarly we have classes of the form $[a^i c] = \{a^{i+j} c b^j \mid j \geq 0\}$ and $[cb^i] = \{a^j c b^{i+j} \mid j \geq 0\}$ which again are composite.*

L is not always prime as the following trivial example demonstrates.

Example 3 *Consider the finite language $L = \{ab\}$. This language has 5 congruence classes: $[a], [b], [ab], [\lambda]$ and the zero congruence class. The first 4 are all singleton sets. $[a]$ and $[b]$ are prime but $[ab] = \{ab\} = [a][b]$, and so L is not prime.*

Proposition 8 *For every $a \in \Sigma$, for any language L, if $[a]$ is non-zero and non-unit then $[a]$ is prime.*

Proof Let a be some letter in a language L and let $[a]$ be its congruence class. Suppose there are two congruence classes X, Y such that $XY = [a]$. Since $a \in [a]$, a must be in XY. Since we cannot split a string of length 1 into two non-empty strings, one of X and Y must be the unit. ■

We can now define the class of languages that we target with our learning algorithm.

Definition 9 *Let $\mathcal{L}_{sc}$ be the set of all languages which are substitutable, non-empty, do not contain λ and have a finite number of prime congruence classes.*

Given that there are substitutable languages which are not CFLs — the MIX language (Kanazawa and Salvati, 2012) being a good example — we need to restrict the class in some way. Here we consider only languages where there are a finite number of prime congruence classes. This implies, as we shall see later, that the language is a CFL. Every regular language of course has a finite number of primes as it has a finite number of congruence classes. Not all substitutable context-free languages have a finite number of primes, as this next example shows.

Example 4 *Consider the language $L = \{c^i b a^i b \mid i > 0\} \cup \{c^i d e^i d \mid i > 0\}$. This is a substitutable context-free language. The distribution of $ba^i b$ is the single context $\{(c^i, \lambda)\}$ which is the same as that of $de^i d$. Therefore we have an infinite number of congruence classes of the form $\{ba^i b, de^i d\}$, each of which is prime.*

Definition 10 *A* prime decomposition *of a congruence class X is a finite sequence of one or more prime congruence classes $\alpha = \langle X_1, \ldots, X_k \rangle$ such that $X = \bar{\alpha}$.*

Clearly any prime congruence class X has a trivial prime decomposition of length one, namely $\langle X \rangle$. We have a prime factorization lemma for substitutable languages; we can rather pompously call this the 'fundamental lemma' by analogy with the fundamental lemma of arithmetic. This lemma means that we can represent all of the congruence classes exactly using just concatenations of the prime congruence classes.

Lemma 11 *Every non-zero non-unit congruence class of a language in $\mathcal{L}_{sc}$ has a unique prime factorisation.*

For proof see Lemma 33 in the appendix. Note that this is not the case in general for languages which are not substitutable, as the following example demonstrates.

Example 5 *Let $L = \{abcd, apcd, bx\}$. Note that L is finite but not substitutable since $p \not\equiv b$. Among the congruence classes are $\{a\}, \{b\}, \{c\}$ $\{ab, ap\}$, $\{bc, pc\}$ and $\{abc, apc\}$. Clearly $\{ab, ap\}$, $\{bc, pc\}$ are both prime but $\{abc, apc\}$ is composite and has the two distinct prime decompositions $\{ab, ap\} \cdot \{c\}$ and $\{a\} \cdot \{bc, pc\}$.*

If we restrict ourselves to languages in $\mathcal{L}_{sc}$ then we can assume without loss of generality that the nonterminals of the generating grammar correspond to congruence classes. In a substitutable language, a trim CFG cannot have a nonterminal that generates two strings that are not congruent. Similarly, if the grammar had two distinct nonterminals that generated congruent strings, we could merge them without altering the generated language.

Given that non-regular languages will have an infinite number of congruence classes, and that CFGs have by definition only a finite number of nonterminals, we cannot have one nonterminal for every congruence class. However in languages in $\mathcal{L}_{sc}$ there are only finitely many prime congruence classes, and since every other congruence class can be represented perfectly as a sequence of primes, it is sufficient to consider a grammar which has nonterminals that correspond to the primes. Therefore we will consider grammars whose nonterminals correspond only to the prime congruence classes of the grammar: we add one extra nonterminal S, a start symbol, which will not appear on the right hand side of any rule.

5.1 Productions

We now consider an abstract notion of a production where the nonterminals are the prime congruence classes.

Definition 12 *A* correct *branching production is of the form $[\bar{\alpha}] \to \alpha$ where α is a sequence of at least 2 primes and $[\bar{\alpha}]$ is a prime congruence class. A correct lexical production is one of the form $[a] \to a$ where $a \in \Sigma$, and $[a]$ is prime.*

Example 6 *Consider the language $L = \{a^n c b^n \mid n \geq 0\}$. This has primes $[a], [c]$ and $[b]$. The correct lexical productions are the three obvious ones $[a] \to a$, $[b] \to b$ and $[c] \to c$. The only correct branching productions have $[c]$ on the left hand side, and are $[c] \to [a][c][b]$, $[c] \to [a][a][c][b][b]$ and so on.*

Clearly in the previous example we want to rule out productions like $[c] \to [a][a][c][b][b]$ since the right hand sides are too long, and will make the derivation trees too flat. We want each production to be as simple as possible. Informally we say that the right hand side of the production $[a][a][c][b][b]$ is too long since there is a proper subsequence $[a][c][b]$ which generates strings in a prime congruence class, and should be represented just by the prime $[c]$.

Definition 13 *We say that a sequence of primes α is* pleonastic *(too long) if $\alpha = \gamma\beta\delta$ for some γ, β, δ, which are sequences of primes, such that $|\gamma| + |\delta| > 0$, $[\bar{\beta}]$ is a prime, and $|\beta| > 1$.*

Definition 14 *We say that a correct production $N \to \alpha$ is* pleonastic *if α is pleonastic. A correct production is* valid *if it is not pleonastic.*

Note that a pleonastic production by definition must have a right hand side of length at least 3.

For any string w in a prime congruence class where $w = a_1 \dots a_n$, $a_i \in \Sigma$ we can construct a correct production $[w] \to [a_1] \dots [a_n]$. Such productions may in general be pleonastic because there may be substrings that can be represented by prime congruence classes. From a structural perspective, the local trees derived from these productions are too shallow as they flatten out relevant parts of the structure of the string. Nonetheless we can find a set of valid productions that will generate the string w from the nonterminal $[w]$, as Lemma 18 below shows.

5.2 Canonical Grammars

We will now define canonical grammars for every language L in $\mathcal{L}_{sc}$. Note that for every language in $\mathcal{L}_{sc}$, L is a congruence class.

First of all we need the following lemma to establish that the grammar will be finite: see proof of Lemma 35 in the appendix.

Lemma 15 *If $L \in \mathcal{L}_{sc}$ then there are a finite number of valid productions.*

Definition 16 *Let L be some language in $\mathcal{L}_{sc}$. We define the following grammar, $G_*(L)$. The nonterminals are the prime congruence classes of L, together with an additional symbol S, which is the start symbol. Let $\alpha(L)$ be the unique prime decomposition of L. We define the set of productions P to be the union of the following three sets:*

- $\{S \to \alpha(L)\}$,
- *The set of all valid productions, which is finite by Lemma 15,*
- *For each terminal symbol a that occurs in the language, the production $[a] \to a$.*

This is a unique CFG for every language in $\mathcal{L}_{sc}$. We now show that $G_*(L)$ generates L.

Lemma 17 *If $L \in \mathcal{L}_{sc}$ is a substitutable language, then for any prime congruence class X, $\mathcal{L}(G_*(L), X) \subseteq X$.*

Proof This is a simple induction on the length of the derivation. For $X \to a$, we know that $a \in X$ by construction. Suppose $X_i \stackrel{*}{\Rightarrow} u_i$ for all $1 \leq i \leq n$ and $X_0 \to X_1 \dots X_n$ is a production in the grammar. Then by the inductive hypothesis $u_i \in X_i$ and by the correctness of the production, $u_1 \dots u_n \in X_0$. ■

Lemma 18 *Suppose $X \to \alpha$ is a correct production. Then $X \stackrel{*}{\Rightarrow}_{G_*(L)} \alpha$.*

Proof By induction on the length of α. Base case: α is of length 2, in which case it cannot be pleonastic, and so $X \to \alpha$ is valid and in $G_*(L)$, and therefore $X \stackrel{*}{\Rightarrow} \alpha$. Inductive step: consider a correct production $X \to \alpha$ where α is of length k. If it is not pleonastic, then it is valid, and so $X \to \alpha$ is a production in $G_*(L)$, and so $X \stackrel{*}{\Rightarrow} \alpha$. Alternatively it is pleonastic and therefore $\alpha = \beta\gamma\delta$ where γ is the right hand side of a correct production, $Y \to \gamma$. Consider $X \to \beta Y \delta$, and $Y \to \gamma$. Both $\beta Y \delta$ and γ are shorter than α, and so by the inductive hypothesis $X \stackrel{*}{\Rightarrow} \beta Y \delta$ and $Y \stackrel{*}{\Rightarrow} \gamma$ so $X \stackrel{*}{\Rightarrow} \alpha$. So the lemma follows by induction. ■

Lemma 19 *Suppose X is a prime, and $w \in X$. Then $X \stackrel{*}{\Rightarrow}_{G_*(L)} w$.*

Proof If w is of length 1, then we have $X \to w$. Let $w = a_1 \dots a_n$ be some string of length $n > 1$. Let $\alpha = [a_1] \dots [a_n]$. So $X \to \alpha$ is a correct production. Therefore by Lemma 18 $X \stackrel{*}{\Rightarrow} \alpha$. Since we have the lexical rules $[a_i] \to a_i$ we can also derive $\alpha \stackrel{*}{\Rightarrow} w$. ■

Proposition 20 *For any $L \in \mathcal{L}_{sc}$, $\mathcal{L}(G_*(L)) = L$.*

Proof Suppose L has prime factorisation $A_1 \dots A_n$. S occurs on the left hand side of the single production $S \to A_1 \dots A_n$. Since $\mathcal{L}(G_*(L), A_i) = A_i$ by Lemmas 17 and 19, $\mathcal{L}(G_*(L), S) = A_1 \dots A_n = L$. ■

Definition 21 *We write $\mathcal{G}_{sc}$ to be the set of canonical context-free grammars for the languages in $\mathcal{L}_{sc}$.*

$$\mathcal{G}_{sc} = \{G_*(L) \mid L \in \mathcal{L}_{sc}\}$$

Lemma 22 *$\mathcal{G}_{sc}$ is not redundant.*

Proof Suppose we have two weakly equivalent grammars G_1, G_2 in this class; then $G_1 = G_*(\mathcal{L}(G_1)) = G_*(\mathcal{L}(G_2)) = G_2$ and so they are isomorphic. ■

6. An algorithm for strong learning

We now present a strong learning algorithm. We then demonstrate in Section 7 that for all grammars in $\mathcal{G}_{sc}$ the algorithm strongly converges in the SGOLD framework.

In outline, the algorithm works as follows; we accumulate all of the data that we have seen so far into a finite set D. We start by using Algorithm 1 to construct a CFG G^w which will be weakly correct for a sufficiently large input data. Using this observed data, together with the grammar which is used for parsing, we can then compute the canonical grammar for the language as follows.

1. We partition $\mathrm{Sub}(D)$ into congruence classes, with respect to our learned grammar G^w.
2. We pick the lexicographically shortest string in each class as the label we will use for the nonterminal.
3. We then test to see which of the congruence classes are prime.
4. Each class is decomposed uniquely into a sequence of primes.
5. A set of valid rules is constructed from the strings in the prime congruence classes.
6. We then eliminate pleonastic productions from this set of productions.
7. Finally, we return a grammar G^s constructed from these productions.

We can perform the first task efficiently, using the grammar and the substitutability property. Given that each string in $\mathrm{Sub}(D)$ occurs in the sample D, for each substring u we have some context (l, r) such that $lur \in D$. Given the substitutability condition, v is congruent to u iff $lvr \in \mathcal{L}(G_*)$. Under the assumption that the grammar is correct we can test this by seeing whether $lvr \in \mathcal{L}(G^w)$, using a standard polynomial parser, such as a CKY parser.

We now have a partition of $\mathrm{Sub}(D)$ into k classes $C_1, \ldots, C_k$. We pick the lexicographically shortest element of each class (with respect to $\prec$) which we denote by $u_1, \ldots, u_k$. Given a class, we want to test whether it is prime or not. Take the shortest element w in the class. Test every possible split of w into non-zero strings u, v such that $uv = w$. Clearly there are $|w| - 1$ possible splits — for each split, identify the classes of u, v and test to see whether every element in the class can be formed as a concatenation of these two. If there is some string that cannot be split, then we know that the congruence class must be prime. If on the other hand we conclude that the class is not prime, we might potentially be wrong: we might for example think that $X = YZ$ simply because we have not yet observed one of the strings in $X \setminus YZ$. We present the pseudocode for this procedure in Algorithm 2.

Algorithm 2 Testing for primality

Data: A set of strings X
Data: A partition of strings $\mathcal{X} = \{X_1, \ldots, X_n\}$, such that $\mathrm{Sub}(X) \subseteq \bigcup \mathcal{X}$
Result: True or false
Select shortest $w \in X$;
for $u, v \in \Sigma^+$ such that $uv = w$ **do**
  $X_i \in \mathcal{X}$ is the set such that $u \in X_i$
  $X_j \in \mathcal{X}$ is the set such that $v \in X_j$
  if $X \subseteq X_i X_j$ **then**
    return false;
  end if
end for
return true;

For all of the non-prime congruence classes, we now want to compute the unique decomposition into primes. There are a number of obvious polynomial algorithms. We start by taking the shortest string w in a class; suppose it is of length n consisting of $a_1 \dots a_n$. We convert this into a sequence of primes $[a_1] \dots [a_n]$. We then greedily convert this into a unique shortest sequence of primes by checking every proper subsequence of length at least 2, and seeing if that string is in a prime congruence class. If it is then we replace that subsequence by the prime. We repeat until there are no proper subsequences that are primes. Alternatively we can use a shortest path algorithm. We create a graph which has one node for each $0, 1, \dots, n$. We create an arc from $i \to j$ if the substring spanning $[i, j]$ is prime. We then take the shortest path from 0 to n; and read off the sequence of primes by looking at the primes of the relevant segments. Note that since the lexical congruence classes are all prime, we know there will be at least one such path; since the language is substitutable we know this will be unique.

We then identify a set of valid productions. Every valid production will be of the form $N \to M\alpha$ where N, M are primes and α is a prime decomposition of length at least 1. For any given N, M there will be at most one such rule. Accordingly we loop through all triples of N and M, Q as follows: for each prime N, for each prime M, for each class Q, take α to be the prime decomposition of Q, and test to see if $N \to M\alpha$ is valid. We can test if it is correct easily by taking any shortest string u from M and any shortest string v from α and seeing if $uv \in N$; if it does then the rule is correct. Then we can test if it is valid by taking every proper prefix of $M\alpha$ of length at least two and testing if it corresponds to a prime. If no prefix does then the production is not pleonastic and is therefore valid.

For the lexical productions, we simply add all productions of the form $[a] \to a$ where $a \in \Sigma$. For the initial symbol S, we identify the unique congruence class of strings in the language X. If it is prime, then we add a rule $S \to X$. If it is not prime, and α is its unique prime decomposition then we add the rule $S \to \alpha$.

7. Analysis

We now proceed to the analysis of Algorithm 3, the learner $\mathcal{A}_{\text{SGOLD}}$. We want to prove three things: first that the algorithm strongly learns a certain class; secondly, that the algorithm runs in polynomial update time; finally that the algorithm converges rapidly, in the technical sense that it has a polynomially sized characteristic set.

We now are in a position to state our main result. We have defined a learning model, SGOLD, an algorithm A_{SGOLD} and a class of grammars $\mathcal{G}_{sc}$.

Theorem 23 *A_{SGOLD} SGOLD-learns the class of grammars $\mathcal{G}_{sc}$.*

In order to prove this we will show that for any presentation of a grammar in the class we will converge strongly to a grammar isomorphic to the canonical grammar. In what follows we suppose G_* is a grammar in $\mathcal{G}_{sc}$, and that $L_* = \mathcal{L}(G_*)$. For a grammar $G_* \in \mathcal{G}_{sc}$, we define $\chi(G_*)$ to be a sufficiently large, yet polynomially bounded set of strings from $\mathcal{L}(G_*)$ such that when the input data includes this set, the weak grammar output will be correct (Clark and Eyraud, 2007) and which contains the shortest string in each prime congruence class.

Algorithm 3 $\mathcal{A}_{\text{SGOLD}}$ Strong Gold Learning Algorithm

Data: A sequence of strings $w_1, w_2, \ldots$
Data: Σ
Result: A sequence of CFGs $G_1, G_2, \ldots$
let $D := \varnothing$;
for $n = 1, 2, \ldots$ **do**
  let $D := D \cup \{w_n\}$;
  $\hat{G} = \hat{G}(D)$;
  Let C be the partition of $\text{Sub}(D)$ into classes;
  Let Pr be the set of primes computed using Algorithm 2;
  For each class N in C compute the prime decomposition $\alpha(N) \in Pr^+$;
  Let $V = \{[[N]] \mid N \in Pr\}$ be a set of nonterminals each labeled with the lexicographically shortest element in its class;
  Let S be a start symbol;
  $P_L = \{[[N]] \to a \mid [[N]] \in V, a \in \Sigma \cap N\}$
  $P_I = \{S \to [[N]] \mid \exists w \in N, w \in D\}$
  $P_B = \emptyset$
  for $N \in Pr$, $M \in Pr$, $Q \in C$ **do**
    $R = (N \to M\alpha(Q))$
    if R is correct and valid **then**
      $P_B = P_B \cup \{R\}$;
    end if
  end for
  output $G_n := \langle \Sigma, V \cup \{S\}, \{S\}, P_L \cup P_B \cup P_I \rangle$;
end for

Definition 24 *For a grammar $G = \langle \Sigma, V, I, P \rangle$ we define $\chi(G)$ as follows. For any $\alpha \in (\Sigma \cup V)^+$ we define $w(\alpha) \in \Sigma^+$ to be the smallest word, according to $\prec$, generated by α. 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 $S \stackrel{*}{\Rightarrow} lNr$. We now define the characteristic set $\chi(G_*) = \{lwr \mid (N \to \alpha) \in P, (l, r) = c(N), w = w(\alpha)\}$.*

We prove the correctness of the rest of the model under the assumption that the input data contains $\chi(G_*)$ and as a result that G^w is weakly correct: $\mathcal{L}(G^w) = \mathcal{L}(G_*)$. First, if G^w is correct, then the partition of $\text{Sub}(D)$ into congruence classes will be correct in the sense that two strings of $\text{Sub}(D)$ will be in the same class iff they are congruent.

Lemma 25 *Suppose $X_1, \ldots, X_n$ is a correct partition of $\text{Sub}(D)$ into congruence classes. Then if Algorithm 2 returns true when applied to X_i, then $[X_i]$ is in fact prime.*

Proof Suppose $[X_i]$ is not prime: then there are two congruence classes Y, Z such that $[X_i] = YZ$. Consider a string $w \in X_i$. There must be strings u, v such that $w = uv$ and $u \in Y, v \in Z$. Since $w \in \text{Sub}(D)$, $u, v \in \text{Sub}(D)$. Since the partition of $\text{Sub}(D)$ is correct, there must be sets X_j, X_k such that $u \in X_j, v \in X_k$. Therefore, using again the correctness

and the fact that Sub(D) is substring closed, we have that $X_i \subseteq X_j X_k$, in which case Algorithm 2 will return false. ■

Lemma 26 *Suppose $X_1, \ldots, X_n$ is a correct partition of* Sub(D) *into congruence classes, and $D \supseteq \chi(G_*)$. Then Algorithm 2 returns true when applied to X_i, iff $[X_i]$ is in fact prime.*

Proof X_i is a finite subset of Sub(D), and we assume that all of the elements of X_i are in fact congruent. We already showed one direction, namely that if the algorithm returns true then $[X_i]$ is prime (Lemma 25). We now need to show that if $[X_i]$ is prime, then the algorithm correctly returns true. If $[X_i]$ is prime, then it will correspond to some nonterminal in the canonical grammar G_*, say N. There will be more than one production in G_* with N on the left hand side, and so by the construction of $\chi(G_*)$, and the correctness of the weak grammar, we will have at least one string from each production in Sub(D), which means that since it is a correct partition the algorithm cannot find any pair of classes whose concatenation contains X_i. ■

As a consequence of this Lemma, we know that the algorithm will be able to correctly identify the set of primes of the language, and as a result will converge to the right set of nonterminals.

Proposition 27 *If the input data includes $\chi(G_*)$, then $G^s \cong G_*$.*

Proof We can verify that all and only the valid productions will be generated by the algorithm by the construction of the characteristic set.

Suppose $N \to X_1 \ldots X_n$ is a valid production in the grammar. Then by the construction of the characteristic set we will have a unique congruence class in the grammar corresponding to $[X_2 \cdots X_n]$. If $n > 2$ then this will be composite, and if $n = 2$ this will be prime, but in any event it will have a unique prime decomposition which will be exactly $\langle X_2, \ldots, X_n \rangle$, by Lemma 33. Therefore this production will be produced by the algorithm.

Secondly suppose the algorithm produces some production $N \to X_1, \ldots X_n$. We know that this will be valid since $X_2, \ldots X_n$ is a prime decomposition and is thus not pleonastic, and we tested all of the prefixes. We know that it will be correct, by the correctness of the weak learner and the fact that the congruence classes are correctly divided. It is easy to verify that the lexical and initial rules are also correctly extracted. ■

To conclude the proof of Theorem 23, we just need to observe that since the characteristic set includes the shortest element of each prime congruence class, and so the labels for each nonterminal will not change which means that the output grammars will converge exactly.

We now consider the efficiency of the algorithm. It is easy show that this algorithm runs in polynomial time in the size of the data set $\|D\|$, noting first that $|\text{Sub}(D)|$ is polynomial in $\|D\|$, and that as a result the grammars generated are all of polynomial size. Moreover the characteristic set has cardinality which is polynomial in the size of the grammar, and whose size is polynomially bounded in the thickness (Wakatsuki and Tomita, 1993).

S → NT0	**S → NT0**	**S → NT0**
NT1 → b	**NT3 → open**	**NT2 → NT2 NT0**
NT2 → a	**NT2 → close**	**NT2 → NT0 NT2**
NT0 → c	**NT4 → neg**	**NT2 → a**
NT0 → NT2 NT0 NT1	**NT4 → NT0 NT1**	**NT0 → NT0 NT0**
	NT0 → a	**NT0 → NT2 NT1**
	NT0 → b	**NT0 → NT1 NT2**
	NT0 → c	**NT1 → b**
	NT0 → NT3 NT4 NT0 NT2	**NT1 → NT1 NT0**
	NT1 → and	**NT1 → NT0 NT1**
	NT1 → iff	
	NT1 → implies	
	NT1 → or	

Table 1: Output grammars for the three examples; on the left the grammar for $\{a^n cb^n \mid n \geq 0\}$, in the middlem the language of propositional logic, and on the right, the ambiguous grammar for $\{w \in \{a, b\}^+ \mid |w|_a = |w|_b\}$.

8. Examples

We have implemented the algorithm presented here.[2] We present the results of running this algorithm on small data sets that illustrate the properties of the canonical grammars for the learned languages. These examples are not intended to demonstrate the effectiveness of the algorithm but merely as illustrative examples to help the reader understand the representational assumptions, and as a result we have restricted ourselves to very simple languages which will be easy to understand. Nonterminals in the output grammar are either **S** for the start symbol or **NT** followed by a digit for the congruence classes that correspond to primes.

8.1 Trivial Context-Free Language

Consider the running example of $\{a^n cb^n \mid n \geq 0\}$. A characteristic set for this is just $\{c, acb\}$. Given this input data, we get the grammar shown on the left of Table 1. This defines the correct language; Figure 1 shows the parse trees for the three shortest strings in the language. This grammar is unambiguous so every string has only one tree.

8.2 Propositional Logic

Our next example is the language of sentential logic, with a finite number of propositional symbols. We have the alphabet $\{A_1, \ldots, A_k, (,), \neg, \vee, \wedge, \Rightarrow, \Leftrightarrow\}$. We would standardly define this language with the CFG: $S \to A_i$, $S \to (\neg S)$, $S \to (S \vee S)$, $S \to (S \wedge S)$, $S \to (S \Rightarrow S)$ and $S \to (S \Leftrightarrow S)$. Note that in this language the brackets are part of the object language not the meta-language — the algorithm does not know that they are brackets or what their function is. We replace them with other symbols in the experiment to

2. A Java implementation will be made available on the author's website.

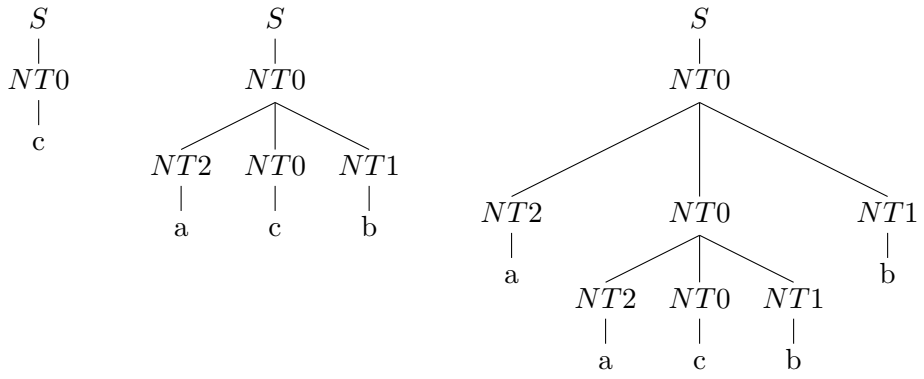

Figure 1: Example parse trees for the example $\{a^n c b^n \mid n \geq 0\}$.

emphasize this point. Thus the algorithm is given only flat sequences of strings — there is implicitly structural information here, but the algorithm must discover it, as it must discover that the correct grammar is unambiguous. Sentential logic is an interesting example because it illustrates a case where the algorithm works but produces a different parse tree, but one that is still adequate for semantic interpretation. The canonical structure does not look like the ancestral tree we would see in a textbook presentation (Enderton, 2001).

Since $(\neg A)$ and $(A \vee A)$ are both in the language, $\neg \cong A\vee$, so the parse tree for $(A \vee B)$ will look a little strange: the canonical grammar has pulled out some more structure than the textbook grammar does: see Figure 2 for example trees. Nonetheless this is still suitable for semantic interpretation and the grammar is still unambiguous.

We fix some input data, replacing the symbols with strings to obtain input data of { *a, b, c, open a and b close, open a or b close, open a implies c close, open a iff c close, open neg a close* }. This produces the grammar shown in the middle of Table 1, which is weakly correct. This generates one tree for each string in the language as shown in Figure 2.

8.3 Ambiguous language

The next example is the language which consists of equal numbers of a's and b's in any order: $\{w \in \{a, b\}^+ \mid |w|_a = |w|_b\}$. We give the input data: $\{ab, ba, abab, abba, baba, bbaa\}$. The resulting grammar has 10 productions as shown on the right of Table 1.

In this case the grammar is ambiguous and the number of parses for each string varies, depending on properties of the string that are more complex than just the length. For example, the string *abab* has 5 parses, the string *abba* has 3 and the string *aabb* has only 2.

9. Discussion

Our goal in this paper is to take a small but theoretically well-founded step in a novel direction. This is not merely a new learning result but a new type of learning result: a *strong* learning result for a class of languages that includes non-regular languages. The main points of this paper are to define the learning model, and to establish that it is possible to obtain such results for at least some CFGs from positive strings alone. To the best of

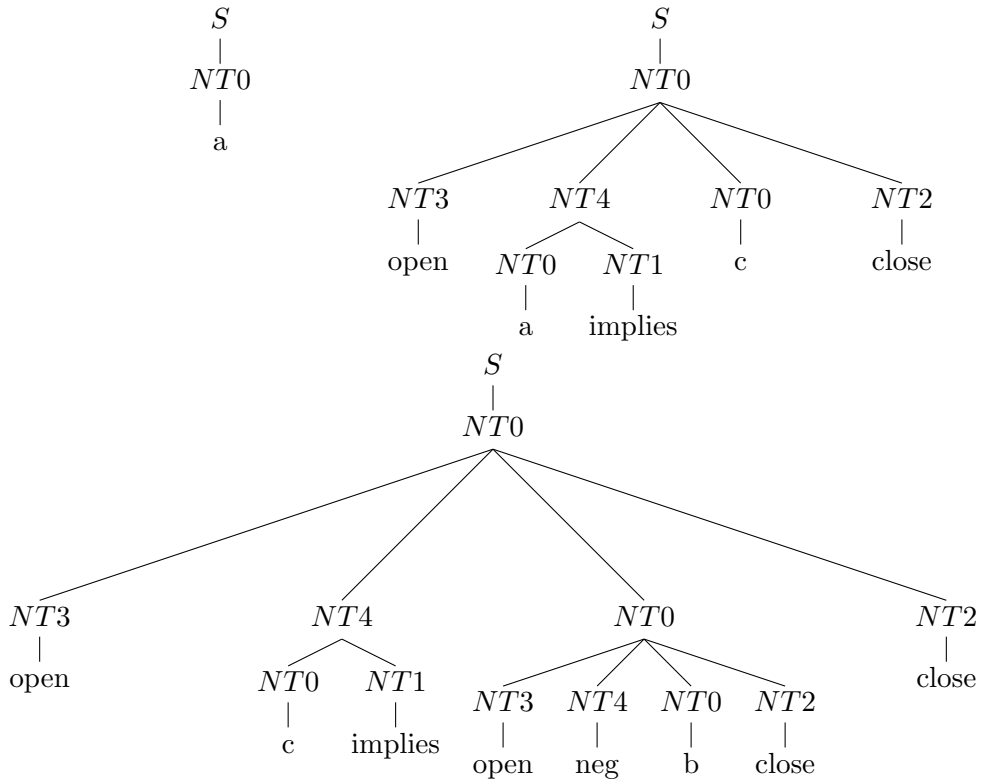

Figure 2: Example parse trees for the sentential logic example. Each example has only one parse tree.

the author's knowledge this is the first non-trivial learning result of this type. There are of course trivial enumerative algorithms that can strongly learn any non-redundant finite class of CFGs from positive data given a list of the elements of the class ordered by inclusion, and as mentioned before, the algorithm presented by Angluin (1982) can be viewed also as a strong learner for deterministic regular grammars. The Gold learning model is too onerous and as a result the class of languages that can be learned is very limited, but nonetheless includes some interesting natural examples as we showed in the previous section.

Strong learning is hard — accordingly we decompose it into two subproblems of rather different flavors. The first is a weak learning algorithm, and the second is a component that converts a weak learner to a strong learner; the latter component can be thought of as the computation of a canonical form. In general it will not be possible to compute a canonical form for an arbitrary grammar as this will be undecidable; however we may be able to do this for the grammars output by weak learners which will typically produce grammars in a restricted class.

In this paper, we have chosen to work using the simplest type of weak learner, and using only CFGs. The algorithm we have obtained therefore lacks some important features of natural language; notably lexical ambiguity and displacement. It also relies on an overly

strong language theoretic closure property (substitutability) that natural languages do not satisfy. It is natural therefore to extend this in two ways. Firstly instead of using congruence classes as the basis for the nonterminals in the grammar, we can use syntactic concepts (Clark, 2013) which can be used to represent all CFGs, and secondly we can move from CFGs to a much richer class of grammars — the class of well-nested 2-MCFGs (Seki et al., 1991). The fundamental lemma is a nice technical result which simplifies the algorithm and the proof; however we will not have such a clean property in the case of larger classes of languages. Nonetheless we can extend the notion of a prime congruence class naturally to the richer mathematical structures that we need to model the more complex grammar formalisms required for natural language syntax.

Acknowledgments

I am grateful to Rémi Eyraud and Anna Kasprzik for helpful comments and discussion, and to Ryo Yoshinaka for detailed comments on an earlier draft. I would also like to thank the anonymous reviewers for their extremely constructive and helpful comments, which have greatly improved the paper. Any remaining errors are of course my own.

References

D. Angluin. Inference of reversible languages. *Journal of the ACM*, 29(3):741–765, 1982.

R.C. Berwick, P. Pietroski, B. Yankama, and N. Chomsky. Poverty of the stimulus revisited. *Cognitive Science*, 35:1207–1242, 2011.

L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28(2):125–155, 1975.

J. Case and C. Lynes. Machine inductive inference and language identification. *Automata, Languages and Programming*, pages 107–115, 1982.

A. Clark. A language theoretic approach to syntactic structure. In Makoto Kanazawa, András Kornai, Marcus Kracht, and Hiroyuki Seki, editors, *The Mathematics of Language*, pages 39–56. Springer Berlin Heidelberg, 2011.

A. Clark. The syntactic concept lattice: Another algebraic theory of the context-free languages? *Journal of Logic and Computation*, 2013. doi: 10.1093/logcom/ext037.

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

T. Cohn, P. Blunsom, and S. Goldwater. Inducing tree-substitution grammars. *Journal of Machine Learning Research*, 11:3053–3096, 2010.

F. Drewes and J. Högberg. Learning a regular tree language from a teacher. In Zoltán Ésik and Zoltán Fülöp, editors, *Developments in Language Theory*, pages 279–291. Springer Berlin Heidelberg, 2003.

H. Enderton. *A mathematical introduction to logic*. Academic press, 2001.

S. Ginsburg. *The mathematical theory of context-free languages*. McGraw-Hill, New York, 1966.

E. Mark Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

D. Hsu, S. M. Kakade, and P. Liang. Identifiability and unmixing of latent parse trees. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1520–1528, 2013.

M. Kanazawa and S. Salvati. MIX is not a tree-adjoining language. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 666–674. Association for Computational Linguistics, 2012.

D. López, J.M. Sempere, and P. García. Inference of reversible tree languages. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(4):1658–1665, 2004.

P.H. Miller. *Strong generative capacity: The semantics of linguistic formalism*. CSLI Publications, Stanford, CA, 1999.

D. Osherson, M. Stob, and S. Weinstein. *Systems that Learn: An introduction to Learning Theory for Cognitive and Computer Scientists*. MIT Press, first edition, 1986.

T. Petrie. Probabilistic functions of finite state Markov chains. *The Annals of Mathematical Statistics*, 40(1):97–115, 1969.

R.C. Read and D.G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1 (4):339–363, 1977.

Y. Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76(2):223–242, 1990.

Y. Sakakibara. Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, 97(1):23 – 60, 1992.

R. E. Schapire. A brief introduction to boosting. In *Proceedings of 16th Interntational Joint Conference on Artifical Intelligence*, pages 1401–1406, 1999.

H. Seki, T. Matsumura, M. Fujii, and T. Kasami. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):229, 1991.

M. Wakatsuki and E. Tomita. A fast algorithm for checking the inclusion for very simple deterministic pushdown automata. *IEICE TRANSACTIONS on Information and Systems*, 76(10):1224–1233, 1993.

K. Wexler and P. W. Culicover. *Formal Principles of Language Acquisition*. MIT Press, Cambridge, MA, 1980.

R. Yoshinaka. Identification in the Limit of k-l-Substitutable Context-Free Languages. In Alexander Clark, François Coste, and Laurent Miclet, editors, *Grammatical Inference: Algorithms and Applications*, pages 266–279. Springer Berlin Heidelberg, 2008.

R. Yoshinaka. Efficient learning of multiple context-free languages with multidimensional substitutability from positive data. *Theoretical Computer Science*, 412(19):1821 – 1831, 2011.

R. Yoshinaka. Integration of the dual approaches in the distributional learning of context-free grammars. In Adrian-Horia Dediu and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications*, pages 538–550. Springer Berlin Heidelberg, 2012.

R. Yoshinaka and M. Kanazawa. Distributional learning of abstract categorial grammars. In Sylvain Pogodalla and Jean-Philippe Prost, editors, *Logical Aspects of Computational Linguistics*, pages 251–266. Springer Berlin Heidelberg, 2011.

V.N. Zemlyachenko, N.M. Korneenko, and R.I. Tyshkevich. Graph isomorphism problem. *Journal of Mathematical Sciences*, 29(4):1426–1481, 1985.

Appendix

This appendix contains the proofs of some technical lemmas that we use earlier that are not important from a learning theoretic point of view, but merely concern the algebraic properties of substitutable languages and their congruence classes. In all of the lemmas that follow, we assume we have a fixed language $L \in \mathcal{L}_{sc}$.

Lemma 28 *If X is a prime, and Y is a congruence class which is not equal to X, then there is a string in X which does not start with an element of Y.*

Proof Suppose every string in X starts with Y. Let x, x' be strings in X; then $x = yv$ and $x' = y'v'$ for some $y, y' \in Y$ and some other strings v, v'. Then $v \equiv v'$ by substitutability so $X = Y[v]$ and X is not prime. ■

Lemma 29 *Suppose $\alpha = A_1 \dots A_m$ and $\beta = B_1 \dots B_n$ are sequences of primes such that $\bar{\alpha} \supseteq \bar{\beta}$ then there is some j, $1 \leq j \leq n$ such that $A_1 \supseteq B_1 \dots B_j$.*

Proof If $B_1 = A_1$ then we are done. Alternatively pick some element $b_1 \in B_1$ which does not start with an element of A_1 (by Lemma 28). Now let w be some string in $B_2 \dots B_n$. Since $b_1 w \in \bar{\alpha}$ we must have some a_1, p_1 such that $a_1 = b_1 p_1$, where $a_1 \in A_1$. If $p_1 \in B_2$ then $B_1 B_2 \subseteq A_1$, so $j = 2$ and we are done. Otherwise take some element of B_2 that does not start with an element of $[p_1]$, say b_2. By the same argument we must have some $a_2 \in A_1$ and a p_2 such that $a_2 = b_1 b_2 p_2$, and where $b_2 p_2 \in [p_1]$. We repeat the process, and if we do not find some suitable j then we will have constructed a string in $\bar{\beta}$ which does not start with A_1 which contradicts the assumption that $\bar{\beta} \subseteq \bar{\alpha}$. Therefore there must be some j such that $B_1 \dots B_j \subseteq A_1$. ■

Lemma 30 *Suppose X is a prime, and α, β are strings of primes such that $X\bar{\alpha} \subseteq X\bar{\beta}$, where $X\bar{\beta} \subseteq \mathrm{Sub}(L)$, then $\bar{\alpha} \subseteq \bar{\beta}$.*

Proof Suppose $\alpha = A_1 \dots A_m$ and $\beta = B_1 \dots B_n$ are sequences of primes that satisfy the conditions of the lemma. Take some string in $\bar{\alpha}$, say a. Let x be a shortest string in X. xa is in the set $X\bar{\alpha}$ so we must have $xa = x'b$, for some $x' \in X, b \in \bar{\beta}$. Now x is the shortest string so either $x = x'$ and $a = b$ in which case the lemma holds, or $|x'| > |x|$ in which case we have $xcb = xa = x'b$, for some non-empty string c. So $xc = x'$ and x', x are both in X so $xc \equiv x$. Therefore $xcb \equiv xccb$ and so $b \equiv cb$, by substitutability. Now we can write b as a sequence of elements of β say $b = b_1 \dots b_n$, where $b_i \in B_i$. Since we have some context (l, r) such that $lbr \in L$ therefore $lcbr \in L$ by substitutability we will have $b_1 \equiv cb_1$ so $cb_1 \in B_1$ since it is a congruence class. This means that $cb \in \beta$ so $a \in \beta$ since $a = cb$. So $\bar{\alpha} \subseteq \bar{\beta}$. ■

An immediate corollary is this:

Lemma 31 *If X is a prime, and α, β are strings of primes such that $X\bar{\alpha} = X\bar{\beta}$, where $X\bar{\beta} \subseteq \mathrm{Sub}(L)$, then $\bar{\alpha} = \bar{\beta}$.*

Lemma 32 *If α and β are non-empty sequences of prime congruence classes such that $\bar{\alpha} = \bar{\beta} = [\bar{\alpha}]$, and $\bar{\alpha} \subseteq \mathrm{Sub}(L)$, then $\alpha = \beta$.*

Proof By induction on the length of the shortest string w in $\bar{\alpha}$. If this is 1 then clearly $\alpha = [w] = \beta$. Inductive step: suppose $\alpha = \langle A_1 \dots A_m \rangle$ and $\beta = \langle B_1, \dots B_n \rangle$. Since $\bar{\alpha} \subseteq \bar{\beta}$, we know by Lemma 29 that there must be some i such that $A_1 \dots A_i \subseteq B_1$ and similarly, since $\bar{\beta} \subseteq \bar{\alpha}$, there must be some j such that $B_1 \dots B_j \subseteq A_1$. Consider the shortest string $w \in \bar{\alpha}$. This means that $w = a_1 \dots a_m = b_1 \dots b_n$, where $a_k \in A_k, b_k \in B_k$. This means that all of the a_k, b_k are the shortest strings in their respective classes.

Suppose $a_1 \neq b_1$. Without loss of generality assume that $|a_1| > |b_1|$. This implies that $a_1 = b_1 s$, for some s. Now as we have seen, $A_1 \supseteq B_1 \dots B_i$, so $s \equiv b_2 \dots b_i$, by substitutability. If $|s| > |b_2 \dots b_i|$ then $a'_1 = s b_2 \dots b_i$ would be an even shorter element of A_1. If $|s| < |b_2 \dots b_i|$ then $b_1 s b_{i+1} \dots b_n$ would be a shorter element of $\bar{\beta}$ (using the fact that $\bar{\beta} = [\bar{\beta}]$. So $s = b_2 \dots b_i$ and $a_1 = b_1 \dots b_i$. This means that $a_2 \dots a_m = b_{i+1} \dots b_m$.

Pick an $a' \in A_1$ which does not start with an element of B_1 (which exists by Lemma 28). Consider $w' = a' a_2 \dots a_m$ which must also be equal to $b'_1 \dots b'_n$, where $b'_k \in B_k$ as before.

So a' must be a prefix of b'_1 which means that $a' a_2 \dots a_j = b'_1$ by substitutability and so $a_{j+1} \dots a_m = b'_2 \dots b'_n$. So $|b'_2 \dots b'_n| = |a_{j+1} \dots a_m| < |a_2 \dots a_m| = |b_{i+1} \dots b_n| < |b_2 \dots b_n|$, which is a contradiction since $b_2, \dots b_n$ are the shortest strings in $B_2 \dots B_n$. So $a_1 = b_1$ and $A_1 = B_1$. By Lemma 31 and by induction this means that $\alpha = \beta$. ■

We now prove the 'fundamental lemma' of substitutable languages.

Lemma 33 *Every non-zero non-unit congruence class has a unique prime factorisation.*

Proof We show that every congruence class can be represented as a product of primes; uniqueness then follows immediately by Lemma 32. Base case: the shortest string in X

of length 1. (X is not the unit, so we know it is not of length 0). Then it is prime, and can be represented uniquely as a product of 1 prime, itself. Inductive step: suppose X is a congruence class whose shortest string is of length k. If X is prime, then again it is uniquely representable so suppose it is not prime, and there is at least one decomposition into two congruence classes Y, Z. Y, Z must contain strings of length less than k and so by the inductive hypothesis, Y and Z are both decomposable into sequences of prime congruence classes, $Y = Y_1 \dots Y_i$ and $Z = Z_1 \dots Z_j$ so $X = Y_1 \dots Y_i Z_1 \dots Z_j$. ■

Lemma 34 *Suppose N is a prime and α, γ are nonempty sequences of primes such that $N \to \gamma\alpha$ is a valid production. Then α is the prime decomposition of $[\bar{\alpha}]$.*

Proof By induction on the length of α. The base case where α is of length 1 is trivial by the definition of a prime decomposition. Inductive step: Let β be the prime decomposition of $[\bar{\alpha}]$. Clearly $\bar{\alpha} \subseteq \bar{\beta}$ and so by Lemma 29 we know that there is some j such that $A_1 \dots A_j \subseteq B_1$. If $j > 1$ then this would mean that the rule was pleonastic and thus not valid, therefore $j = 1$ and so $A_1 = B_1$; the result follows by induction. ■

Lemma 35 *$G_*(L)$ only has a finite number of valid productions.*

Proof Let n is the number of primes in the language L. Suppose we have two valid productions $N \to A\alpha$ and $N \to A\beta$, where N, A are primes and α, β are sequences of primes. Therefore by Lemma 34 $\alpha = \beta$, which means that there can be at most one production for each pair of primes N, A; therefore the total number of branching productions is at most n^2. ■