Distributional Learning of Some Context-Free Languages with a Minimally Adequate Teacher

Alexander Clark

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

Abstract. Angluin showed that the class of regular languages could be learned from a Minimally Adequate Teacher (MAT) providing membership and equivalence queries. Clark and Eyraud (2007) showed that some context free grammars can be identified in the limit from positive data alone by identifying the congruence classes of the language. In this paper we consider learnability of context free languages using a MAT. We show that there is a natural class of context free languages, that includes the class of regular languages, that can be polynomially learned from a MAT, using an algorithm that is an extension of Angluin's LSTAR algorithm.

1 Introduction

The inference of context free languages is in a less developed state than the study of the inference of regular languages. Angluin [2] showed that a simple criterion, reversibility, could be used to identify a class of regular languages from positive data alone using Deterministic Finite Automata (DFAs). This approach has two parts; first basing the states of the learned automata on the residual languages or right congruence classes of the language, and secondly a simple test for telling whether two strings are in the same class. Later, she showed [3] that a much larger class, indeed the class of all regular languages, could be learned using a richer model, the Minimally Adequate Teacher (MAT) model. In this model the learner is provided with two sources of information about the language. First, the learner can ask membership queries — for a given string the learner can find out whether that string is in the language — and secondly the learner can ask equivalence queries — the learner provides the teacher with a hypothesis, and the teacher will either confirm that it is correct, or it will provide the learner with a counter-example. The algorithm LSTAR is a classic and well-studied algorithm; it uses the same representational idea — the states again correspond to the residual languages, but the test for equivalence of strings is much more sophisticated, using a series of test suffixes to define the equivalence classes.

When it comes to context free inference, Clark and Eyraud [1] showed a result which is an exact analogue of the first paper of Angluin. They show that a learnability result could be established from positive data alone, by combining a representational decision – the non-terminals correspond to the syntactic congruence class – with a simple test – weak substitutability. In this paper we try

J.M. Sempere and P. García (Eds.): ICGI 2010, LNAI 6339, pp. 24–37, 2010.

to extend this approach to produce an algorithm for context free grammatical inference that is similar to the LSTAR algorithm. In the process, we will borrow some ideas and terminology from [4]. We will make equivalence queries where the hypothesis may not be in the learnable class – this is sometimes called an Extended Minimally Adequate Teacher [5].

The minimal DFA has a special status – there is a bijection between the states of the automaton and the residual languages. Though it is "minimal" it may be exponentially larger than the smallest non-deterministic automaton for the same language. We can consider the equivalent construction for CFGs to be where we have a correspondence between the congruence classes of the language and the non-terminals of the grammar – we define these precisely in Section 2. An important difference is that the number of congruence classes of a language will be infinite if it is not regular. We will therefore only model some finite subset of the congruence classes; as a result the class of languages that we can learn is limited, and as we shall see in Section 3, does not correspond to the class of all context free languages. In this algorithm we will use an observation table that is similar to that used in the LSTAR algorithm, but one that consists of substrings and contexts, rather that prefixes and suffixes (Section 4). In the remaining sections of the paper, we will define the algorithm and prove its correctness and polynomial efficiency in the standard way.

2 Notation

We have a finite non-empty alphabet Σ which is known; we use Σ^* to be the set of all finite strings of Σ, and λ is the empty string. A language is a subset of Σ^*. A context (l, r) is just a pair of strings; an element of $\Sigma^* \times \Sigma^*$. The distribution of a string $C_L(w) = \{(l, r)|lwr \in L\}$. We define $(l, r) \odot u$ to be lur – the wrapping operation which combines a context with a substring, and we extend this to sets of contexts and sets of strings in the natural way. We will write $Sub(w)$ for the set of all substrings of a string, so $Sub(w) = \{u|\exists l, r \in \Sigma^*, lur = w\}$.

Two strings are congruent with respect to a language L, written $u \equiv_L v$ iff $C_L(u) = C_L(v)$. This is an equivalence relation and we write $[u]_L = \{v|u \equiv_L v\}$, for the equivalence class of u.

This elementary lemma [1] establishes the basis for this learning approach:

Lemma 1. *For any language L, if $u \equiv_L u'$ and $v \equiv_L v'$ then $uv \equiv_L u'v'$.*

This means that we can define a concatenation operation between these classes: $[u] \circ [v] = [uv]$. This is a monoid, as it is associative and contains a unit $[\lambda]$; it is called the syntactic monoid: $\Sigma^*/\equiv_L$.

3 Objective CFGs

Our representations are context free grammars (CFG). We define a very slightly non-standard definition of a CFG. A CFG is a tuple $\langle \Sigma, V, P, I \rangle$ where Σ is a set

of terminal symbols, V is a finite non-empty set, (non-terminals), I is a non-empty subset of V, the set of initial symbols, and P is a finite set of productions of the form $V \times (V \cup \Sigma)^*$, which we write $N \to \alpha$, where $N \in V$, and α is a possibly empty string of terminal and non-terminal symbols. We define a standard derivation relation $\gamma N \delta \to \gamma \alpha \delta$, when $N \to \alpha \in P$, and let $\stackrel{*}{\Rightarrow}_G$ be the transitive reflexive closure of this relation. We define $L(G, A) = \{w | A \stackrel{*}{\Rightarrow}_G w\}$ and $L(G) = \bigcup_{S \in I} L(G, S)$. Note that we allow multiple start symbols. Clearly this does not change the class of languages that can be defined, since we could add one more symbol S' and add productions $S' \to S$ for all $S \in I$. Secondly, we will allow the alphabet Σ to be empty. We will consider the case of CFGs in Chomsky Normal Form (CNF): all productions are of the form $N \to PQ$, or $N \to a$ or $N \to \lambda$. While in general we can assume w.l.o.g. that grammars are in CNF, this assumption might in this case limit the class of languages that can be represented.

3.1 Congruential Grammars

We are interested in grammars where there is a relation between the non-terminals and the congruence classes. We say that a CFG G is *congruential* if for every non-terminal N it is the case that $L(G, N)$ is a subset of a congruence class of $L(G)$; i.e. if $N \stackrel{*}{\Rightarrow} u$ and $N \stackrel{*}{\Rightarrow} v$ implies that $u \equiv_{L(G)} v$. This means that $L(G, N)$ will be a subset of a congruence class but each non-terminal need not generate every string in the congruence class. Note that this differs from the definition of congruential given in [6].

There are a few interesting properties of these grammars: first, note that we can assume w.l.o.g. that we only have one non-terminal for each congruence class. If we have two non-terminals that both generate strings from the same congruence class $[w]$ then we can clearly merge them, without changing the language defined by the grammar. That is to say, if $N \stackrel{*}{\Rightarrow} u$ and $u \equiv_L v$, then if we add productions so that $N \stackrel{*}{\Rightarrow} v$ then this will leave the language unchanged. Secondly, the binary productions will all be of the form $[u] \circ [v] \to [u][v]$.

We consider a simple example: the context-free language $L = \{a^n b^n | n \geq 0\}$. This has an infinite number of congruence classes including the following five: $[\lambda] = \{\lambda\}$, $[a] = \{a\}$, $[b] = \{b\}$, $[ab] = \{ab, aabb, \dots\} = L \setminus \{\lambda\}$ and $[aab] = \{aab, aaabb, \dots\}$ Let us define a CFG whose non-terminals correspond to these 5 congruence classes; we will label them as Z, A, B, S, T. We have productions $Z \to \lambda$, $S \to AB$, $S \to TB$, $T \to AS$, $A \to a$ and $B \to b$. It is easy to verify that this is congruential in that $L(G, A) = [a]$, $L(G, B) = [b]$, $L(G, S) = [ab]$ and so on. The set of initial symbols is $I = \{S, Z\}$; we see that $L(I) = L$.

Definition 1. *Let $\mathcal{L}_{\text{CCFG}}$ to be the set of all languages definable by a congruential* CFG.

Space does not permit a full exploration of the relationship of this class to other learnable classes, but we can make a few basic points. We can assume that the grammar is in Chomsky Normal Form, as we discuss below.

First note that $\mathcal{L}_{\text{CCFG}}$ includes all regular languages. Regular languages have only a finite number of congruence classes [7]. We can therefore construct a grammar which has one non-terminal for every congruence class of a regular language L, together with the set of productions $[a] \to a$, and $[uv] \to [u][v]$, $[\lambda] \to \lambda$; it is trivial to show that this will have the property that $[u] \stackrel{*}{\Rightarrow} u$ for all u, and thus if we set $I = \{[u] | u \in L\}$, we will have a grammar in $\mathcal{L}_{\text{CCFG}}$ that defines the language L.

Secondly, $\mathcal{L}_{\text{CCFG}}$ includes all NTS languages [6]. An NTS grammar is a grammar such that for any non-terminals N, M and strings l, w, r if $N \stackrel{*}{\Rightarrow} w$ and $M \stackrel{*}{\Rightarrow} lwr$ then $M \stackrel{*}{\Rightarrow} lNr$. NTS grammars are clearly congruential. Suppose G is an NTS grammar defining a language L. Suppose $N \stackrel{*}{\Rightarrow} u$ and $lur \in L$, and $N \stackrel{*}{\Rightarrow} v$. then there is an $S \in I$ such that $S \stackrel{*}{\Rightarrow} lur$; and therefore by the NTS property, $S \stackrel{*}{\Rightarrow} lNR \stackrel{*}{\Rightarrow} lvr$; and so $u \equiv_L v$. Moreover, if $\alpha \in (V \cup \Sigma)^+$ then if $\alpha \stackrel{*}{\Rightarrow} u$ and $\alpha \stackrel{*}{\Rightarrow} v$ then $u \equiv_L v$, by induction on the length of α using Lemma 1. Therefore we can binarise the right-hand sides of the rules of G; the resulting grammar may not be NTS but will still be congruential.

Not all CFLs are in $\mathcal{L}_{\text{CCFG}}$. In particular, languages which are a union of infinitely many congruence classes are not. For example, the languages $\{a^n b^m | n > m > 0\}$, and $\{a^n b^n\} \cup \{a^n b^{2n}\}$ are not in $\mathcal{L}_{\text{CCFG}}$ for exactly this reason. Similarly, this class does not include the palindrome language over a, b or the even palindrome language over a, b and thus does not include the class of even linear languages [8]. However $\mathcal{L}_{\text{CCFG}}$ does include the substitutable CFLs [1], and the k-l-substitutable languages [9]. As we shall see it includes the Dyck language, which is neither substitutable, linear nor regular.

We conjecture that the classes of NTS languages, pre-NTS languages and congruential languages all coincide, though the corresponding classes of grammars are clearly distinct, but the exact relationships are still not fully established.

4 Observation Table

We now define the basic data structure that we will use which is a modification of the observation table used by Gold[10] and Angluin [3].

It is an observation table which consists of a non-empty finite set of strings K a non-empty finite set of contexts F and a finite function mapping $F \odot KK$ to $\{0, 1\}$. Since K always contains λ, K is a subset of KK. Given a context (l, r) in F and a substring $w \in KK$, we have a 1 in the table if lwr is in the language and a 0 if it is not. We will write this as a tuple $\langle K, D, F \rangle$, where D is the set of grammatical strings in $F \odot KK$. Figure 1 illustrates a simple example.

For two strings $u, v \in KK$ we say they are equivalent if they appear in the same set of contexts; and we write this $u \sim_F v$; in Angluin's terms this is $row[u] = row[v]$. This means that $C_L(u) \cap F = C_L(v) \cap F$. Note that if $u \equiv_L v$ then $u \sim_F v$ for any set of contexts; conversely, if it is not the case that $u \equiv_L v$ then we can find some context (l, r) such that if $(l, r) \in F$, it is not the case that $u \sim_F v$. Of course, our set of contexts may be too small, in which case we may

	(λ,λ)	(a,λ)	(λ,b)	(λ,ab)
λ	1	0	0	1
a	0	0	1	0
b	0	1	0	0
ab	1	0	0	0

	(λ,λ)	(a,λ)	(λ,b)	(λ,ab)
aab	0	0	1	0
abb	0	1	0	0
aa	0	0	0	0
ba	0	0	0	0
bb	0	0	0	0
bab	0	0	0	0
aba	0	0	0	0
$abab$	0	0	0	0

Fig. 1. Observation table for $L = \{a^n b^n | n \geq 0\}$. We have $F = \{(\lambda,\lambda),(a,\lambda),(\lambda,b),(\lambda,ab)\}$, which has 4 elements; these head the 4 columns in the diagram. Each row corresponds to an element of KK. K consists of the 4 strings λ, a, b, ab. We have split the table into two parts; on the left we have K, and on the right we have $KK \setminus K$.

have $u \sim_F v$ even though u and v are not congruent. In general we will want to increase F so that this does not happen.

4.1 Construction of Grammar

Given an observation table we can construct a CFG from it in a fairly straightforward way. First we assume that we have all the information we need: no holes in the table. In the learning model we use, we will have a membership oracle **Mem**, and we can use this to fill in the cell of the table corresponding to context (l,r) and substring u, by querying $\mathbf{Mem}(lur)$.

Definition 2. *An observation table, K, F, D is consistent if for all u_1, u_2, v_1, v_2 in K, if $u_1 \sim_F u_2$ and $v_1 \sim_F v_2$, then $u_1 u_2 \sim_F v_1 v_2$.*

If a table is not consistent, then we know that we do not have a large enough set of features, by Lemma 1 and we could add additional features until it is consistent.

Algorithm 1 selects appropriate contexts to make sure that the table is consistent. We want to limit the number of contexts we add; note that every time we add a context we will increase the number of congruence classes of K. Thus it is clear that Algorithm 1 will terminate after at most $|K|$ iterations. We use a similar approach in Algorithm 3. However, though the number of iterations that Algorithm 1 will make is bounded, we do not yet see how to bound the total run-time of the algorithm as the length of the contexts being generated might become very large. It is also not necessary to have a consistent table; if the table is inconsistent, then we may generate a grammar which will have two rules of the form $N_1 \to AB$ and $N_2 \to AB$, for distinct non-terminals N_1 and N_2.

Algorithm 2 presents the algorithm for constructing the grammar from an observation table. This runs in polynomial time, and always returns a valid CFG. We will refer to the output of this algorithm as $G(K, D, F)$.

Data: K, D, F;
Result: A set of features that is consistent
while $\langle K, D, F\rangle$ *is not consistent* **do**
  Find u_1, u_2, and v_1, v_2 in K, and $(l, r) \in F$ such that $u_1 \sim_F u_2$, $v_1 \sim_F v_2$, $lu_1v_1r \in D$, and lu_2v_2r is not in D ;
  if $\texttt{Mem}(lu_1v_2r) = 1$ **then**
    $F \leftarrow F \cup \{(l, v_2r)\}$;
  else
    $F \leftarrow F \cup \{(lu_1, r)\}$;
  Use `Mem()` to increase D to fill in the observation table ;
return F;

Algorithm 1. MakeConsistent

Data: K, D , F ;
Result: A Context Free Grammar G
Divide K into equivalence classes according to $\sim_F$;
Let V be the set of these equivalence classes ;
Let $I = \{N \in V | \forall w \in N, w \in D\}$;
$P \leftarrow \{N \rightarrow a | a \in N \cap \Sigma\}$;
$P \leftarrow P \cup \{N \rightarrow PQ | u \in P, v \in Q, w \in N, uv \sim_F w\}$;
$P \leftarrow P \cup \{N \rightarrow \lambda | \lambda \in N\}$;
return $G = \langle \Sigma, V, P, I\rangle$;

Algorithm 2. MakeGrammar

5 Adding Features

There are two sorts of errors to deal with; undergeneration and overgeneration. The more difficult one is to cope with overgeneralisation, so before we define the full algorithm, we will discuss how this is dealt with. First, if the partition of KK into classes is correct, then we will not overgeneralise; that is to say, if for any u, v in KK, we have that $u \sim_F v$ implies $u \equiv_L v$, then we will not overgeneralise. If we overgeneralise, then there must be two strings w_1, w_2 in KK that appear to be congruent but are not. So we need to add a feature/context to have a more fine division into classes so that the two strings w_1 and w_2 are in different classes.

The categories are equivalence classes of KK under $\sim_F$, that contain at least one element of K. We will consider the categories to be subsets of KK, and write $w \in X$, where X is one of these categories. Note that some elements of KK will not be in a category, if they do not contain any element of K. These will correspond to congruence classes that we do not model, but are aware of.

In Algorithm 3 we are given a derivation that is incorrect: we have a derivation of a string w from a non-terminal X such that we know that w is not congruent to the strings in X. Formally we have a context (l, r) such that $X \stackrel{*}{\Rightarrow} w$ and

$lwr \in L(G) \setminus L$, and yet there is a $w' \in X$ such that $lw'r \in L$. We return a context that *splits* some category X in the grammar. We say that a category X is split by a context (l, r) if there are $u, v \in X$ such that $lux \in L$ and $lvx \notin L$.

We will explain the algorithm informally: suppose we have a non-terminal X, that generates a string w. X corresponds to a subset of strings in KK, say $\{x_1, \ldots x_k\}$. Ideally we will have that w is congruent to all of these x_i and that all of these x_i are congruent to each other. Suppose we observe that this is not the case and that w has some context that an x_i does not. If some of the x_i have it and some do not, then we can use this to split the category X. Otherwise it might be that all of the x_i are in fact congruent to each other, and that the problem is with some other productions. Consider a derivation of w from X.

Suppose the derivation starts with the production $X \to YZ$, such that $Y \stackrel{*}{\Rightarrow} u$ and $Z \stackrel{*}{\Rightarrow} v$ and $w = uv$. We will have two strings $u', v' \in K$ and $u'v' \in KK$ such that u' is in the category Y, $v' \in Z$ and $u'v' \in X$, and $lu'v'r \in L$. Now crucially, we know that $w = uv$ is not congruent to $u'v'$; since the context (l, r) distinguishes them. Therefore either u is not congruent to u' or v is not congruent to v', or possibly both are not congruent. If they were both congruent, this would violate Lemma 1. Note that we might have that $u = u'$ or $v = v'$ in which case we know immediatxely which of the pair is different. Suppose not; then we consider the intermediate string $u'v$. If $lu'vr \in L$, then we know that u' has the context (l, vr) but u does not. If $lu'vr \notin L$ then we know that v' has the context (lu', r) and v does not. We then recurse.[1]

When we reach a leaf, we know that the algorithm must terminate. Since K contains the strings of length 1, if our derivation is of the form $X \to a$, then we know that the feature (l, r) will split it, since $a \in X$ by construction.

Lemma 2. *Algorithm 4 works in polynomial time in $|w|$ and $|K|$; and terminates with a set of features such that the derived grammar no longer generates w.*

Proof. We can prove that `FindContext` will always find a context that splits a class of KK. Since we can have at most $|K|^2$ equivalence classes, it will terminate after adding at most $|K|^2$ new contexts.

More generally we can see that, as with Binary Feature Grammars [4], if we increase F then we decrease the language defined by the grammar.

Lemma 3. *Suppose $F_1 \subseteq F_2$ and $G_i = G(K, D, F_i)$. If $u \in K$, we write $N_i[u]$ for the category in G_i that contains u. For all $u \in K$, if $N_2[u] \stackrel{*}{\Rightarrow}_{G_2} v$ then $N_1[u] \stackrel{*}{\Rightarrow}_{G_1} v$*

Proof. Note that since F_2 is bigger than F_1 and they are based on the same data, $u \sim_{F_2} v$ implies $u \sim_{F_1} v$. We prove the result by induction on length of derivation. It is clearly true for strings of length 1, since if $u \sim_{F_2} a$, $u \sim_{F_1} a$. Suppose true for all strings up to length k, and let v be a string of length $k+1$, $N_2[u] \stackrel{*}{\Rightarrow}_{G_2} v$. Expanding the first step of this derivation, there must be

[1] A slightly more complex and efficient approach would be to consider also uv' and possibly add two contexts if necessary.

Data: A finite set of strings K, a finite set of contexts F, a finite set of strings D, and a triple X,(l,r),w, where w is a string, (l,r) is a context, X is a non-terminal such that $X \stackrel{*}{\Rightarrow} w$;
and $lwr \in L(G) \setminus L$;
Result: A context that splits some category of G

if (l,r) *splits* X **then**
  return (l,r);
else
  Let $X \to YZ \stackrel{*}{\Rightarrow} uv = w$ be a derivation of w such that $Y \stackrel{*}{\Rightarrow} u, Z \stackrel{*}{\Rightarrow} v$;
  Find a pair of strings $u', v' \in K$ such that $u' \in Y, v' \in Z, u'v' \in X$;
  if `Mem`$(lu'vr) = 1$ **then**
    return `FindContext`$(Y, (l, vr), u)$;
  else
    return `FindContext`$(Z, (lu', r), v)$;

Algorithm 3. FindContext

Data: K, D, F and a string w that is not in L
Result: A set of features including F, such that the grammar does not generate w

$G \leftarrow$ `MakeGrammar`(K,D,F);
while G *generates* w **do**
  Suppose $S \stackrel{*}{\Rightarrow} w$ for some $S \in I$;
  f be `FindContext`$(S,(\lambda,\lambda),\ w)$;
  $F \leftarrow F \cup \{f\}$;
  Increase $D = L \cap (F \odot KK)$ using `Mem()` ;
  $G =$ `MakeGrammar`(K,D,F) ;
return F ;

Algorithm 4. AddContexts

$N_2[u] \to N_2[x]N_2[y]$, where $x, y \in K$, where $N_2[x] \stackrel{*}{\Rightarrow}_{G_2} p$ and $N_2[y] \stackrel{*}{\Rightarrow}_{G_2} q$ and $v = pq$. By the inductive hypothesis, we have that $N_1[x] \stackrel{*}{\Rightarrow}_{G_1} p$ and $N_1[y] \stackrel{*}{\Rightarrow}_{G_1} q$. We know that by construction of G_2 and its consistency, we have $u \sim_{F_2} xy$; therefore $u \sim_{F_1} xy$, therefore there is a production $N_1[u] \to N_1[x]N_1[y]$, and therefore $N_1[u] \stackrel{*}{\Rightarrow}_{G_1} v$ and the result follows.

6 Algorithm

We now informally describe our algorithm: we maintain a set of strings K, a set of contexts F and some data D. We initialise $K = \{\lambda\}$ and $F = \{(\lambda, \lambda)\}$. We fill in D using the membership oracle. We make it consistent, generate a grammar and then query. Note that we only want to add strings to the grammar as a result of counter-examples. Since the class of grammars includes ones which

define languages where every string is exponentially large in the number of non-terminals, we need to wait until we are given a long string; otherwise we will violate the polynomial bound.

If the grammar is correct, then we terminate; otherwise if we overgenerate, we add more features until we no longer generate that string. If, on the other hand we undergenerate, then we add more strings to K to increase the number of congruence classes of the languge that we generate.

Suppose the target $G = \langle V, P, I \rangle$ is a congruential CFG in CNF; and let V' be a subset of V. we say that K is sufficient for V' if it contains one string from each non-terminal yield; i.e. for all $N \in V'$ there is a string $u \in K$ such that $u \in L(G, A)$.

Lemma 4. *If K is sufficient for V', and F is consistent, then let $\hat{G} = G(\langle K, D, F \rangle)$ denote $w(N)$ for a string in K for some $N \in V'$. Then for every derivation of a string in G, that contains only non-terminals of V' say $N \stackrel{*}{\Rightarrow}_G w$, we have a derivation in $\hat{G}$ of $N(w(N)) \stackrel{*}{\Rightarrow}_{\hat{G}} w$;*

Proof. If N is a non-terminal in G let $w(N)$ be one of the corresponding strings in K, by sufficiency. Suppose we have a rule $N \to PQ$ in G; then $w(N) \equiv_L w(P)w(Q)$; which means that $w(N) \sim_F w(P)w(Q)$; which means there is a production $N(w(N)) \to N(w(P))N(w(Q))$ in the set of productions of $\hat{G}$. Similarly for the productions $N \to a$ and the production $N \to \lambda$, and thus every derivation of a string w with respect to G, can be converted into a derivation of a string w in $\hat{G}$.

A consequence of this is that if we undergenerate, this can only be because we do not have a large enough set of K; crucially, if we observe a positive counter-example, the derivation of that string must use at least one non-terminal that we do not have an example of in K. Therefore if we add all substrings of this counter-example to K, we will increase the number of non-terminals we have covered by at least 1. Therefore, if there is a grammar with n non-terminals for the target language, we will only need at most n positive counter-examples before we have a grammar that includes all of the target language.

6.1 Convergence Proof

We now prove that this algorithm will learn the class.

Lemma 5. *Given a sub-congruential CFG in CNF, G, with non-terminals V; For K, let $n(K)$ be $|\{N \in V | \exists u \in K, N \stackrel{*}{\Rightarrow} u\}|$. Suppose $\hat{(G)} = G(K, L, F)$, If w is a positive countexample, (i.e $w \in L(G) \setminus L(\hat{G})$) then $n(K \cup Sub(w)) > n(K)$.*

Proof. Let w be such an example; there must be a non-terminal that we have not yet observed used in the derivation; call this N. Therefore there is a derivation $S \stackrel{*}{\Rightarrow} lNr \stackrel{*}{\Rightarrow} lur = w$; $u \in Sub(w)$, therefore $n(K \cup Sub(w)) > n(K)$.

Theorem 1. *There is a polynomial p, such that if $L \in \mathcal{L}_{\text{CCFG}}$ and is generated by a sub-congruential grammar in CNF with n non-terminals, then Algorithm 5*

Result: A CFG G
$K \leftarrow \Sigma \cup \{\lambda\}$, $K_2 = K$;
$F \leftarrow \{(\lambda, \lambda)\}$;
$D = L \cap \{\lambda\}$;
$G = \langle K, D, F \rangle$;
while *true* **do**
  if `Equiv(`G`)` *returns correct* **then**
    return G ;
  $w \leftarrow$ `Equiv(`G`)` ;
  if w *is not in* $L(G)$ **then**
    $K \leftarrow K \cup Sub(w)$;
  else
    $F \leftarrow$ `AddContexts(`G`,`w`)`;
  $G \leftarrow$ `MakeGrammar(`K, D, F`)` ;

Algorithm 5. LearnCFG

will terminate, returning a correct grammar, after $p(n, l)$ steps, where l is the maximum length of examples returned by the equivalence oracle.

Proof. First, if it terminates at all, then the result is correct. Note that we will only add positive counterexamples at most n times. Each positive example will add at most $\frac{1}{2}l(l+1)$ elements to K; and at the start K is of size 1. Therefore $|K|$ is always at most $1 + \frac{n}{2}l(l+1)$. Every time we add a feature we will split a class of KK, and the total number of equivalence classes of KK cannot be more than $|K|^2$. Each time we get a negative example, we will add at least one more feature, and so the total number of negative examples cannot be greater than $|K|^2$. Therefore Algorithm 5 will terminate after at most $n + |K|^2$ iterations; by previous lemmas the overall complexity is polynomial.

This proof is a short-cut proof – while valid it perhaps does not explain the result fully. When we add contexts, we reduce the language, until eventually we will only undergenerate. In particular, eventually we will have that $u \sim_F v$ implies $u \equiv_L v$; i.e. that we have split the classes up as finely as possible until they correspond to the actual congruence classes of the language. We can explain this with an additional lemma, which we won't prove:

Lemma 6. *For any K and any L there is a finite set of contexts F_0 such that for all $F \supseteq F_0$, $L(G(K, L, F)) \subseteq L$.*

Note additionally that the algorithm is polynomial at every step – it is possible to create an algorithm that "cheats" by using an exponential amount of data and then constructing a hypothesis with only an exponentially long counter-example [11], in order to force l to be exponentially large, so that the overall complexity will be polynomial. This algorithm satisfies the stricter condition that at each step the amount of computation used is polynomially bounded in n and l.

7 Sample Run

We will now illustrate this with a sample run of the algorithm on the Dyck language over the alphabet $\{a, b\}$ where $L = \{\lambda, ab, abab, aabb, \dots\}$. This is an infinite non-linear context free language.

We initialise K and F to the trivial starting points. Our observation table is shown as Step 0 in Figure 2. This is vacuously consistent, so we create a grammar with one non-terminal, S and the rules $S \to \lambda$, and $S \to SS$, which just generates the language $\{\lambda\}$. We query this, and it undergenerates; let us suppose we receive the positive counterexample ab. We add $Sub(ab)$ to K, getting $K = \{\lambda, a, b, ab\}$; this is shown as Step 1.

Step 0	(λ,λ)
λ	1

Step 1	(λ,λ)
λ	1
a	0
b	0
ab	1
aa	0
ba	0
bb	0
abb	0
aba	0
aab	0
bab	0
abab	1

Step 2	(λ,λ)	(a,λ)
λ	1	0
a	0	0
b	0	1
ab	1	0
aa	0	0
ba	0	0
bb	0	0
abb	0	1
aba	0	0
aab	0	0
bab	0	1
abab	1	0

Step 3	(λ,λ)	(a,λ)	(λ,b)
λ	1	0	0
a	0	0	1
b	0	1	0
ab	1	0	0
aa	0	0	0
ba	0	0	0
bb	0	0	0
abb	0	1	0
aba	0	0	1
aab	0	0	1
bab	0	1	0
abab	1	0	0

Fig. 2. States of observation table; in each case, the elements of K are above the line, and $KK \setminus K$ below the line

So this is not consistent since $a \sim_F b$ but aa is not equivalent to ab. We add the feature (a, λ), which will separate a and b. This gives us Step 2 in the figure; this is consistent, since the only two strings in K that are similar are λ and ab and these are in fact congruent. So we define the grammar which has 3 non-terminals S, A, B.

This has the three lexical rules $S \to \lambda$ $A \to a$ and $B \to b$. We also have these binary rules: $S \to SS$, $S \to AB$ $A \to AA$ $A \to BA$ and $A \to BB$.

The set of rules expanding A are clearly bizarre; but we get them since for example if we combine an a with an a we get aa which has the same features as A; the problem is that $a \sim_F aa$ but clearly they are not congruent. This defines a language which overgenerates since we have $S \to AB \to AAB \to aab$. So we then equivalence query this grammar, and get, let us suppose, the string aab. We now call **FindContext** with arguments $S, (\lambda, \lambda), aab$; we have a derivation $S \to AB$ from the strings $ab \to a, b$. We test ab which is in the language, so we then call **FindContext** with arguments $A, (\lambda, b), aa$. A has the set of strings a, aa and we note that (λ, b) splits these, so we return (λ, b) and add it to F.

This is again consistent, shown as Step 3, and gives us the grammar with lexical rules as before and binary rules:

- $S \to SS$, $S \to AB$
- $A \to AS, A \to SA$,
- $B \to BS, B \to SB$

This is consistent and defines the right language so we query and terminate.

8 Discussion

The MAT model, though standard in grammatical inference, is unrealistic for classes of representations where the equivalence queries are undecidable.

We decided to use this model for several reasons: the first is the ZULU competition [12] which looks again at practical issues in the use of the LSTAR algorithm.

In the context of the specific representation classes we use here, we note that the equivalence of NTS grammars is decidable [13], and that it is polynomially decidable whether a given CFG is NTS [14]. Therefore synthetic experiments with this algorithm are certainly practical. We have implemented the algorithm using a sampling approximation to the equivalence oracle and, though we do not present any experimental results here, it is simple and efficient. More generally, one can approximate the equivalence oracle by generating examples from a fixed distribution or from both target and hypothesis grammars, either using a probabilistic CFG or otherwise[15].

Finally, the MAT model divides representation classes along interesting lines: DFAs are learnable and NFAs are not. Gold style identification in the limit models tend to be either too restrictive or too permissive. MAT learning, on the other hand, seems to have the right level of difficulty: it accords well with practical experiences in learning competitions such as the Tenjinno [16] and Omphalos competitions [17].

de la Higuera [18] says

> the question as to whether it (the class of context free grammars) can be identified with a polynomial number of queries to a MAT is still an open question, but widely believed to be also intractable.

The work presented here provides a partial answer to this question: it seems clear that using these approaches only a limited subclass can be learned efficiently; however this work does seem to have overcome the barrier of "linearity" that has been observed. There is still a form of determinism in the grammars, but this is a bottom up determinism, rather than a left-to-right determinism.

Shirakawa and Yokomori [5] propose the following definition: A grammar G is context-deterministic iff whenever the derivation $S \stackrel{*}{\Rightarrow} lAr$ exists $L(G, A) = \{w | lwr \in L\}$. This is a very interesting approach that is closely related to our own; this requirement states that for every non-terminal any of the contexts will uniquely pick out the language. Thus we have the same idea, albeit in a much

weaker form: we consider grammars where the sets of strings generated by non-terminals are distributionally defined. Note that the Dyck language for example, is not context-deterministic, since ab occurs in the context (a, b) but so does ba.

The approach here is less powerful than the lattice based approaches in [4,19], but they are easier to understand since they are based on the classic CFG representation. Nonetheless this paper gives a very natural "textbook" algorithm for a reasonable class of context free languages. The class of languages definable is the largest class we can hope to represent using the natural one non-terminal per congruence class representation. This is also a properly polynomial result for a class of representations that includes some languages where the smallest strings are exponentially large. [1] shows only a characteristic set of polynomial cardinality, which is strictly speaking too weak, since one could specify an additional very long string in the set in order to give the algorithm arbitrary amounts of additional computational time, and [4] only gives a polynomial update time algorithm. [9] suggests using the thickness as an additional parameter.

Comparing this result to the Binary Feature Grammar (BFG) result in [4], one important difference is that the BFG approach uses as its representational primitives sets of the form $\{w|C_L(w) \supseteq C_L(v)\}$ rather than the congruence classes here. This increases the class of languages, but means that it is more difficult to get a polynomial MAT result.

Acknowledgments

I am very grateful to two anonymous reviewers for their helpful suggestions, and to Ryo Yoshinaka for some technical suggestions and corrections.

References

1. Clark, A., Eyraud, R.: Polynomial identification in the limit of substitutable context-free languages. Journal of Machine Learning Research 8, 1725–1745 (2007)
2. Angluin, D.: Inference of reversible languages. Communications of the ACM 29, 741–765 (1982)
3. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(2), 87–106 (1987)
4. Clark, A., Eyraud, R., Habrard, A.: A polynomial algorithm for the inference of context free languages. In: Clark, A., Coste, F., Miclet, L. (eds.) ICGI 2008. LNCS (LNAI), vol. 5278, pp. 29–42. Springer, Heidelberg (2008)
5. Shirakawa, H., Yokomori, T.: Polynomial-time MAT Learning of C-Deterministic Context-free Grammars. Transactions of the information processing society of Japan 34, 380–390 (1993)
6. Boasson, L., Sénizergues, S.: NTS languages are deterministic and congruential. J. Comput. Syst. Sci. 31(3), 332–342 (1985)
7. Ginsburg, S.: The Mathematical Theory of Context-Free Languages. McGraw-Hill, Inc., New York (1966)
8. Takada, Y.: Grammatical inference for even linear languages based on control sets. Information Processing Letters 28(4), 193–199 (1988)

9. Yoshinaka, R.: Identification in the Limit of k-l-Substitutable Context-Free Languages. In: Clark, A., Coste, F., Miclet, L. (eds.) ICGI 2008. LNCS (LNAI), vol. 5278, pp. 266–279. Springer, Heidelberg (2008)
10. Gold, E.M.: Complexity of automaton identification from given data. Information and Control 37(3), 302–320 (1978)
11. Angluin, D.: Negative results for equivalence queries. Machine Learning 5(2), 121–150 (1990)
12. Combe, D., de la Higuera, C., Janodet, J.C.: Zulu: an interactive learning competition. In: Yli-Jyrä, A. (ed.) FSMNLP 2009. LNCS(LNAI), vol. 6062, pp. 139–146. Springer, Heidelberg (2010)
13. Sénizergues, G.: The equivalence and inclusion problems for NTS languages. J. Comput. Syst. Sci. 31(3), 303–331 (1985)
14. Engelfriet, J.: Deciding the NTS property of context-free grammars. Results and Trends in Theoretical Computer Science, 124–130 (1994)
15. Gore, V., Jerrum, M., Kannan, S., Sweedyk, Z., Mahaney, S.: A Quasi-polynomial-time Algorithm for Sampling Words from a Context-Free Language. Information and Computation 134(1), 59–74 (1997)
16. Starkie, B., van Zaanen, M., Estival, D.: The Tenjinno Machine Translation Competition. In: Sakakibara, Y., Kobayashi, S., Sato, K., Nishino, T., Tomita, E. (eds.) ICGI 2006. LNCS (LNAI), vol. 4201, pp. 214–226. Springer, Heidelberg (2006)
17. Starkie, B., Coste, F., van Zaanen, M.: The Omphalos context-free grammar learning competition. In: Paliouras, G., Sakakibara, Y. (eds.) ICGI 2004. LNCS (LNAI), vol. 3264, pp. 16–27. Springer, Heidelberg (2004)
18. de la Higuera, C.: A bibliographical study of grammatical inference. Pattern Recognition 38(9), 1332–1348 (2005)
19. Clark, A.: A learnable representation for syntax using residuated lattices. In: Proceedings of the 14th Conference on Formal Grammar, Bordeaux, France (2009)