Learning Trees from Strings: A Strong Learning Algorithm for Some Context Free Grammars

Alexander Clark
Department of Philosophy,
King's College London,
The Strand,
London WC2R 2LS
alexc@cs.rhul.ac.uk

Abstract

Standard models of language learning are only 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. We take as our starting point a simple learning algorithm for substitutable context-free languages, 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; the first such result.

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, 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. As far as we are aware this is the first algorithm for learning trees from strings which has any sort of theoretical guarantee.

It is a sound methodological principle to try to solve the simplest problems first. Through a variety of historical accidents grammatical inference algorithms for context-free grammars were not developed at the same time as parsing algorithms (like the CKY algorithm); the current paper therefore has a somewhat anachronistic flavour.

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.

1.1 Unsupervised parsing in Computational linguistics

A related but different task is that of unsupervised parsing from natural language corpora such as exemplified by Cohn et al. (2010).

We can contrast our approach by considering the problem of (supervised) parsing in general. Polynomial algorithms for recognising and parsing context-free grammars were first presented in the 1960s (Kasami, 1965): the problem there is given a grammar G and a string w to determine whether w is in the language defined by G and if so to produce the set of parse trees for that string, typically as a forest or acyclic CFG. Modern statistical parsers (Collins and Koo, 2005) on the other hand try to find the most likely parse for a given string given a treebank annotated with one parse tree for each string; this is a different problem because the grammars are normally highly ambiguous, and disambiguating requires semantic or real world knowledge, which is typically implicitly replaced by lexical information. Such statistical parsers have at their core some probabilistic modification of one of the original symbolic parsing algorithms (Klein and Manning, 2005). The underlying core grammar is implicitly defined by an annotation manual, which specifies the labels and sets of allowable structures that can be applied to a corpus. The treebank is then produced by native speakers who manually annotate the corpus using the guidelines in the annotation manual.

In unsupervised parsing, the learning problem is to try to recover the most likely parse, but without using the annotations on the training data. Thus it combines two different problems: the first is the problem of inducing a set of labels and allowable structures, and the second is the statistical task of identifying which of these structures is most likely.

As Cohn et al. (2010) say

> Inducing a grammar from text has proven to be a notoriously challenging learning task despite decades of research.

We feel that part of the reason for the particular difficulty is that it conflates two tasks which are intrinsically different. In contrast here, we try to solve only the first problem but in a theoretically well founded way.

In this paper we approach this problem from a very different direction to that of Cohn et al. (2010). Rather than trying to learn from natural language corpora using nonparametric Bayesian approaches, which lack any theoretical guarantees when used with bounded computational resources, we try to develop a well-founded learning algorithm with theoretical guarantees for simple artificial problems. While this seems an obvious research strategy it has not been followed through before, because there are formidable technical and theoretical obstacles to this task; indeed the first efficient context free grammar learning algorithms with theoretical guarantees are still very recent (Clark and Eyraud, 2007).

1.2 Linguistics

The notions of weak and strong generation are fundamental in the field of theoretical linguistics. [1] 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 involved with which model is appropriate for studying linguistics, which questions depend indeed 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 capacity and as a result take the weak results to be largely irrelevant (Berwick et al., 2011). 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

[1]We use the term 'theoretical linguistics' non-standardly to refer to mathematical linguistics and formal computational linguistics (Bar-Hillel, 1963) .

these can be observed, together with some information about the available semantic interpretations.

We can define corresponding notions of weak and strong learning[2]. 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).

We do not consider in this paper the problem of learning when the input to the learner are *trees* (see (Drewes and Högberg, 2003; López et al., 2004; Sakakibara, 1990, 1992)); we consider only the problem where the learner has access only to the strings, but must infer the structure nonetheless, a much harder problem.

Weak learning of context free grammars and richer formalisms has made significant progress in recent years (Clark and Eyraud, 2007; Yoshinaka, 2011, 2012; Yoshinaka and Kanazawa, 2011) 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 precisely in Section 3, and the associated learning models in Section 4. This 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.

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 PCFGs that define the same set of distributions; in other words PCFGs are not identifiable from strings (though they are from parse trees (Chi, 1999)), and in contrast to discrete HMMs which are (Petrie, 1969). See Hsu et al. (2013) for discussion. Given a set of parse trees, one can learn a good distribution using for example spectral methods(Cohen et al., 2012), but this leaves the problem of learning the trees unsolved, as in the non-probabilistic case.

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

[2] 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).

substitutable context free languages defined in Clark and Eyraud (2007), which we recall in Section 5, 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 6, present an algorithm in Section 7, and prove its correctness and efficiency in Section 8. An appendix contains some detailed proofs of various technical lemmas regarding the properties of the languages we consider in this paper.

2 Notation

We use standard notation.Σ is a finite nonempty set of atomic symbols. Σ^* is the set of all finite strings over Σ. We denote the empty string by λ. The set of nonempty strings is $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. We write $|u|$ for the length of a string. We use the symbol $=_\lambda$ to say that two languages are equal if we ignore the empty string. I.e. $L =_\lambda L'$ iff $L \setminus \{\lambda\} = L' \setminus \{\lambda\}$.

A language L is any subset of Σ^*. Given two sets of strings $M, N \subseteq \Sigma^*$ 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 $\text{Sub}(D)$ to be the set of nonempty substrings of elements of D.

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

Given a non zero sequence of sets of strings $\alpha = \langle X_1, \ldots, X_n \rangle$ we write $\bar{\alpha}$ for the concatenation i.e. $\bar{\alpha} = X_1 \cdot \cdots \cdot X_n$.

The set of all substrings of a language L is denoted $\text{Sub}(L) = \{u \in \Sigma^+ : \exists l, r \in \Sigma^*$ such that $lur \in L\}$ (notice that the empty word does not belong to $\text{Sub}(L)$). 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.

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

$$D_L(u) = \{(l, r) \mid lur \in L\}$$

.

We say that $u \equiv_L v$ iff $D_L(u) = D_L(v)$. This is the syntactic congruence. This 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. Some languages do not have a zero. We are mostly concerned with nonzero 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 in this case, 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 $\bar{\alpha}$.*

Clearly $[\bar{\alpha}] \supseteq \bar{\alpha}$. One crucial question to which we will return later relates to the conditions under which this inclusion is strict or not: i.e. when does $[\bar{\alpha}] = \bar{\alpha}$? If α is of length one this is always true, but when it is of length greater than one, sometimes it will be true and sometimes not.

We write $\Sigma^*/\equiv_L$ for the set of congruence classes of L. 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 nonempty distribution then $u \doteq_L v$, but the converse is clearly not true. Two words can be weakly substitutable without being congruent.

A language L is *substitutable* if for all $u, v \in \Sigma^+$ if $u \doteq_L v$ then $u \equiv_L v$. In other words, for any two nonempty 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 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 ϵ-productions, (productions with a right hand side of length 0) and as a result the languages we consider will not contain the empty string.

We define the standard notion of single-step derivation as $\Rightarrow$ and define $\overset{*}{\Rightarrow}$ as the reflexive transitive closure of $\to$. We define $\mathcal{L}(G, N) = \{w \in \Sigma^* \mid$

$N \overset{*}{\Rightarrow} w\}$. We define $\mathcal{L}(G) = \bigcup_{S \in I} \mathcal{L}(G, S)$. Using a set of start symbols rather than a single one 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.

A regular grammar is a context-free grammar where every production is of the form $N \to aP$ or $N \to a$. A deterministic regular grammar is a grammar where if we have productions $N \to aP$ and $N \to aQ$ then $P = Q$. Recall that for every regular language we can find a deterministic regular grammar that is weakly equivalent and that there is a minimal deterministic regular grammar for each regular language (that does not include λ), where the nonterminals correspond to the Myhill-Nerode classes, which is unique up to isomorphism.

3 Trees

We define sets of structural descriptions for strings as being trees: we consider trees to be singly rooted ordered trees. We do not want to use a ranked alphabet – accordingly our theory is best viewed as based formally on Gorn-style tree domains, but we omit the details as they are not essential, and we assume the notion of a tree is pretheoretically clear.

Let V be a set of labels. Then a structural description (SD) is a tree with at least two nodes, whose leaf nodes are labeled with elements of Σ, and whose nonleaf nodes are labeled with elements of V. We assume that a SD always has at least 2 nodes: this means that we cannot have satisfactory parse trees for the empty string; a parse tree for a string of length 1 will have at least 2 nodes: a single leaf node, a root node, and possibly some additional non-branching non-leaf nodes in between.

We write $\mathbb{T}(V, \Sigma)$ for the set of trees whose leafs are labeled with elements of Σ and whose non leaf nodes are labeled with elements of V.

Given a function $\phi : V \to W$, we extend this to a function $\mathbb{T}(V, \Sigma) \to \mathbb{T}(W, \Sigma)$ in the natural way (i.e. we just relabel every node labeled with $N \in V$ with $\phi(N) \in W$.). For $w \in \Sigma^+$, we write $\mathbb{T}(V, w) \subseteq \mathbb{T}(V, \Sigma)$ for the set of trees whose boundary taken left to right is w. If we have a tree $t \in \mathbb{T}(V, w)$ and a tree $t' \in \mathbb{T}(V, w)$ we say that they are structurally congruent if we take a single element set $\{0\}$ and define $\phi : V \cup V' \to \{0\}$ then $\phi(t) = \phi(t')$.

Given a CFG, $G = \langle \Sigma, V, I, P \rangle$, and a string $w \in \Sigma^+$ we write $\mathbb{P}(G, w) \subseteq$

$\mathbb{T}(V, w)$ for the set of parse trees for w, and where each local tree corresponds to a production in P. We consider parse trees also for string generated from nonterminals that are not in I; so we might have parse trees for strings that are not in the language. Clearly $\mathbb{P}(G, w)$ is nonempty if and only if $w \in \mathcal{L}(G, A)$ for some $A \in V$.

We say that two sets of trees $X \subseteq \mathbb{T}(V, \Sigma)$ and $X' \subseteq \mathbb{T}(V', \Sigma)$ are isomorphic if there is a bijection between them induced by a relabeling function $\phi : V \to V'$.

We say that two CFGs, G and G' are weakly equivalent if $\mathcal{L}(G) = \mathcal{L}(G')$. We say that they are isomorphic if there is a bijection between the sets of nonterminals and this extends to a bijection between the productions. In other words they are identical up to a relabeling of nonterminals. We say $G \cong G'$ is this is the case. Clearly if two grammars are isomorphic then they are weakly equivalent.

We say that G and G' are strongly equivalent iff for all $w \in \Sigma^+$, $\mathbb{P}(G, w)$ is isomorphic to $\mathbb{P}(G', w)$.

We can see that two trim CFGs with no ϵ-productions are strongly equivalent if and only if they are isomorphic. There is therefore no interesting theoretical difference, in the case of CFGs, between these two concepts.

There is a weaker notion of structural equivalence of CFGs (Paull and Unger, 1968), in the literature, but this is too weak. This involves an equivalence of unlabeled tree structures. In We need an equivalence of the labels as well – the tree structures on their own are not sufficient. Berwick et al. (2011) argue for what we can call cardinality equivalence: G and G' are equivalent wrt cardinality if for every w, $|\mathbb{P}(G, w)| = |\mathbb{P}(G', w)|$. Clearly cardinality equivalence is intermediate between isomorphism and weak equivalence; but it seems too weak; indeed it seems clear that Berwick et al. (2011) only intend this as a necessary condition for a learner. Accordingly we will focus on the case of isomorphism, denoted by $\cong$.

Note that for other formalisms the notions of isomorphism and strong equivalence might be quite different. For example, we could have as our formalism the class of regular tree grammars. The set of yields of a regular tree language is a context free set – and indeed the derivation trees of context free grammars are regular tree languages. However we may two very different (non-isomorphic) regular tree grammars that define the same tree language, and are therefore in our sense strongly equivalent.

If we restrict ourselves to trim CFGs, then there are three relevant progressively weaker equivalence relations: equality, isomorphism and weak equivalence. If two CFGs are equal they are clearly isomorphic and if they are isomorphic they are clearly weakly equivalent.

Proposition 1. *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. Note that this is GI-complete: as hard as the problem of graph isomorphism (Read and Corneil, 1977; Zemlyachenko et al., 1985). We may not be able to do this efficiently for general CFGs.

Proposition 2. *(Ginsburg, 1966) Given two* CFG*s, G and G' it is undecidable whether $\mathcal{L}(G) = \mathcal{L}(G')$.*

Looking ahead we will be trying to find learning models, where we are required to find a grammar that is isomorphic to the target and not just weakly equivalent.

4 Learning models

We start by reviewing the basic theory of learnability using the Gold identification in the limit paradigm (Gold, 1967). For simplicity of presentation we consider only the model of given text – where the learner is provided with positive data only.

The standard definition of weak learnability in the Gold style is as follows. We assume a class of representations $\mathcal{R}$. For a $G \in \mathcal{R}$ we can define the language defined by G as $\mathcal{L}(G)$; This is a subset of Σ^*. For simplicity, we will assume that all grammars define nonempty languages.

A presentation of a nonempty 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 an polynomially computable function from sequences of positive examples to elements of $\mathcal{R}$. That is to say A is a function from elements of $\langle w_1, \ldots, w_n \rangle$ to $\mathcal{R}$.

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 $\mathcal{R}$, $A(T_1), A(T_2), \ldots$. These are hypothesis grammars. Implicitly this gives us a sequence of languages that are hypothesized by the learner. $\mathcal{L}(A(T_1)), \ldots$. We call these the hypothesized languages. We will denote $A(T_i)$ by G_i, the ith hypothesis output by the learning algorithm.

Consider a sequence of grammars $G_1, G_2, \ldots$ and a target grammar 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.

WBC There is an N such that for all $n > N$, $L(G_n) = L(G_*)$. (Weak BC convergence)

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

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

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

For each of these 4 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 grammars $\mathcal{G}$. 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).

For CFGs, we can decide whether $G_i \cong G_i$, but it is in general undecidable whether $L(G_i) = L(G_j)$. If A is an algorithm that SBC learns a class $\mathcal{G}$ then we can convert this into an algorithm for SGOLD learning, since we can decide whether two grammars are isomorphic. Note that this is not entirely straightforward if we are interested in polynomial learners, since the test of isomorphism may not be polynomially computable.

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_1 of grammar G produces a grammar G_1 and on presentation

T_2 of the same grammar, produces a grammar G_2, where G_1 and G_2 are weakly equivalent ($L(G_1) = L(G_2)$ but not isomorphic. That is to say, the structural descriptions produced by the grammar will depend on the precise sequence of data that the learner has seen; this is undesirable in some circumstances.

Once could 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 exactly one grammar for each language, and we can compute that, then we could find a strong algorithm. For example one might try to define the smallest grammar that weakly generates a given language.

$$G_{min}(G_i) = \text{argmin}_G\{\|G\| \mid L(G) = L(G_i)\}$$

This is indeed the basis for the minimal DFA approach – the unique up to relabeling minimal DFA that generates a given language. Note that we have restricted ourselves to the class of DFAs not of all FAs. The minimal DFA for a language can be found quite easily. The minimal FA is quite hard to find: and the minimal CFG for a given context-free language will in general be undecidable. We could therefore try to restrict ourselves to some suitable subclass of the class of CFGs that has this property. However, it is not necessarily going to be the case that there is a unique minimum, and the problem of defining precisely a suitable size measure is problematic. In the case of a DFA, we can consider just the number of states. For a CFG, given the variable size of productions it is less straightforward. For example, if we restrict ourselves to grammars in Chomsky normal form (CNF), we find multiple minima.

Example 1. *The language $L = \{a^n b^n | n > 0\}$ for example has two distinct minimal* CFG*s in CNF. There is no grammar in CNF with one or two nonterminals. We clearly need at least 3: one that generates a, one that generates b and one start symbol. There are two non-isomorphic minimal grammars with these three nonterminals, each of which has 4 productions. Labeling the nonterminals as S and A and B the productions are*

- $A \to a, B \to b$
- $S \to AB$ *and either* $B \to SB$ *or* $A \to AS$.

Definition 2. *We say that a class of context-free grammars is redundant if it contains two grammars G_1, G_2 such that $G_1 \not\cong G_2$ and $\mathcal{L}(G_1) = \mathcal{L}(G_2)$.*

Proposition 3. *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 IIL even from informed data, that is to say from labelled positive and negative examples; since there are an infinite number of nonisomorphic grammars for every nonempty language.

4.1 Strong results for regular languages

It may appear that this learning paradigm is too strong to get nontrivial results. However, as noted previously regular learning algorithms can already produce results of this type.

For example, consider Angluin (1982)'s famous algorithm for inferring reversible languages. Since it converges to a canonical representation, we can convert it into a strong learning algorithm as follows.

Take a minimal DFA for a regular language L, which is unique up to relabeling. For each state q in the automaton, we construct a state $[[q]]$ in the CFG. The start symbol of the grammar is $[[q_0]]$ where q_0 is the initial state of the automaton. For every transition $p \rightarrow^a q$ we add a production $[[p]] \rightarrow a[[q]]$. For every final state q, we create a production $[[q]] \rightarrow \lambda$. It is easy to see then that an algorithm that learns a minimal DFA for a regular grammar is also a strong learner in our sense, since any two minimal DFAs will give rise to isomorphic CFGs.

Proposition 4. *The class of deterministic regular grammars that define reversible languages can be* SGOLD*-learned using polynomial time.*

Of course, these are not interesting, but they demonstrate that it is not mathematically impossible to find polynomial strong learning algorithms. From the point of view of structural learning they 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.

5 Weak Learning of Substitutable Languages

We now present an algorithm for strongly learning a class of context-free languages that includes some nonregular languages. We recap the Clark and Eyraud (2007) result and explain why it is only a weak result in this sense rather than a strong result. We use a simplified version of the algorithm as presented by Yoshinaka (2008).

Given a finite nonempty set of strings $D = \{w_1, \dots, w_n\}$ 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 then defined to be the CFG $\langle \Sigma, V, S, P_L \cup P_B \cup P_U \rangle$ as shown in the pseudocode in Figure 1. The sets P_L, P_B and P_U are the sets of lexical, branching and unary productions respectively.

Algorithm 1 $\mathcal{C}_1$

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

Given for example a set $D = \{c, acb\}$, we have $\mathrm{Sub}(D) = \{a, b, c, ac, cb, acb\}$, and corresponding sets:

- $V = \{[[a]], [[b]], [[c]], [[ac]], [[cb]], [[acb]]\}$
- $I = \{[[c]], [[acb]]\}$
- $P_L = \{[[a]] \to a, [[b]] \to b, [[c]] \to c\}$
- $P_B = \{[[ac]] \to [[a]][[c]], [[cb]] \to [[c]][[b]], [[acb]] \to [[ac]][[b]], [[acb]] \to [[a]][[cb]]\}$
- $P_U = \{[[c]] \to [[acb]], [[acb]] \to [[c]]\}$

As can be verified this grammar 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 as in Algorithm 2. This will give a BC learner for the class of substitutable CFGs.

Alternatively we can convert this into a Gold learner, by only changing the hypothesis when the hypothesis is demonstrably too small. This means

Algorithm 2 $\mathcal{A}_1$

```
Data: A sequence of strings $w_1, w_2, \dots \in L_*$
Result: A sequence of CFGs $G_1, G_2, \dots$
let $D := \varnothing$;
for $n = 1, 2, \dots$ do
  let $D := D \cup \{w_n\}$;
  output $\hat{G} = \hat{G}(D)$ as $G_n$;
end for
```

that once we have a weakly correct hypothesis the learner will no longer change its output. This simple modification is shown in Algorithm 3.

Algorithm 3 $\mathcal{A}_2$

```
Data: A sequence of strings $w_1, w_2, \dots \in L_*$
Result: A sequence of CFGs $G_1, G_2, \dots$
let $D := \{w_1\}$; $\hat{G} = \hat{G}(D)$;
output $\hat{G}$ as $G_1$;
for $n = 2, 3, \dots$ do
  let $D := D \cup \{w_n\}$;
  if $D \nsubseteq \mathcal{L}(\hat{G})$ then
    let $\hat{G} := \hat{G}(D)$;
  end if
  output $\hat{G}$ as $G_n$;
end for
```

Proposition 5. *Algorithm 3 identifies in the limit with polynomial computation the class of substitutable context free languages.*

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 L_* starts $\{a, acb, \dots\}$ then the learner will converge in two steps to the grammar shown above. 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.

6 The syntactic structure of substitutable languages

In this section we use the ideas of Clark (2011) as the basis for our idea of syntactic structure. 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 has a simple intrinsic structure that we can learn efficiently.

We make the following definition:

Definition 3. *A nonzero, nonunit congruence class X is prime if, for any two congruence classes Y, Z such that $X = Y \cdot Z$ then either Y or Z is a unit.*

In other words a class is not prime if it can be decomposed into the concatenation of two other congruence classes. We have stipulated that the unit congruence class is not prime, and that the zero congruence class is also not prime. This 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, then if we have a nonterminal that corresponds to a non-prime class N, then that means that we can decompose this into two classes P, Q such that $N = PQ$. In that case we can replace every occurrence of N on the rhs of a rule by the pair PQ; assuming that P and Q can be represented adequately, nothing will be lost by this assumption. Thus non-prime congruence classes can always be replaced by a sequence of prime congruence classes. Therefore we can limit our attention to the primes which informally are those where "the whole is greater than the sum of the parts". If we have two congruence classes P and Q such that the congruence class of PQ is just PQ then the rule is in a sense vacuous; the congruence class is no greater than the simple concatenation of the two sets of strings. More algebraically, we can think of the primes as representing the points where the free monoid and the syntactic monoid differ in interesting ways.

Example 2. *Consider the language $L = \{a^n cb^n \mid n \geq 0\}$. This language is not regular and therefore has an infinite number of congruence classes. The congruence classes are as follows: L is a congruence class. $\{\lambda\}$ is a congruence class with just one element; this is the unit congruence class. We have the zero congruence class which consists of all strings that have empty distribution: those that are not a substring of any string in L; this is infinite and contains $ab, bc, cc, bca, \ldots$ and so on. We also have an infinite number of congruence classes of the form $\{a^i\}$ for any positive i. These are congruence classes that are all singleton (have only one element). So $[a] = \{a\}$. Similarly we have classes like $\{b^i\}$ for any positive i. There are exactly 3 prime congruence classes, which are $[c] = L$, $[a] = \{a\}$ and $[b] = \{b\}$. All of the other congruence classes, such as $[aa]$ are not prime as they can be divided into the concatenation of two other classes. For example $[aa] = [a] \cdot [a]$, $[aac] = [a] \cdot [ac]$.*

Example 3. *Consider the language $L = \{a^n b^n \mid n > 0\}$. This language is not regular and has a countable infinite set of congruence classes, but there are exactly 3 prime congruence classes. $[ab] = L$, $[a] = \{a\}$ and $[b] = \{b\}$.*

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

Example 4. *Consider the finite singleton language $L = \{ab\}$. This language has 5 congruence classes: $[a], [b], [ab], [\lambda], [aa]$. The first 4 are all singleton sets. The only one which is not prime is $[ab] = \{ab\} = [a][b]$.*

Proposition 6. *For every $a \in \Sigma$, for any language L, if $[a]$ is nonzero 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. Therefore there must be two strings u, v such that $u \in X$ and $v \in Y$ and $a = uv$. Since a is of length 1, there are only two possibilities: either $u = a$ and $v = \lambda$ or $u = \lambda$ and $v = a$. This means that either $X = [a]$ or $Y = [a]$ and therefore a is prime. □

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

Definition 4. *Let $\mathcal{L}_{sc}$ be the set of all languages which are substitutable, nonempty, 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 being a good example – we need to restrict ourselves weakly in some way. For the weak learner, the restriction was to context free grammars. Here we will take a different condition: 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 has a finite number of primes as it has a finite number of congruence classes. Note that this is a purely language theoretic property, as we do not appeal to any properties of representations.

Not all substitutable context free grammars have a finite number of primes, as this next example shows.

Example 5. *Consider the language $L = \{c^i ba^i b \mid i > 0\} \cup \{c^i de^i d \mid i > 0\}$. 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 can be seen to be prime.*

Definition 5. *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 $X = \bar{\alpha}$ i.e. $X = X_1 \cdot X_2 \cdot \cdots \cdot X_k$, the concatenation of the classes.*

Clearly any prime congruence class X has the trivial length one prime decomposition $\langle X \rangle$. We are interested in the case where a congruence class is not prime, and where the decomposition consists of more than one prime.

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 1. *Every nonzero non-unit congruence class in a language in $\mathcal{L}_{sc}$ can be represented uniquely as a product of prime congruence classes.*

For proof see Lemma 15 in the appendix. Note that this is not the case in general for languages which are not substitutable, as the following example demonstrates.

Example 6. *Let $L = \{abcd, apcd, bx\}$. Note that L is finite but not substitutable since $p \not\equiv b$. Among the congruence classes are $\{a\}, \{b\}, \{c\}$ $\{ab, ap\}$, $\{bc, pc\}$, $\{abc, apc\}$. Clearly $\{ab, ap\}$, $\{bc, pc\}$ are both prime and $\{abc, apc\}$ is composite and has two distinct prime decompositions $\{ab, ap\}\{c\}$ and $\{a\}\{bc, pc\}$.*

If we restrict ourselves to languages in $\mathcal{L}_{sc}$ then we can assume wlog that the symbols of the grammar correspond to congruence classes. In a substitutable language, a trim context free grammar cannot have nonterminals that generate strings that are not congruent. Similarly, if we have two nonterminals that generate congruent strings, we can 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 symbol S so that we can deal with the case where L itself is not prime. S will not appear on the right hand side of any rule.

6.1 Productions

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

Definition 6. *A* correct *branching production is of the form $N \to \alpha$ where N is a prime congruence class, α is a sequence of at least 2 primes and $N \supseteq \bar{\alpha}$, or equivalently $N = [\bar{\alpha}]$.*

A correct lexical production is one of the form $N \to a$ where $a \in N$ and $a \in \Sigma$, or equivalently when $N = [a]$.

Note that if $a \in N$ and N is a non-unit congruence class then N will always be prime.

Example 7. *Consider the language $L = \{a^n c b^n \mid n \geq 0\}$. This has primes $[a], [c], [b]$. 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 lhs, and are $[c] \to [a][c][b]$, $[c] \to [a][a][c][b][b]$ and so on. There are infinitely many all of the form $[c] \to [a]^i[c][b]^i$.*

Clearly in the previous example we want to rule out productions like $[c] \to [a][a][c][b][b]$ since they 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 rhs of the production $[a][a][c][b][b]$ is too long since there is a proper subsequence $[a][c][b]$ which is a prime congruence class, and should be represented just by the prime $[c]$. We use the word *pleonastic* to define this property of being too long, which we define formally now.

Definition 7. *We say that a sequence of primes α is* pleonastic *(too long) if $\alpha = \gamma\beta\delta$ for some γ, δ, which are sequences of primes, at least one of which is nonempty, and $[\bar{\beta}]$ is a prime, and $|\beta| > 1$. (i.e. β is a sequence of more than one prime)*

Note that for a language in $\mathcal{L}_{sc}$ if $[\bar{\beta}]$ is a prime and $|\beta| > 1$, then $[\bar{\beta}] \supsetneq \bar{\beta}$ since if they were equal then β would not be a prime.

Definition 8. *We say that a correct production $N \to \alpha$ is* pleonastic *if α is pleonastic.*

A correct production/local tree 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, and that $N = [\bar{\alpha}]$*

For any string in a prime congruence class $w = a_1 \dots a_n$, $a_i \in \Sigma$ we can construct a correct production $[w] \to [a_i] \dots [a_n]$: such productions may in general be pleonastic because there will be substrings that can be represented by prime congruence classes. From a structural perspective, these local trees are too shallow as they flatten out relevant parts of the structure of the string. Nonetheless we can find a finite set of valid local trees that will generate this, as the following lemmas establish.

Proposition 7. *If $L \in \mathcal{L}_{sc}$ then there are a finite number of valid productions.*

For proof see proof of Lemma 17 in the appendix.

6.2 Canonical Grammars

We will now define canonical grammars for every language in the class $\mathcal{L}_{sc}$. Note that for every language in $\mathcal{L}_{sc}$, L is a single congruence class. That is to say, if $u, v \in L$ then $u \equiv v$ and if $u \in L$ and $u \equiv v$ then $v \in L$.

Definition 9. *Let L be some language in $\mathcal{L}_{sc}$. We define the following grammar, $G^*(L)$. The nonterminals are the set of 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 have the set of productions which is the union of the following three sets:*

- $\{S \to \alpha(L)\}$
- *The set of all valid (i.e. correct and not pleonastic) productions using the prime congruence classes.*
- *For each terminal symbol that occurs in the language, the production* $X \to a$ *if* $a \in X$

Note that this is a well defined, unique finite CFG for every language in $\mathcal{L}_{sc}$. We now show that this is a grammar for the language, in the sense that it generates the language it should.

Lemma 2. *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 $X_i \overset{*}{\Rightarrow} u_i$ for some $1 \leq i \leq n$ and $X_0 \to X_1 \ldots X_n$ is a production in the grammar. Then by the inductive hypothesis $u_i \in X_i$ and by the correctness of the productions, $u_1 \ldots u_n \in X_0$. $\square$

Lemma 3. *Suppose* $L \in \mathcal{L}_{sc}$, *then suppose* X *is a prime, and* $w \in X$. *Then* $X \overset{*}{\Rightarrow}_{G_*(L)} w$.

For proof see proof of Lemma 9 in the appendix.

Proposition 8. *For any* $L \in \mathcal{L}_{sc}$, $\mathcal{L}(G_*(L)) = L$.

Proof. Immediate by the two preceding lemmas. $\square$

Definition 10. *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 4. $\mathcal{G}_{sc}$ *is not redundant.*

Proof. Suppose we have two grammars G_1, G_2 in this class; then if they are weakly equivalent, they will be isomorphic. $\square$

Proposition 9. *For all* $G \in \mathcal{G}_{sc}$, G *is an NTS grammar.(Boasson and Sénizergues, 1985; Clark, 2006)*

Proof. Suppose $X \overset{*}{\Rightarrow} \alpha\beta\gamma$ and $Y \overset{*}{\Rightarrow} \beta$. Then $X \to \alpha Y \gamma$ is a correct production and therefore is derivable. $\square$

7 An algorithm for strong learning

We now present an algorithm that takes a finite sample of strings and produces a grammar in polynomial time. We will demonstrate then in the next section 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.

1. We start by using Algorithm 1 to construct a context free grammar G^w which will be weakly correct.
2. We then partition $\mathrm{Sub}(D)$ into congruence classes, with respect to our learned grammar G^w.
3. Pick the lexicographically shortest string in each class as the label we will use for the nonterminal.
4. Test to see which of the congruence classes are prime.
5. Decompose each class uniquely into a sequence of primes.
6. Construct valid rules.
7. We then eliminate pleonastic productions from this set of productions.
8. Return a grammar G^s constructed from these productions.

Given a sample D, we construct a grammar in polynomial time using Algorithm 1. This grammar G will only have binary branching rules.

We can perform task 2 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 $lvr \in L_*$. Under the assumption that the grammar is correct we can test this by seeing whether $lvr \in \mathcal{L}(\hat{G})$, using a 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 this is prime or not. Take the shortest element w in the class. Test every possible split of w into nonzero 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 $X \supseteq YZ$ simply because we have not yet observed one of the strings in $X \setminus YZ$. We present the pseudocode for this procedure in Algorithm 4.

Algorithm 4 Test primality

```
Data: A set of strings $X$
Data: A set of disjoint sets of strings: $\mathcal{X} = \{X_1, \ldots, X_n\}$
Result: True or false
for $w \in X$ do
  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
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 $a_1 \ldots a_n$. We convert this into a sequence of primes $[a_1] \ldots [a_n]$. 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 $0, 1, \ldots, n$. We create an arc from $i \rightarrow j$ 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 $N \rightarrow M\alpha$ 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 congruence class Q, take α to be the prime decomposition of Q, test to see if $N \rightarrow M\alpha$ 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 lies in a prime. If no prefix does then the production is not pleonastic and is therefore valid.

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

8 Analysis

We now proceed to the analysis of Algorithm 5. 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.

8.1 Strong convergence

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 1. 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 language/grammar in the class we will converge strongly to a grammar isomorphic to the canonical grammar. We will show in a sequence of lemmas that the algorithm weakly converges; that it will correctly identify the set of all prime congruence classes; that it will converge to the correct label for each of these. and finally that it will converge to the right set of productions. Finally we will conclude that it then never changes its hypothesis.

We first prove the correctness of some parts of the model under the assumption that G^w is correct.

Lemma 5. *Suppose* $X_1, \dots, X_n$ *is a correct partition of* $\text{Sub}(D)$ *into congruence classes. Then if Algorithm 4 returns true when applied to* X_i*, then* $[X_i]$ *is in fact prime.*

Algorithm 5 $\mathcal{A}_{\text{SGOLD}}$

```
  Data: A sequence of strings $w_1, w_2, \dots \in L_*$
  Data: $\Sigma$
  Result: A sequence of CFGs $G_1, G_2, \dots$
  let $D := \varnothing$;
  for $n = 1, 2, \dots$ do
    let $D := D \cup \{w_n\}$;
    $\hat{G} = \hat{G}(D)$;
    Let $C$ be the partition of $\mathrm{Sub}(D)$ into classes;
    Let $Pr$ be the set of primes computed using Algorithm 4 ;
    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 $N \in Pr$ do
      for $M \in Pr$ do
        for $Q \in C$ do
          $R = (N \to M\alpha(Q))$
          if $R$ is correct and valid then
            $P_B = P_B \cup \{R\}$;
          end if
        end for
      end for
    end for
    output$G_n := \langle \Sigma, V \cup \{S\}, \{S\}, P_L \cup P_B \cup P_I \rangle$;
  end for
```

Proof. Suppose $[X_i]$ is not prime: then there are two congruence classes Y, Z such that $[X_i] = YZ$. Consider a string in X_i, w. There must be strings u, v such that $w = uv$ and $u \in Y, v \in Z$. Since $w \in \mathrm{Sub}(D)$, $u, v \in \mathrm{Sub}(D)$. Since the partition of $\mathrm{Sub}(D)$ 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 $\mathrm{Sub}(D)$ is substring closed, we have that $X_i \subseteq X_j X_k$, in which case Algorithm 4 will return false. $\square$

We will first construct a small (polynomially bounded) set of examples such that once the learner has seen this set, it will have converged.

Definition 11. *For a grammar G we define $\chi(G)$ to be the following set of strings (Clark and Eyraud, 2007).*

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

We can now define the characteristic set $\chi(G_*) = \{lwr \mid (N \to \alpha) \in P, (l, r) = c(N), w = w(\alpha)\}$.

In what follows we suppose G_* is a grammar in the $\mathcal{G}_{sc}$, and that $L_* = \mathcal{L}(G_*) \in \mathcal{L}_{sc}$. Suppose T is some presentation of L_*, $T = w_1, \ldots, w_n, \ldots$. We let $D_n = \{w_1, \ldots, w_n\}$, and we will talk about the behavior of the learner at step n.

Since T is a presentation, and $\chi(G_*) \subseteq L_*$ we immediately have that any sufficiently long presentation will include the characteristic set.

Proposition 10. *There is some N such that for all $n > N$, $D_n \supseteq \chi(G_*)$.*

By the correctness of the weak learner we have that:

Lemma 6. *(Clark and Eyraud, 2007) If $D_n \supseteq \chi(G_*)$, then the grammar G_n^w is weakly correct – i.e. $\mathcal{L}(G_n^w) = L_*$.*

Lemma 7. *Suppose $X_1, \ldots, X_n$ is a correct partition of* $\mathrm{Sub}(D)$ *into congruence classes, and $D \supseteq \chi(G_*)$. Then Algorithm 4 returns true when applied to X_i, iff $[X_i]$ is in fact prime.*

Proof. X_i is a finite subset of $\mathrm{Sub}(D)$, 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 5). 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 $\mathrm{Sub}(D)$, which means that since it is a correct partition the algorithm cannot find any pair of classes whose concatenation contains X_i. □

We now show that the strong learner can correctly extract the canonical form.

Proposition 11. *If D_n and D_N both include $\chi(G_*)$, then $G_n^s = G_N^s$ and $G_n^s \cong G_*$.*

Proof. If the data includes the characteristic set, then $\mathrm{Sub}(D)$ will contain at least one element of each prime congruence class. Since G^w is correct, we will have a correct partition of $\mathrm{Sub}(D)$ into congruence classes. By Lemma 5 the algorithm will correctly identify the set of prime congruence classes. Since the characteristic set includes the shortest element of each congruence class, the labels for each nonterminal will be the same. The start symbol is fixed, and there is a unique decomposition of L, so the single start production is fixed.

We just need to show then that we get the correct set of productions. First we show that every valid production will be produced by the algorithm, then we show that every production produced by the algorithm will be valid.

Suppose $N \rightarrow X_1 \dots X_n$ 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 $\langle X_2, \dots, X_n \rangle$, by Lemma 15. Therefore this production will be produced by the algorithm.

Secondly suppose the algorithm produces some production $N \rightarrow X_1, \dots X_n$. We know that this will be valid since $X_2, \dots X_n$ 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.

□

Combining Propositions 10 and 11 concludes the proof of Theorem 1.

8.2 Running time

We first show that this algorithm runs in polynomial time in the size of the data set D. That is to say there is a polynomial p such that the number of steps taken by the algorithm at each point is less than $p(\|D\|)$.

We do this by verifying that each phase of the algorithm is polynomial. This is fairly straightforward. We note first that $|\mathrm{Sub}(D)|$ is polynomial in $\|D\|$. We recall that for a CFG G, the CYK algorithm can parse in polynomial time, in both the size of the grammar and the length of the string.

8.3 Characteristic set

We now show that the learner converges rapidly in the sense that it has a characteristic set which is bounded in cardinality and size using the notion of thickness (Wakatsuki and Tomita, 1993).

For a CFG we define the thickness of a nonterminal to be the length of the shortest string generated by it: $t(G, A) = \min\{|w| \mid w \in \mathcal{L}(G, A)\}$, and the thickness of a grammar t_G to be the maximum thickness of a nonterminal in the grammar $t(G) = \max\{t(G, A) \mid A \in V\}$.

We define the size of a grammar G, $\|G\|$ to be the sum of the number of nonterminals and the lengths of all of the productions.

$$\|G\| = |V| + \sum_{N \to \alpha \in P} |\alpha|$$

.

Given a finite set of strings X we define the size of the set as $\|X\| = \sum_{w \in X} |w|$, in contrast to the cardinality which is simply the number of strings in the set, denoted by $|X|$.

Given a learner A, and a grammar G, we say that a set X is *characteristic* for G if whenever the input data in a presentation of $L(G)$ includes X, the hypothesis output by A is isomorphic to G. That is to say once it has seen this set, it will have the right answer. Clearly by the above results, we have that $\chi(G_*)$ is a characteristic set.

Proposition 12. *$\chi(G)$ is a polynomial characteristic set for G wrt the learner A_{SGOLD}, in the sense that there are polynomials p, q such that for every grammar in the class $\mathcal{G}_{sc}$ there is a characteristic set $\chi(G)$ such that $|\chi(G)| < p(\|G\|)$ and $\|\chi(G)\| < q(\|G\|, t(G))$.*

Proof. Immediate by construction. $|\chi(G)|$ is linear in the number of productions, and the length of each example is linearly bounded by the thickness and the maximum length of the rhs of the rules. □

9 Examples

We have implemented the algorithm presented here. [3] We present the results of running this algorithm on small data sets that illustrate the properties of the canonical grammars for the learned languages. Some of these correspond with desirable properties of structural descriptions for natural language sentences and others do not. 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. We give examples at various levels of language theoretic complexity; we include the input data, the output grammar and some parse trees for some example sentences. 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.

9.1 Finite language

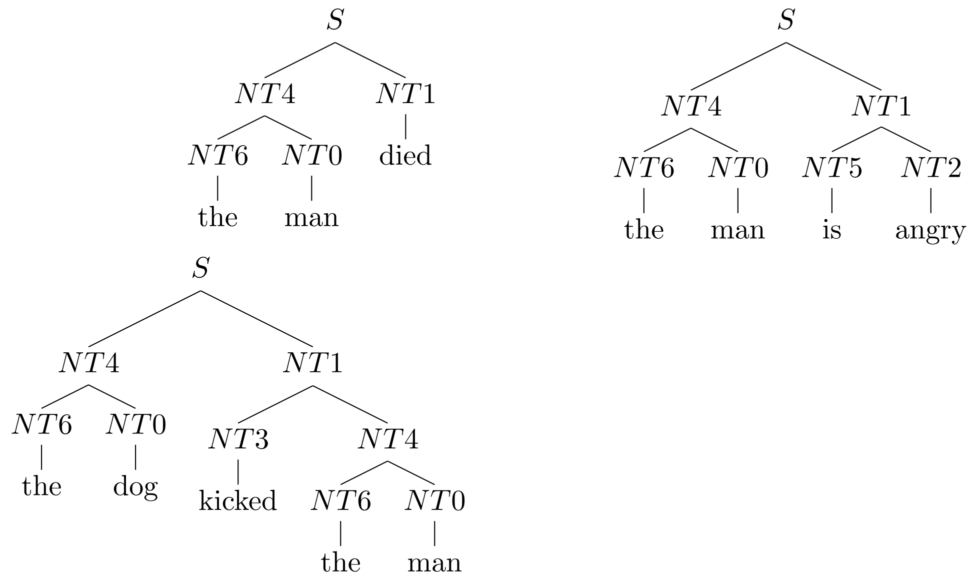
We give the following input data:

1. the man died
2. the dog died
3. the man is angry
4. it died
5. he died
6. the man kicked the dog

The algorithm produces the following grammar, which gives the derivation trees shown in Figure 1.

[3] A Java implementation will be made available on the author's website.

Figure 1: Toy English; sample trees.



S $\rightarrow$ **NT4 NT1**
NT2 $\rightarrow$ **angry**
NT0 $\rightarrow$ **man**
NT1 $\rightarrow$ **NT5 NT2**
NT1 $\rightarrow$ **died**
NT5 $\rightarrow$ **is**
NT0 $\rightarrow$ **dog**
NT4 $\rightarrow$ **NT6 NT0**
NT1 $\rightarrow$ **NT3 NT4**
NT4 $\rightarrow$ **he**
NT6 $\rightarrow$ **the**
NT4 $\rightarrow$ **it**
NT3 $\rightarrow$ **kicked**

9.2 Regular language

We now consider some data that leads to a regular language:

1. one
2. two
3. three

4. one and two

5. two times three

6. one and two times three

Given this input we get the following grammar:
S → **NT0**
NT0 → **three**
NT0 → **NT0 NT1 NT0**
NT0 → **two**
NT1 → **times**
NT1 → **and**
NT1 → **NT1 NT0 NT1**
NT0 → **one**

This CFG defines an infinite regular language: if d is the set { one two, three } and o is the set { times, and } then the grammar defines the language $d(od)^*$. In Figure 2 we show some trees. For sentences length 1 and 3, the grammar is unambiguous. For sentences of length 5, the grammar produces 3 parses: we show all 3 for one example sentence. For sentences of length 7, there are 12 parses and so on.

9.3 Trivial Context-Free Language

Consider the running example of $\{a^n c b^n \mid n \geq 0\}$. In this case the only prime congruence classes are $[a], [b]$ and $L = [c]$. The canonical labels for the primes are therefore a, b, c and we have a grammar with 4 nonterminals $\{S, [[a]], [[b]], [[c]]\}$. The initial production is just $S \to [[c]]$. There are an infinite number of correct productions of the form $[[c]] \to ([[a]])^n [[c]] ([[b]])^n$, of which only one is valid – $[[c]] \to [[a]][[c]][[b]]$. Therefore the full set of productions is

$$\{S \to [[c]], [[c]] \to [[a]][[c]][[b]], [[c]] \to c, [[a]] \to a, [[b]] \to b\}$$

. A characteristic set for this is just $\{c, acb\}$. Given this input data:

1. c

2. a c b

We get the grammar

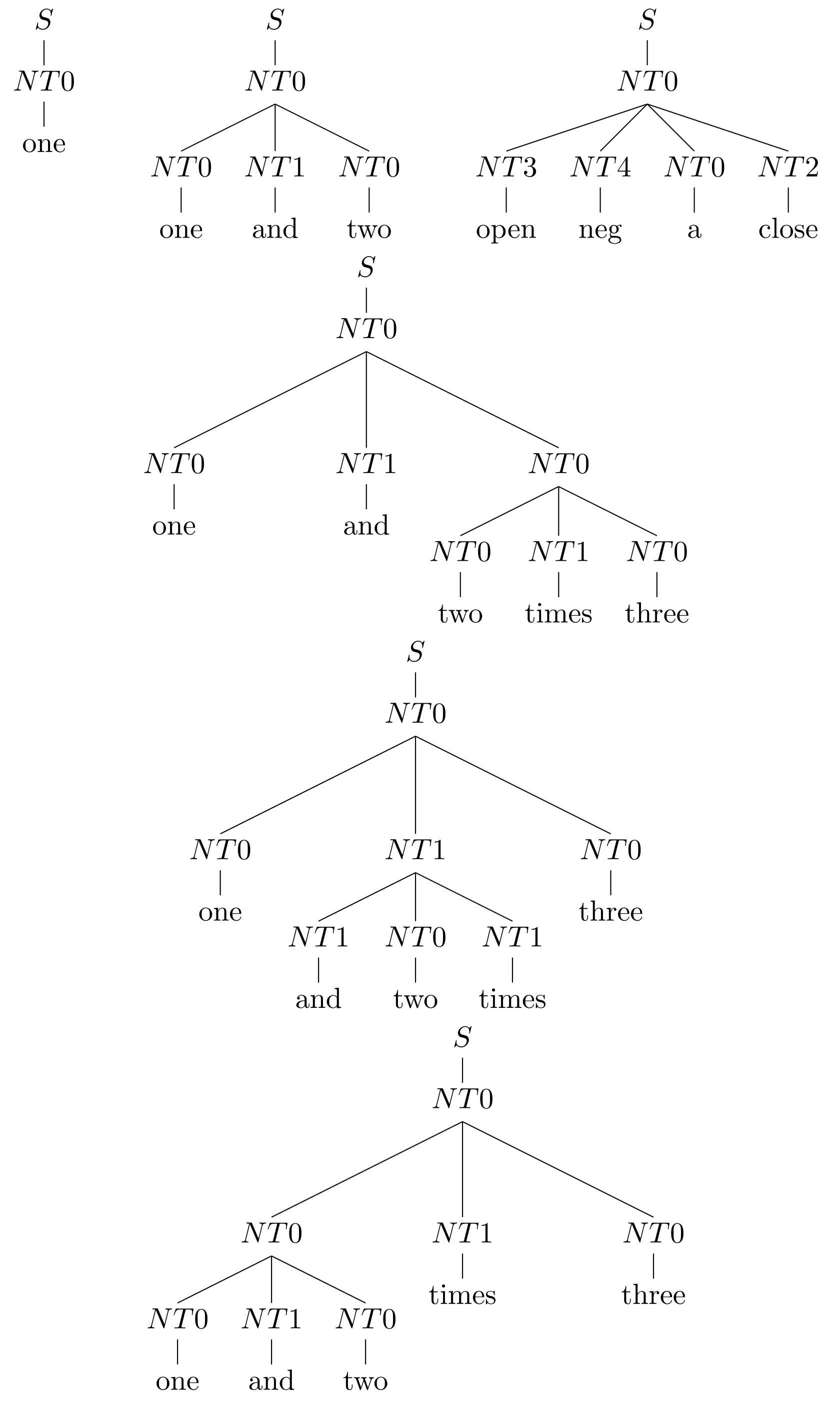
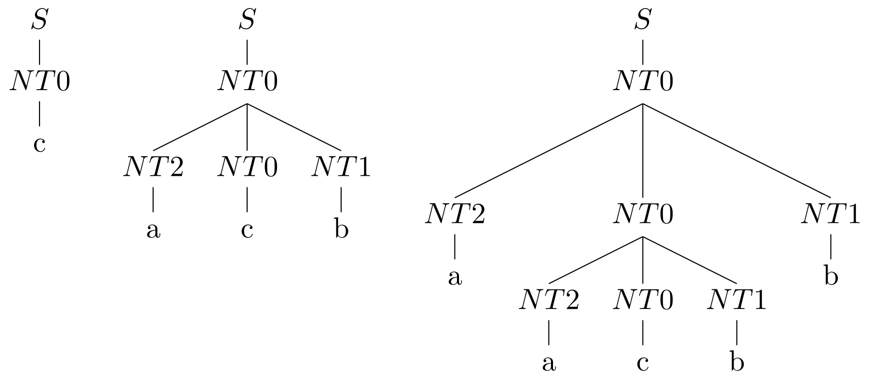


Figure 2: The unique parse trees for expressions of length 1 and 3, and the 3 parse trees for length 5.

Figure 3: Example parse trees for the example $\{a^n cb^n \mid n \geq 0\}$.



S → NT0
NT1 → b
NT0 → c
NT2 → a
NT0 → NT2 NT0 NT1

This defines the language $\{a^n cb^n \mid n \geq 0\}$, which is not a regular language. Figure 3 shows some parse trees for the three shortest strings in the language. This grammar is unambiguous so every string has only one tree.

9.4 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 can define by the CFG: $S \rightarrow A_i$, where A_i are a finite set of sentence symbols. $S \rightarrow (\neg S)$, $S \rightarrow (S \vee S)$, $S \rightarrow (S \wedge S)$, $S \rightarrow (S \Rightarrow S)$ and $S \rightarrow (S \Leftrightarrow S)$. 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 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 case 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 (Enderton, 2001).

Since $(\neg A)$ and $(A \vee A)$ are both formula then we have that $\neg \cong A\vee$, 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 4 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 like so:

1. a
2. b
3. c
4. open a and b close
5. open a or b close
6. open a implies c close
7. open a iff c close
8. open neg a close

This produces the following grammar, which is weakly correct:

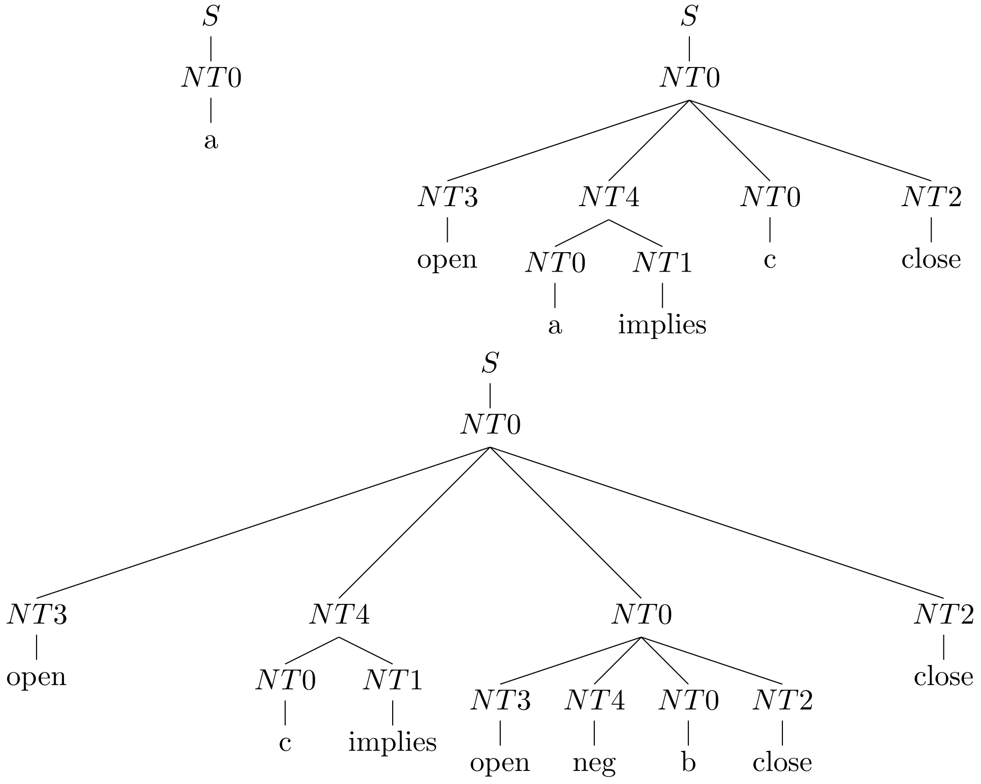
S $\rightarrow$ **NT0**
NT2 $\rightarrow$ **close**
NT4 $\rightarrow$ **neg**
NT0 $\rightarrow$ **a**
NT4 $\rightarrow$ **NT0 NT1**
NT0 $\rightarrow$ **b**
NT0 $\rightarrow$ **NT3 NT4 NT0 NT2**
NT1 $\rightarrow$ **or**
NT0 $\rightarrow$ **c**
NT3 $\rightarrow$ **open**
NT1 $\rightarrow$ **and**
NT1 $\rightarrow$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

This generates one tree for each wff in the language as shown in Figure 4.

9.5 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 slightly excessive input data:

Figure 4: Example parse trees for the sentential logic example. Each example has only one parse tree.



1. a b
2. b a
3. a b a b
4. a b b a
5. b a b a
6. b b a a

The resulting grammar has 10 productions:

S → NT0
NT2 → NT2 NT0
NT0 → NT0 NT0
NT2 → NT0 NT2
NT1 → b
NT1 → NT1 NT0
NT0 → NT2 NT1
NT0 → NT1 NT2
NT2 → a
NT1 → NT0 NT1

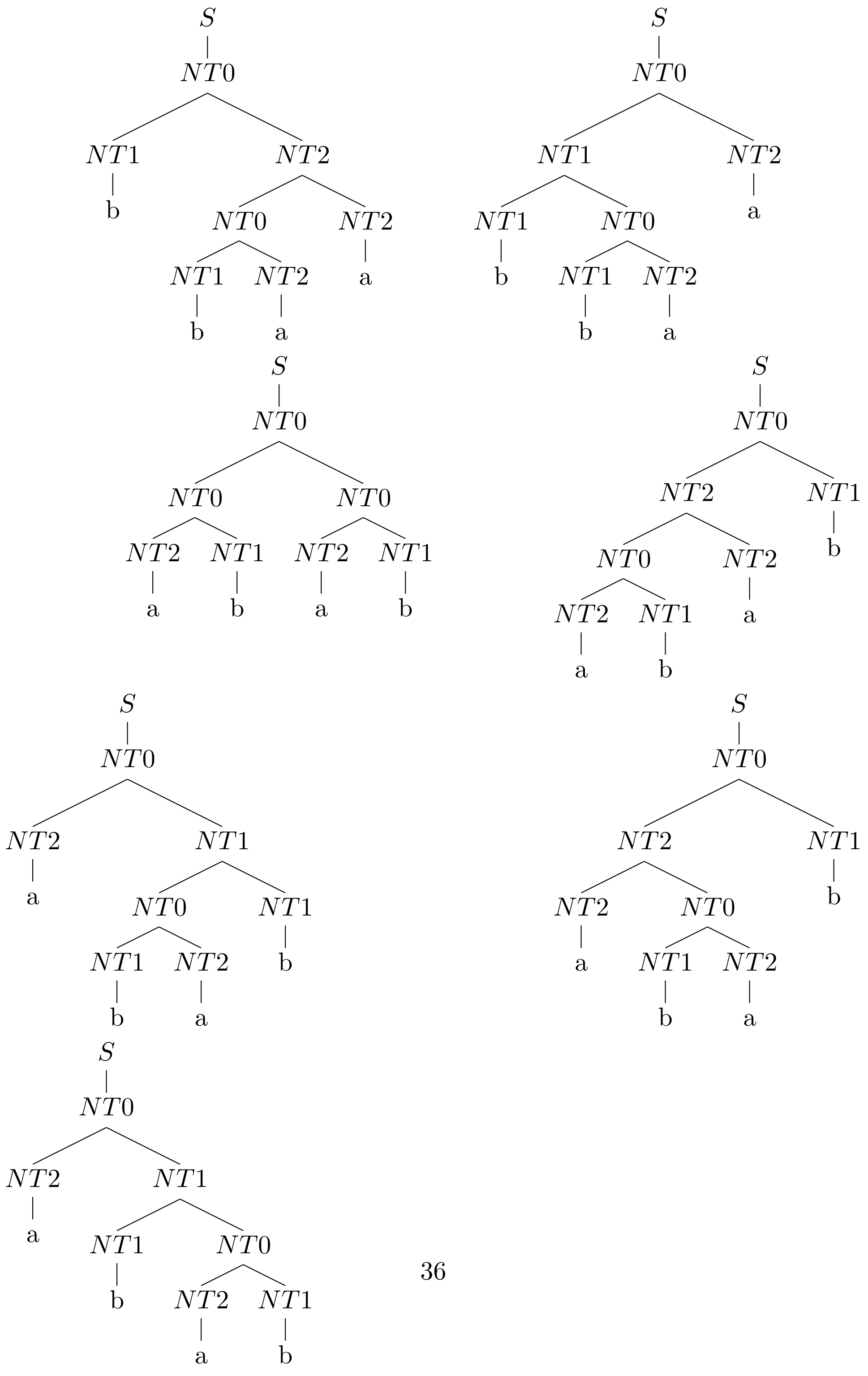
In this case the number of parses depends on properties of the string that are more complex than just the length. For example: the string 'abab' has 5 parses, the string 'abba' has 3 and the string 'aabb' has only 2. Figure 5 shows some trees.

10 Discussion

The main point of this paper is to establish that it is possible to obtain strong learning results for at least some proper CFGs from positive strings alone. To the best of our knowledge this is the first such learning result. The Gold learning model is too onerous, and the class of languages that can be learned is as a result rather limited, but nonetheless includes some interesting examples as we showed in the previous section.

We have defined a canonical grammar for every language in a certain class. It is natural to ask the following rather imprecise question: why should we pick that grammar rather than one of the many other different grammars? It is worth asking and answering these questions now, with this rather trivial class, because it is easier to see the various factors when the

Figure 5: Examples from language O_1; 2 parses for the string 'bbaa' and 5 parses for the string 'abab'



situation is so simple; later, when we expand these results to MCFGs, we will face the same issues but in a more complex setting.

One uninteresting reason is a technical one – because it works. We can, in the case of this definition of a canonical grammar, define a strong learning algorithm and prove its correctness. In this framework, we need to design a strong learning algorithm based on a weak one – thus there must be a strong concordance between the representational assumptions of the weak learner and the strong learner. At the moment, this is the only way we see how to do this.

But there is a more satisfying reason which we now sketch, based on the idea of minimality/simplicity. Given a particular substitutable context free language, L, we can consider the smallest grammar for this language. It is easy to see that any trim CFG for the language L will have non terminals that generate only subsets of congruence classes. Pick some grammar G such that $L(G) = L$. If we have a nonterminal N such that $S \overset{*}{\Rightarrow} lNr$ and $N \overset{*}{\Rightarrow} u$ and $N \overset{*}{\Rightarrow} v$ then $u \equiv_L v$ by substitutability. So for any nonterminal that is reachable, (i.e. such that there are l, r such that $S \overset{*}{\Rightarrow} lNr$), the set of strings $\mathcal{L}(G, N)$ will be a subset of a congruence class.

Moreover, if we have two nonterminals N, M such that $\mathcal{L}(G, N)$ and $\mathcal{L}(G, M)$ are both subsets of the same congruence class then we can merge M and N. We can create a new nonterminal, A. We can create a new grammar by replacing every occurrence of M or N by the new symbol A. This will give us a new context free grammar, G' with one less nonterminal, and we can verify that $L(G')$ is unchanged. Therefore if we are looking for simple grammars for L, then we can assume that the nonterminals of the grammar are in bijection with the nonzero congruence classes of the language.

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. Strong learning is hard – accordingly we decompose it into two problems of rather different flavors. The first is a weak learning result, 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 we can't compute a canonical form for an arbitrary grammar as this will be undecidable; however we can do this for the grammars output by the weak learner which come in a restricted form. We could, in this particular case, find a grammar equal to the target by picking a canonical set of labels, but this is an unnecessarily strong requirement.

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; this allows learnability under a very rigorous learning model.

There is a gap here between the class of languages that the strong learner here can learn and the weak learner of Clark and Eyraud (2007): the weak learner can learn all substitutable CFGs; the strong learner we define here can learn only substitutable languages with a finite number of primes, which is strictly smaller. From one point of view the latter criterion is cleaner – we have two purely language theoretic criteria rather than one language theoretic (substitutability) and one syntactic (context-freeness). Possibly this difference indicates instead that the criterion of being prime is too weak – we need to add additional restrictions to pick out some restricted subset of prime congruence classes to use as nonterminals. We think instead that the problem is deeper – that it lies in the representational assumption of nonterminals being congruence classes which is in any event far too restrictive.

In the sequel we will discuss how this can be applied to a much richer class of languages – a larger subclass of the class of well-nested 2-MCFGs (which are equivalent to tree-adjoining grammars), under a less rigorous learning model where we idealize and allow the learner to use membership queries. 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 richer 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. Any remaining errors are of course my own.

References

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

Bar-Hillel, Y. (1963). Four lectures on algebraic linguistics and machine translation. Technical report.

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

Boasson, L. and Sénizergues, S. (1985). NTS languages are deterministic and congruential. *J. Comput. Syst. Sci.*, 31(3):332–342.

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

Chi, Z. (1999). Statistical properties of probabilistic context-free grammars. *Computational Linguistics*, 25(1):131–160.

Clark, A. (2006). PAC-learning unambiguous NTS languages. In *Proceedings of the 8th International Colloquium on Grammatical Inference (ICGI)*, pages 59–71.

Clark, A. (2011). A language theoretic approach to syntactic structure. In Kanazawa, M., Kornai, A., Kracht, M., and Seki, H., editors, *The Mathematics of Language*, volume 6878 of *Lecture Notes in Computer Science*, pages 39–56. Springer Berlin Heidelberg.

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

Cohen, S. B., Stratos, K., Collins, M., Foster, D. P., and Ungar, L. (2012). Spectral learning of latent-variable PCFGs. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 223–231, Jeju Island, Korea. Association for Computational Linguistics.

Cohn, T., Blunsom, P., and Goldwater, S. (2010). Inducing tree-substitution grammars. *The Journal of Machine Learning Research*, 9999:3053–3096.

Collins, M. and Koo, T. (2005). Discriminative Reranking for Natural Language Parsing. *Computational Linguistics*, 31(1):25–70.

Drewes, F. and Högberg, J. (2003). Learning a regular tree language from a teacher. In *Developments in Language Theory*, pages 164–164. Springer.

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

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

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

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

Kasami, T. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Technical report, DTIC Document.

Klein, D. and Manning, C. (2005). Parsing and hypergraphs. *New developments in parsing technology*, pages 351–372.

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

Miller, P. (1999). *Strong generative capacity: The semantics of linguistic formalism.* CSLI Publications.

Paull, M. C. and Unger, S. H. (1968). Structural equivalence of context-free grammars. *Journal of Computer and System Sciences*, 2(4):427 – 463.

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

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

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

Sakakibara, Y. (1992). Efficient learning of context free grammars from positive structural samples. *Inf Computing*, 97(1):23–60.

Schapire, R. E. (1999). A brief introduction to boosting. In *Proceedings of 16th IJCAI.*

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

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

Yoshinaka, R. (2008). Identification in the limit of k, l-substitutable context-free languages. In *Proceedings of the 9th International colloquium on Grammatical Inference*, pages 266–279. Springer.

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

Yoshinaka, R. (2012). Integration of the dual approaches in the distributional learning of context-free grammars. In *LATA*, pages 538–550.

Yoshinaka, R. and Kanazawa, M. (2011). Distributional learning of abstract categorial grammars. In *Logical Aspects of Computational Linguistics*, pages 251–266. Springer.

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

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.

Lemma 8. *Suppose $X \rightarrow \alpha$ is a correct production, where X is a prime, and α is a string of at least two primes. Then $X \overset{*}{\Rightarrow}_{G_*} \alpha$.*

Proof. By induction on the length of α. Base case: α is of length 2, in which case it cannot be pleonastic. Suppose true for all α, $|\alpha| < k$. (α is a sequence of primes, so $|\alpha|$ is the number of primes in the sequence) Consider the correct production $X \rightarrow \alpha$ where α is of length k. If it is not pleonastic, then it is valid, and so $X \rightarrow \alpha$ is a production in the grammar, and so $X \Rightarrow \alpha$. Alternatively it is pleonastic and therefore $\alpha = \beta\gamma\delta$ where γ is the right hand side of a correct production, $Y \rightarrow \gamma$. Consider $X \rightarrow \beta Y \delta$, and $Y \rightarrow \gamma$. Both $\beta Y \delta$ and γ are shorter than α, and so by the induction hypothesis $X \overset{*}{\Rightarrow} \beta Y \delta$ and $Y \overset{*}{\Rightarrow} \gamma$ so $X \overset{*}{\Rightarrow} \alpha$. So the lemma follows by induction. □

Lemma 9. *Suppose $L \in \mathcal{L}_{sc}$, then suppose X is a prime, and $w \in X$. Then $X \overset{*}{\Rightarrow}_{G_*(L)} w$.*

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 8 $X \overset{*}{\Rightarrow} \alpha$. Since we have the lexical rules $[a_i] \to a_i$ we can also derive $\alpha \overset{*}{\Rightarrow} w$. $\square$

Lemma 10. *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 $x \in X$ and $x' \in X$ then $x = yv$ and $x' = y'v'$. Then $v \equiv v'$ by substitutability so $X = Y[v]$ so X is not prime. $\square$

Lemma 11. *In a substitutable language, 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. (we want to show it is in $\bar{\beta}$). 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 nonempty 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 components β say $b = b_1 \dots b_n$, where $b_i \in B_i$. Since we have some context l, r such that $lbr \in L$ so $lcbr \in L$ therefore 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 $\bar{\alpha} \subseteq \bar{\beta}$. $\square$

An immediate corollary is this:

Lemma 12. *In a substitutable language, if X is a prime, and α, β are strings of primes such that $X\bar{\alpha} = X\bar{\beta}$, where $X\bar{\beta} \subseteq L$, then $\bar{\alpha} = \bar{\beta}$.*

Lemma 13. *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 of B_1 which does not start with A_1. We know there must be such an element or B_1 would be composite. Let b_1 be such an element. 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 14. *If α and β are nonempty sequences of prime congruence classes in a language $L \in \mathcal{L}_{sc}$ 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: since $\bar{\alpha} \subseteq \bar{\beta}$, we know that there must be some i such that $A_1 \dots A_i \subseteq B_1$ and similarly some j such that $B_1 \dots B_j \subseteq A_1$. Consider the shortest string $w \in \bar{\alpha}$. This means that $w = a_1 \dots a_m = b_1 \dots b_n$. This means that all of the a_k, b_k are the shortest strings in their respective classes.

Suppose $a_1 \neq b_1$. Wlog assume that $|a_1| > |b_1|$. This implies that $a_1 = b_1 s$. So $A_1 \supseteq B_1 \dots B_i$ for some $i > 1$. So $s \equiv b_2 \dots b_i$. 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' which does not start with an element of B_1. Consider $w' = a' a_2 \dots a_m = b_1' \dots b_n'$ So a' must be a prefix of b_1' which means that $a' a_2 \dots a_j = b_1'$, 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 n $B_2 \dots B_n$. So $a_1 = b_1$ and $A_1 = B_1$. By Lemma 12 and by induction this means that $\alpha = \beta$. □

The fundamental lemma.

Lemma 15. *For every non zero non-unit congruence class, X, in a language in $\mathcal{L}_{sc}$ there is a unique sequence of primes α such that $X = \bar{\alpha}$. In other words every such congruence class can be represented uniquely as a product of primes.*

Proof. We show that every congruence class can be represented as a product of primes; uniqueness then follows immediately by Lemma 14. 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: it is true for all congruence classes whose shortest string is of length $< k$. 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, $Y = Y_1 \ldots Y_i$ and $Z = Z_1 \ldots Z_j$ so $X = Y_1 \ldots Y_i Z_1 \ldots Z_j$. □

Lemma 16. *Let N be a prime and α, β, γ sequences of primes such that $N \to \alpha\beta$ and $N \to \alpha\gamma$ are both valid productions. Then $\beta = \gamma$.*

Proof. $[\bar{\beta}] = [\bar{\gamma}]$ by substitutability. Let δ be the prime decomposition of $[\bar{\beta}]$.

So $\bar{\delta} \supseteq \bar{\beta}$. Let $\beta = B_1 \ldots B_m$ and $\delta = D_1 \ldots D_n$, $\gamma = G_1 \ldots G_l$. By Lemma 13 we know that there is some j such that $B_1 \ldots B_j \subseteq D_1$. If $j > 1$ then $N \to \alpha\beta$ would be pleonastic and thus not valid, so $j = 1$. So $B_1 = D_1$ and similarly $G_1 = D_1$ so $B_1 = G_1$. Now if $m = 1$ then we must have $l = 1$ since if $l > 1$ then $G_1 \ldots G_l \subseteq B_1$ would mean that $N \to \alpha\gamma$ was not valid because pleonastic. Therefore, $m, l > 1$. We repeat the argument and therefore by induction $\beta = \gamma$. □

Lemma 17. *If L has a finite number of primes, then it only has a finite number of valid productions.*

Proof. 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 $\alpha = \beta$, by Lemma 16 which means that there can be at most one production for each pair of primes N, A; therefore the total number of productions is at most n^2, if n is the number of primes in the language. □