Towards General Algorithms for Grammatical Inference

Alexander Clark

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

Abstract. Many algorithms for grammatical inference can be viewed as instances of a more general algorithm which maintains a set of primitive elements, which distributionally define sets of strings, and a set of features or tests that constrain various inference rules. Using this general framework, which we cast as a process of logical inference, we re-analyse Angluin's famous LSTAR algorithm and several recent algorithms for the inference of context-free grammars and multiple context-free grammars. Finally, to illustrate the advantages of this approach, we extend it to the inference of functional transductions from positive data only, and we present a new algorithm for the inference of finite state transducers.

1 Introduction

Grammatical inference (GI) is concerned with learning various types of formal languages. In this paper, we consider two classic problems: the first is where we are learning *languages*, that is to say sets of strings over a finite alphabet — subsets of Σ^*; and the second is where we are learning transductions or functions – relations between pairs of strings — a subset of $\Sigma^* \times \Delta^*$. In the first case we will receive information about the language, typically in the form of a sequence of positive examples drawn from the language; in the second we will receive input-output pairs, and in both cases we wish to construct a representation of the language or function from this information. At a certain point, as the amount of information we have increases, we wish our representation to converge exactly to a correct hypothesis; that is to say, a hypothesis that exactly defines the target concept.

We will work here in the Gold paradigm [1], which is mathematically convenient although unrealistic as a model of learning in the real world. We assume that the learner is provided with a sequence of examples from the target language L subject only to the constraint that every string in the language must occur at least once. We will denote such a sequence $w_1, w_2, \ldots$. After processing each example, the learner must produce a representation — $G_1, G_2, \ldots$. We require that for each such sequence, or presentation, there must be some finite point N after which the learner no longer changes the hypothesis, and such that the hypothesis is correct: $\forall n > N$, $G_n = G_N$ and $L(G_N) = L$.

There are two main problems with learning the sorts of richly structured representations that are required to model natural languages. The first are the sorts of information theoretic problems that have been well-studied in many areas of learning – these problems concern whether the learner has enough information about the target to produce an accurate enough hypothesis. Concepts such as finite elasticity, VC-dimension and so on have been developed in attempts, largely successful, to characterise those cases where learning is impossible simply because there is insufficient information to allow a learner to pick out the right hypothesis. Such results consider the learner primarily as a *function* rather than as an *algorithm*: if there is no suitable function then there can of course not be an algorithm that implements that function.

The second class of problems are caused by the complexity involved in constructing a hypothesis given an adequate source of information about the language [2, 3]. These are two rather different types of problems, and in our opinion it is appropriate to try to solve them separately.

In this paper we will focus almost exclusively on the algorithmic aspects of learning — in an attempt to overcome these complexity problems — and we will therefore give ourselves a rich source of information. In addition to the positive examples we will assume that the learner has access to an oracle that can answer membership queries: the learner can construct a string w and query the oracle with this string; the oracle will return *true* if $w \in L$ and false if it is not. This is an extremely powerful source of information. It is easy to see that if we put no constraints on the amount of computation that we use, there are trivial enumerative algorithms that the learner could use to learn any enumerable class of recursive languages.

2 Objective representations

We will now consider various algorithms for grammatical inference. These algorithms all start by constructing the representation based on objectively defined sets of strings; we discuss the methodological and representational basis of this approach in more detail in [4].

We define representations where the symbols of the representation – nonterminals, states etc – have well defined referents as sets of strings, or sets of tuples of strings. Once we have fixed what each of these symbols represents, we can think of the derivation rules of the grammar, the transitions, or productions etc – as being logical deductive relationships between these sets. Some rules will be valid – in the sense that the deductive relationships will be correct— and others will be incorrect, in the sense that we may deduce that a string is in a set when in fact it is not.

The crucial point is that once we have fixed what each representational primitive is meant to do – that is to say defined what set of strings it should generate or produce – each decision about the model becomes a local decision rather than a global one. In the classic representations of the Chomsky hierarchy, the primitive symbols are arbitrary. There is no fixed definition for what each symbol

need represent. In a normal CFG, if there is a rule $N \rightarrow PQ$, there is no way of evaluating that rule in isolation from the rest of the grammar. It is only as part of a global inference about the rules that generate N, and the strings that are generated from P and from Q that we can decide whether this is a good rule or not. This global inference is generally intractable. In the models we consider here, we define in advance what the symbols should "mean". Once we have done this, we can determine for each indivdual production in the grammar whether it is valid or not. This local inference procedure is tractable and indeed is often quite trivial.

We define now the notation we will use in the rest of the paper. We will use Σ (and Δ) to refer to non-empty finite alphabets. We will use Σ^* to refer to the set of all strings over Σ; λ is the empty string. Given two strings u, v we denote their concatenation by uv. A context (l, r) is an element of $\Sigma^* \times \Sigma^*$; we can combine a context with a string by wrapping it around the string: we denote this by $(l, r) \odot u = lur$. A language L is any subset of Σ^*. Given two subsets of Σ^*, X and Y, we define their concatenation in the normal way as $XY = \{uv | u \in X, v \in Y\}$. Given a language L we define the distribution of u in L as $C_L(u) = \{(l, r) | lur \in L\}$. For a string w we define $Sub(w)$ to be the set of all substrings of w, $\{u | \exists l, r \in \Sigma^*, lur = w\}$, and $Con(w) = \{(l, r) | \exists u \in \Sigma^*, lur = w\}$. We extend this to sets of strings in the natural way.

The models maintain two classes of objects: the first is a set of primitive elements; we will denote these by Q. These correspond to the states or non-terminals in standard representations. The second are a class of tests or experiments, which we denote X. These are used to restrict and eliminate incorrect rules.

Each primitive element from Q will define a set of strings given a language L; or more generally a tuple of strings. For a given element $p \in Q$, we will denote by $\mathbf{D}(p)$ the set of strings defined by p; to avoid confusion we shall write $[[p]]$ to refer to the symbol as used in the representation. The definition of $\mathbf{D}(p)$ will determine what the representation class is; different representational decisions will give rise to different representation classes. In many cases, the elements of p will be strings, and then some of the possibilities for $\mathbf{D}(p)$ are as follows:

$\{w \mid wp \in L\}$	left quotient of p
$\{w \mid pw \in L\}$	right quotient of p
$\{w \mid C_L(w) = C_L(p)\}$	congruence class of p
$\{w \mid C_L(w) \supseteq C_L(p)\}$	

For each algorithm, we will pick one of these; different decisions will give rise to different representation classes. For example, we might pick the first of these; $\mathbf{D}(p) = \{w | wp \in L\}$. This will, as we shall see, lead us naturally to a representation for regular languages. Clearly $\mathbf{D}(\lambda) = L$ no matter what L is. If the language $L = (ab)^*$, then $\mathbf{D}(a) = b(ab)^*$, $\mathbf{D}(b) = \emptyset$, $\mathbf{D}(ab) = L$ and so on.

We might also define Q to be a set of pairs of strings. If we do this, and we write an element as $p = (u, v)$ we might have

$$\mathbf{D}((u,v)) = \{w | uwv \in L\} \quad (1)$$

Given these sets of strings, we can then define a logical derivation relation. We wish to define a set of deductive relations between these sets of strings. Given a language L and a set Q, we will have the family of sets $\mathbf{D}(Q) = \{\mathbf{D}(p) | p \in Q\}$. The simplest deductive relation we could have is this: suppose that $\mathbf{D}(p) \subseteq \mathbf{D}(q)$. Then if we know that a string u is in $\mathbf{D}(p)$, we can deduce that it is in $\mathbf{D}(q)$.

Similarly if we know that u is in $\mathbf{D}(p)$ and we know that v is in $\mathbf{D}(q)$, and we know that $\mathbf{D}(p)\mathbf{D}(q) \subseteq \mathbf{D}(r)$ then we can deduce that $uv \in \mathbf{D}(r)$.

Finally if we know for example, that $\mathbf{D}(p)$ is a subset of L, or in particular if $\mathbf{D}(p) = L$, then if we have deduced that $u \in \mathbf{D}(p)$, then we can deduce that $u \in L$. Obviously these deductions must start somewhere – there must be a few base facts where we know that $u \in \mathbf{D}(p)$; typically we will know these for a few short strings, at least the elements of Σ.

Thus the derived language representations will work by trying to predict, on the basis of these deductive relationships, which elements of $\mathbf{D}(Q)$ a particular string is in. In some cases the elements of this class may be a partition, and a string can only occur in a single element, but in general they may overlap, and a string may be in more than one of the classes. A derivation is therefore a deduction; bringing to mind the "Parsing as deduction" slogan [5]. We assume for the moment that we know which of these deductions are valid.

We will have various different rule schemas. We will now list some of these, though these by no means exhaust all of the possibilities. The notation we use is that p is an element of Q, $[[p]]$ represents the corresponding symbol, and $\mathbf{D}(p)$ represents the set of strings defined by p.

Type L (Lexical)
$[[p]] \rightarrow u$. This allows us to deduce that a string u is in $\mathbf{D}(p)$. It is valid iff $u \in \mathbf{D}(p)$. We will consider two special cases:
L0 $[[p]] \rightarrow \lambda$
L1 $[[p]] \rightarrow a$, where $a \in \Sigma$

Type R (Regular) $[[p]] \rightarrow u[[q]]$. This is valid if $u\mathbf{D}(q) \subseteq \mathbf{D}(p)$. We have the special case:
R1 $[[p]] \rightarrow a[[q]]$, where $a \in \Sigma$

Type LIN (Linear) $[[p]] \rightarrow u[[q]]v$. This is valid if $u\mathbf{D}(q)v \subseteq \mathbf{D}(p)$. We will consider also the following special cases:
Type EL (Even Linear) $[[p]] \rightarrow u[[q]]v$ where $|u| = |v|$
Type EL1 $[[p]] \rightarrow a[[q]]b$ where $a, b \in \Sigma$

Type B (Binary Branching) $[[p]] \rightarrow [[q]][[r]]$. This allows us to deduce that if a string $u \in \mathbf{D}(q)$ and $v \in \mathbf{D}(r)$, then $uv \in \mathbf{D}(p)$. This is valid iff $\mathbf{D}(q)\mathbf{D}(r) \subseteq \mathbf{D}(p)$.

Type S (Subset) $[[p]] \rightarrow [[q]]$. This allows us to deduce that if a string $u \in \mathbf{D}(q)$ then $u \in \mathbf{D}(p)$. This is valid iff $\mathbf{D}(q) \subseteq \mathbf{D}(p)$.

Type E (Equality) $[[p]] \leftrightarrow [[q]]$ or both $[[p]] \leftarrow [[q]]$ and $[[p]] \rightarrow [[q]]$. This allows us to deduce that if a string $u \in \mathbf{D}(q)$ then $u \in \mathbf{D}(p)$ and vice-versa. This is valid iff $\mathbf{D}(q) = \mathbf{D}(p)$.

Type C (Conjunction) $[[p]] \rightarrow [[q]] \wedge [[r]]$. This allows us to deduce that if a string $u \in \mathbf{D}(q)$ and u is also in $\mathbf{D}(r)$ then $u \in \mathbf{D}(p)$. This is valid iff $\mathbf{D}(q) \cap \mathbf{D}(r) \subseteq \mathbf{D}(p)$.

We will also have one final class of rules, where we use the distinguished start symbol S, just as we do in CFGs. In some cases we may have an element i of Q that such that $\mathbf{D}(i) = L$ by definition, in which case we could dispense with these initial rules and the separate symbols.

Type I (Initial) $S \rightarrow [[p]]$. This is valid iff $\mathbf{D}(p) \subseteq L$.

We will have a collection of these rules which we call G: this collection will consist of the representation for the language. Given a collection of primitive elements, Q, we can consider the set of all valid rules that relate these primitive elements. Some of these rule schemas may give rise to unbounded numbers of rules; in particular the first three schemas (**L,R,LIN**) will need to be bounded in some way. Typically we will restrict these rules to those where u has length at most 1.

An important aspect of this model is the use of the **E** rules. In many of these models we will have a set of primitive elements Q, where several different elements of Q will define the same sets; that is to say we might have $p, q \in Q$, and $\mathbf{D}(p) = \mathbf{D}(q)$. A natural way to handle this is to consider the primitive elements to be equivalence classes of Q, under equality of $\mathbf{D}(q)$; or equivalently considering them to be the elements of $\mathbf{D}(Q)$. This certainly gives some efficiency gains when implementing them. However, Yoshinaka [6] introduced the idea of using "chain rules" – these equality or **E** productions — to link distinct elements of Q that define the same strings. This greatly simplifies the analysis of the algorithms, and while it causes some decrease in efficiency it does not change the polynomial nature of the algorithms. Accordingly we will adopt this refinement.

For a given decision about what $\mathbf{D}(p)$ is we can divide these rule schemas into three types. There are those that we can be sure are correct, as a result of the way $\mathbf{D}(p)$ has been defined – we will call these rules *a priori*. There are those that we are certain are correct as a result of information that has already been received — we will call these rules *certain*. Finally there are those which we believe to be correct but might later turn out to be incorrect on the basis of further information – we will call these *defeasible*. Defeasible rules will be assumed to be correct until we receive a piece of information that tells us that the rule is incorrect. Once we have seen such a piece of information, we will be certain that the rule is incorrect, and no further information will cause us to change our mind – once we know that a rule is incorrect then it is definitely wrong, and until that point we will consider it to be true, though uncertain.

For example, if we have a rule of the type $[[p]] \rightarrow u$, then it might be the case that $\mathbf{D}(p)$ is defined to include u; for example p might be u, and $\mathbf{D}(p)$ might be an equivalence class that includes u. This would be a case where the rule is *a priori*. Alternatively, we might have received information that tells us definitely that $u \in \mathbf{D}(p)$; for example, $\mathbf{D}(p)$ might be defined to be the set of all strings

that occur in a given context, in which case, if we know that that string occurs in a given context then we will be certain, as a result of this single piece of information, that the rule is correct.

Finally, and most importantly, there may be rules that no finite amount of information can make certain – the defeasible rules. In general, checking these rules will involve checking potentially infinite sets. For example, a rule of the form $[[p]] \rightarrow [[q]]$, is only valid if $\mathbf{D}(q) \subseteq \mathbf{D}(p)$ and these two sets will in general be infinite. In general given only a finite amount of information it will always be possible that the language is different from what we would expect. There are an infinite number of possible languages, and it might be that the language does not include some very long strings that we would expect it to contain based on the examples we have seen. This is a possibility that we can never conclusively exclude, at least in the learning models that we consider here.

We therefore use a finite set of experiments that we denote by X, and that we will gradually increase during the learning process. Typically, X will be a set of strings or contexts that we will use to test the validity of rules. In one formalisation, X is a set of strings and we can test whether $\mathbf{D}(q) \subseteq \mathbf{D}(p)$ by testing $X \cap \mathbf{D}(q) \subseteq X \cap \mathbf{D}(p)$. Clearly if the inference is invalid, then when X is sufficiently large we will detect this fact: if it is not the case that $\mathbf{D}(q) \subseteq \mathbf{D}(p)$, then there must be some $x \in \mathbf{D}(q) \setminus \mathbf{D}(p)$, and if $x \in X$ we will detect it by testing whether $X \cap \mathbf{D}(q) \subseteq X \cap \mathbf{D}(p)$.

Initially, we assume that all possible defeasible rules are valid unless they are explicitly contradicted by a piece of information; typically an element or elements of X. We will start with X being either empty or consisting of a small set of elements, often only one. We will monotonically increase X based on the examples that we observe from the language we are trying to learn. During the course of the algorithm X will generally be increased without limit.

As we increase X we will remove incorrect defeasible rules; correct defeasible rules will never be removed. For each incorrect defeasible rule it will suffice to find a single element of X that will remove that rule; therefore in the best case we only need to have an X that is of the same cardinality as the set of possible rules, which will typically be bounded by a polynomial in the size of Q. Whether this is possible or not depends on the learning model; under the Minimally Adequate Teacher (MAT) model we will receive counter-examples [7] and generally we can construct a suitable element of X from the counter-example. If we have only positive examples, we can increase X without limit. The larger X is the more incorrect rules we will remove. The only limit is that the size of X be bounded appropriately so that the overall algorithm is efficient.

We therefore have a deductive system or grammar G that we construct from information about a language L using a defined set of primitive elements Q and a set of tests or experiments X. We will write $G(Q, X, L)$ for this system. Typically L is fixed, and so we will write this as $G(Q, X)$, but it is implicitly used in the definition since we will use an oracle for L when constructing the system G.

3 Derivation

Having constructed this inference system, we are clearly interested in using it to infer properties of novel strings that we have not observed before. Given any string w we wish to be able to tell whether it is in a particular class, and indeed whether it is in L or not; therefore we wish to be able to say for any primitive element q and any string w whether $w \in \mathbf{D}(q)$ or not.

We now consider a derivation relation or proof. Clearly we can chain these inferences together in the natural way. The inference rules allow us to deduce that a long string is in a particular class from the fact that its substrings are in another class; using this, together with a set of rules that tell us which classes the strings of length 1 are will allow us to construct efficient inference procedures. This is a standard insight from logical grammar formalisms; [5]. Thus we consider the grammatical formalisms here to be inference systems between distributionally defined sets of strings. We will not formalise the inference system using a sequent calculus; this seems unnecessarily complex.

These deductive procedures turn out to be the same as the derivation procedures in various forms of grammars, such as context free grammars, conjunctive grammars and so on. Standard techniques for dynamic programming can be used to compute these efficiently. In particular, for a given string w, we will construct a table which maintains for each string u which is a substring of w, a list or set of the elements of Q, p such that we have a proof that u is in $\mathbf{D}(p)$. This can be done in time polynomial in the length of the string and the size of Q.

We will write $[[p]] \stackrel{*}{\Rightarrow}_G w$ if there is a proof that $w \in \mathbf{D}(p)$ using steps in the set of productions or rules of the grammar G. If all of the inference steps are valid then it is clear that we will only deduce that $[[p]] \stackrel{*}{\Rightarrow}_G w$ if w is in fact in $\mathbf{D}(p)$. As a special case, we will only have a proof $S \stackrel{*}{\Rightarrow}_G w$ if w is in L.

Lemma 1. *If all of the inference steps in P are valid, then if $[[p]] \stackrel{*}{\Rightarrow} w$ then $w \in \mathbf{D}(p)$.*

Proof. Immediate by induction on the length of the proof; if each inference is valid.

Clearly as we increase Q, and we assume that all of the rules are valid, then the language will only increase, as the set of rules will increase.

The next step is to establish that the rules are in fact valid. First of all note that as we increase the size of X, the set of tests, the set of rules will monotonically decrease as we will remove defeasible rules. The following lemma is thus immediate.

Lemma 2. *If $X_1 \subseteq X_2$ then $L(G(Q_1, X_1)) \supseteq L(G(Q_1, X_2))$*

Morover, given that there are only a finite number of defeasible rules, at some point we will have removed all incorrect rule. We formalise this as a property of the set of experiments which we call *fiduciality*.

Definition 1. *A set of experiments X is fiducial for a set of primitive elements Q iff every rule is valid; i.e. all incorrect defeasible rules have been removed.*

As we increase the number of primitive elements, the set of rules will monotonically increase, even if some of them are not valid.

Lemma 3. *If $Q_1 \subseteq Q_2$ then $L(G(Q_1, X)) \subseteq L(G(Q_2, X))$*

Moreover if we have X_1 fiducial for Q_1 and X_2 fiducial for Q_2 then $L(G(Q_1, X_1)) \subseteq L(G(Q_2, X_2))$.

We now define a very fundamental idea; at some point the set of primitive elements may be large enough, so that the set of correct rules will define the language. When we reach this point, if we have some additional incorrect defeasible rules then we will overgenerate; indeed no matter how large or small X is, we will always define a language which includes the target language. We formulate this idea as follows:

Definition 2. *A finite set of primitive elements Q is a kernel for the target language L, if the set of valid rules derived from Q is sufficient that for every string $w \in L$, we have a proof using these rules that $S \stackrel{*}{\Rightarrow} w$.*

An easy consequence of the definition is that given any language L, then any $G(Q, X)$ where Q is a kernel for L will define a language that includes the target language L – since we will have enough correct rules, and possibly some defeasible incorrect rules if X is too small.

For any specific algorithm, the learnable class that we will have will be defined as the set of languages which have a finite kernel. This is, broadly speaking, the set of languages that can be finitely defined under the representational assumptions that we make. As we shall see, for the case of regular grammars, the class corresponds exactly to the class of regular languages, but for other representation classes, the classes of languages do not correspond precisely to existing language classes.

We will consider various specific models: but they all satisfy the following criteria.

- As we increase X, the set of rules will monotonically decrease.
- No correct rules will be removed by increasing X
- Every incorrect rule will be removed.
- As we increase Q, the set of rules will monotonically increase.
- We can perform all of the computations in polynomial time. In particular we can compute the derivation relations $S \stackrel{*}{\Rightarrow}_G w$ in time polynomial in the size of the rule set and the length of w.

4 Generic algorithms

We can now define a generic algorithm for inferring these representations. We will use the paradigm of identification in the limit from positive data and (optionally)

membership queries which is easy to handle and quite permissive. Algorithm 1 receives a stream of positive examples, and may use a membership oracle O. It calls several functions:

- **InitQ** returns an initial set of primitive elements.
- **InitX** returns an initial set of experiments.
- **Make** constructs a representation from the available data. We will consider this to be an inference system and thus a collection of rules.
- **IncreaseQ** returns a set of primitive elements
- **IncreaseX** returns a set of experiments.

These must satisfy the following conditions: if there is a kernel for a language, then **IncreaseQ** must eventually return a set that includes a kernel. Secondly, for any incorrect defeasible rule, **IncreaseX** must eventually produce an element that will remove it. **Make** simply produces all possible rules from Q and then removes those that are contradicted by elements of X.

```
1  E ← ∅ ;
2  Q ← InitQ ;
3  X ← InitX ;
4  G ← Make(Q, X, O, E) ;
5  while w_i is a positive example do
6  |  E ← E ∪ {w_i} ;
7  |  for w ∈ E do
8  |  |  if not S ⇒*_G w then
9  |  |  |  Q ← Q ∪ IncreaseQ(E) ;
10 |  X ← X ∪ IncreaseX(w_i) ;
11 |  G = Make(Q, X, O, E) ;
```

Algorithm 1: Generic meta-algorithm for learning from positive data and membership queries. O is a membership oracle for the language.

We can now see that given a specific set of representational assumptions that the algorithm defined here will identify in the limit any language which has a finite kernel. We state the theorem given some set of definitional assumptions, and given definitions of the subroutines called by the algorithm.

Theorem 1. *Algorithm 1 will identify in the limit any language with a finite kernel.*

Proof. We will use E_n, Q_n, X_n, G_n to refer to the state of the variables at iteration n. Note first that if there is some n such that Q_n is a kernel it will never change, and the grammar will always define a superset of the target language. If at some point n there is a kernel then at some point $n' > n$, all incorrect defeasible rules will have been removed, and at that point $G_{n'}$ will be correct and will never change. So we merely need to show that at some point we will

get a kernel. If L has a finite kernel, then let N be the smallest number such that **IncreaseQ**($\{w_1, \ldots w_N\}$) is a kernel. Suppose Q_N is not a kernel. Suppose $L(G_N)$ does not include L, then at some point $n \geq N$ we must find a w_n which will call line 9, and after that point Q is a kernel. Alternatively, suppose $L(G_N)$ does include L, and since it is not a kernel it must include some incorrect rules. Then either there is some point when all incorrect rules will be removed, at which point the hypothesis will under-generate and we will observe a string which will trigger line 9, or line 9 will be triggered before that point, and in either case we end up with a kernel.

5 Regular languages

We will now make this rather abstract discussion concrete by producing a reconstruction of Angluin's celebrated LSTAR algorithm in this model. We will not try to make this algorithm efficient; it will be polynomial, but we will not constrain the representation to be deterministic and as such the algorithm will be much less efficient that the LSTAR algorithm and will not have the elegant algorithmics of that approach.

Our representational primitives will be strings, and so Q will be a finite set of strings that will correspond to prefixes of the language. Each prefix w will define a quotient or residual language as follows:

$$\mathbf{D}(w) = \{v | wv \in L\} \tag{2}$$

In terms of a DFA, each element of Q will therefore correspond to a state q. If we let Q_* denote the set of states of a minimal DFA that generates the languages L_*, then for each $w \in Q$, we will have a corresponding state in the DFA which will be the state $\delta(q_0, w)$, using standard notation.

We will use the following rule schemas:

I $S \rightarrow [[\lambda]]$
R $[[w]] \rightarrow v[[wv]]$ if $w, wv \in Q$
L0 $[[w]] \rightarrow \lambda$ if $w \in L$
E $[[w]] \rightarrow [[v]]$ iff $\mathbf{D}(w) \cap X = \mathbf{D}(v) \cap X$

Note that each of these four rules have slightly different properties. The first two rule schemas are universally valid – we know that they are correct *a priori* without using any evidence from the oracle. The first schema is clearly correct since by definition $\mathbf{D}(\lambda) = L$. The second schema is correct since $v\mathbf{D}(wv) \subseteq \mathbf{D}(w)$. The "proof" is trivial: suppose $u \in \mathbf{D}(wv)$; this means that $wvu \in L$. Therefore $vu \in \mathbf{D}(w)$.

The third rule schema is also certain, but uses information from the oracle. If $w \in L$ then $\lambda \in \mathbf{D}(w)$, but it is only when we have tested the example w for membership that we will know that the rule is valid.

The final **E** schema is non-trivial: it uses information from the oracle but is defeasible. Given two strings in Q, w and v we will assume that they are

equivalent, (i.e. $\mathbf{D}(w) = \mathbf{D}(v)$) unless we observe some string $s \in X$ such that $ws \in L$ and $vs \notin L$ or vice-versa. Once we observe such a string then we remove this equality rule, as we know it is not valid.

Given a membership oracle for a language L, a finite set of strings Q and a finite set of test suffixes X, we can construct a regular grammar based on these rules schemas in time polynomial in the size of Q and X.

Let us now consider the notions of *kernel* and *fiduciality* in this concrete case.

Lemma 4. *The class of languages that have a finite kernel in this case is equal to the class of regular languages.*

Proof. Clearly if it has a finite kernel then it is regular since it will be correctly defined by a regular grammar. Conversely if L is a regular language, then consider a minimal DFA for L. A finite set of strings Q is a kernel for L if Q contains one string for each state in the minimal DFA and a string for each transition. That is to say for each transition $q \rightarrow^a q'$ there is a string u and a string ua in Q such that $\delta(q_0, u) = q$ and $\delta(q_0, ua) = q'$.

Thus the idea of a kernel is closely related to that of a structurally complete sample as defined in for example [8]. Indeed, the set of prefixes of a structurally complete sample for an automaton will be a kernel for the language defined by that automaton.

Lemma 5. *A set X is fiducial for a set of primitives Q iff for every pair of strings that are not congruent there is an element of X that is in the symmetric difference of their quotient languages.*

5.1 Even linear grammars

Recall that an even linear grammar (ELG) is a CFG where all of the productions are either of the form $X \rightarrow uYv$ where $u, v \in \Sigma^+$ and $|u| = |v|$, or of the form $X \rightarrow u$ where $|u|$ is even. We can clearly convert these to regular grammars by "folding" them over and mapping them to automata over an alphabet consisting of pairs of letters [9].

We can also can model them directly in this approach by considering the primitive elements Q to be pairs of strings (u, v) where $|u| = |v|$, and considering the experiments X to be strings of even length.

6 Congruence based approaches

Let us now move onto the theory of context free grammatical inference, in particular the theory of congruence based approaches as explored in [10, 11, 6, 12].

The most basic of these models, presented in [12] makes the representational assumption that the non-terminals of the gramar generate congruence classes of the language.

Recall that the syntactic congruence is defined as the relation $u \equiv_L v$ iff $C_L(u) = C_L(v)$. We will define Q as a set of strings, and for each $u \in Q$, we define $\mathbf{D}(u) = \{w : C_L(u) = C_L(w)\}$. These are the congruence classes of the language L. The set of experiments X will be a finite set of contexts; i.e. $X \subset \Sigma^* \times \Sigma^*$.

We will therefore have the following families of rules

B $[[uv]] \to [[u]][[v]]$
L0 $[[a]] \to a$
L1 $[[\lambda]] \to \lambda$
E $[[u]] \to [[v]]$ iff $C_L(u) \cap X = C_L(v) \cap X$
I $S \to [[u]]$ iff $u \in L$

In terms of the meta-algorithm 1 we will define the following functions:

InitQ returns $\Sigma \cup \{\lambda\}$
InitX returns $\{(\lambda, \lambda)\}$
Make Returns a CFG with the productions defined above
IncreaseQ returns $Sub(E)$
IncreaseX returns $Con(E)$

In [12], a similar algorithm was shown to polynomially learn the class of congruential CFGs from a minimally adequate teacher (MAT).

7 Dual CFG representations

In regular inference we are concerned with the relation between the prefix and the suffix. Given a language L, we define a relation $u \sim_L v$ iff $uv \in L$: the dual relation is basically the same except that we swap prefixes and suffixes. This leads to representations where we have finite automata that read from right to left – this is uninteresting.

In distributional learning we can find that there is a partial duality between the context and the substring. We will now consider a dual representation, where the primitive elements are contexts, and the set of experiments X is a set of substrings.

We will consider Q as a finite set of contexts, i.e. a subset of $\Sigma^* \times \Sigma^*$, and we shall assume that $(\lambda, \lambda) \in Q$. We now define, for a context $p = (l_p, r_p)$ in Q

$$\mathbf{D}((l_p, r_p)) = \{u | l_p u r_p \in L\} \tag{3}$$

Note that $\mathbf{D}((\lambda, \lambda)) = L$. We will have as before various classes of rules. The defeasible class of rules is thus the class of binary rules of the form $[[p]] \to [[q]][[r]]$. We will test these using the following criterion.

$(\mathbf{D}(q) \cap X)(\mathbf{D}(r) \cap X) \subseteq \mathbf{D}(p)$

We can test this simply using a membership oracle by checking for each $u, v \in X$ such that $q \odot u \in L$ and $r \odot v \in L$, and if they are then we test whether $p \odot (uv) \in L$; if this is not the case then we remove the rule. Otherwise we include the rule.

The basic rules are thus

I $S \rightarrow [[(\lambda, \lambda)]]$
L1 $[[p]] \rightarrow a$ iff $p \odot a \in L$
L0 $[[p]] \rightarrow \lambda$ iff $p \odot \lambda \in L$
B $[[p]] \rightarrow [[q]][[r]]$ iff $(\mathbf{D}(q) \cap X)(\mathbf{D}(r) \cap X) \subseteq \mathbf{D}(p)$

Only the final rule is defeasible. we need to show that all and only the incorrect rules will be removed. Suppose that the rule $[[p]] \rightarrow [[q]][[r]]$ is incorrect. Then this means that $\mathbf{D}(q)\mathbf{D}(r)$ is not a subset of $\mathbf{D}(p)$. Therefore there are strings u, v such that $u \in \mathbf{D}(q)$ and $v \in \mathbf{D}(r)$ but $uv \notin \mathbf{D}(p)$.

If we define $\mathbf{IncreaseX}(E)$ to return the set of all substrings in E, then clearly once E includes $q \odot u$ and $r \odot v$, this incorrect rule will be removed. The converse is obvious; if the rule is valid then $(\mathbf{D}(q) \cap X)(\mathbf{D}(r) \cap X) \subseteq \mathbf{D}(p)$ is always true even when $X = \Sigma^*$. Note that the class of languages learnable is rather different as it will include non-deterministic and inherently ambiguous ones, whereas the algorithm of Section 6 appears to only include deterministic languages.

This algorithm corresponds to the algorithm defined in [13] restricted to the case where we consider only single contexts. It is instructive to contrast the primal, congruence-based algorithm with this dual algorithm for context-free inference. For the primal representation, the **B** rules are *a priori* and the **E** rules are defeasible and the **L** and **I** rules are certain; for the dual representation, the **B** rules are defeasible, the **L** rules are certain, the **I** rules are *a priori*, and we do not use **E** rules.

Model	Q	X	$\mathbf{D}(p)$	Rules	Class
Angluin [7]	Σ^*	Σ^*	$\{w \vert pw \in L\}$	**L0, R,E**	DFA
Sempere [9]	$\Sigma^k \times \Sigma^k$	Σ^*	$\{w \vert p \odot w \in L\}$	**L0, EL1, E**	ELG
Clark et al. [14]	Σ^*	$\Sigma^* \times \Sigma^*$	$\{w \vert C_L(w) \supseteq C_L(p)\}$	**L, S,B,C**	BFG
Clark [12]	Σ^*	$\Sigma^* \times \Sigma^*$	$\{w \vert C_L(w) = C_L(p)\}$	**L1,L0,B,E**	CFG
Clark [13]	$(\Sigma^* \times \Sigma^*)^{\leq k}$	Σ^*	$\{w \vert C_L(w) \supseteq p\}$	**L1,L0,B**	CFG
Clark [15]	$(\Sigma^* \times \Sigma^*)^*$	Σ^*	$\{w \vert C_L(w) \supseteq p\}$	**L1,L0,B,S,C**	DLG
Yoshinaka [16]	$(\Sigma^*)^{\leq k}$	$(\Sigma^*)^{\leq (k+1)}$	$\{\mathbf{w} \vert C^k_L(\mathbf{w}) \supseteq C^k_L(\mathbf{p})\}$	**E, L, B+**	MCFG
Oncina [17]	Σ^*	$\Sigma^* \times \Delta^*$	$(p, lcp(\tau_L(p\Sigma^*)))^{-1}L$		OSST
This paper	$\Sigma^* \times \Delta^*$	$\Sigma^* \times \Delta^*$	$(u, v)^{-1}L$		FST

Table 1. Table showing the basic representational assumptions of the models. All models also have **I** rules, so we omit them. In Yoshinaka's algorithm for MCFGs, the range of **B** rules used is much wider. The final column gives the class of representation that is used. OSST stands for onward subsequential transducer.

8 Distributional Lattice Grammars

Distributional Lattice Grammars [15] are an algorithmically more refined version of these approaches, which allow efficient learning and inference even when we

have an exponentially large set of primitive elements Q. The starting point is the dual CFG approach; we start with a finite set F of contexts that includes the empty context (λ, λ). The primitive elements are not, however, these individual contexts but rather the set of all subsets of F. Thus rather than taking for a context $(l, r) \in F$ the set of strings $\{u|lur \in L\}$, we take as our primitive element f a subset of F, say $f = \{(l_1, r_1), \ldots (l_k, r_k)\}$, and define

$$\mathbf{D}(f) = \{u|l_1ur_1 \in L \wedge \cdots \wedge l_kur_k \in L\} \tag{4}$$

In other words, Q is just the power set of F; each element of Q is a subset of F. This clearly allows a much greater representational power: the sets of strings that we define thus correspond to finite intersections of the sets defined by individual contexts. This allows us to represent a larger class of languages. Consider for example the language $\{a^nb^n|n \geq 0\} \cup \{a^nb^{2n}|n \geq 0\}$; this language cannot be represented using sets that are defined by a single context, because the relevant sets of strings, such as $\{a^nb^n|n \geq 0\}$ are not defined by a single context. For example, the context (a, b) defines a set of strings that includes $\{a^nb^n|n \geq 0\}$ but also includes many other strings such as $\{abbb, aabbbbb \ldots\}$. However if we allow our primitives sets to be defined by pairs of contexts, then the pair $(a, b), (aa, bb)$ will succesfully pick out, "triangulate" in a sense, the relevant set of strings. One approach to this, taken in [13] is simply to stipulate a maximum cardinality for the sets of contexts, and consider all sets of contexts of cardinality less than this. Considering this upper bound as fixed, the set of primitive elements becomes polynomial.

This avoids the problem rather than solves it; for natural language, it is essential to recognise that the syntactically and linguistically relevant sets of strings may require a large number of contexts to pick them out precisely.

An important insight of this approach is that there is a natural lattice structure that arises in these forms of learning. Since each primitive element p in Q defines a *set*, $\mathbf{D}(p)$ we can see that there will inevitably be a lattice structure generated by this set. Once we realise this, then it is natural to extend the set of primitive elements, by looking at the meet semi-lattice generated by the set $\{\mathbf{D}(p)|p \in Q\}$, and augmenting the inference system with conjunctive rules (those of type $\mathbf{C}$ above).

DLGs take this path and for computational reasons it turns out to be essential to add conjunctive rules. Given these rules, we can compute for every string w, the set of all sets that it must be a member of $Y(w) = \{\mathbf{D}(p)|w \in \mathbf{D}(p)\}$. The crucial observation is this: given that this is a lattice, rather than considering all of the exponentially many elements of this set of sets, we can sum it up in a single element; namely $\bigcap_{s \in Y(w)} s$. If w lies in all of the sets in $Y(w)$ then it must lie in their intersection. Since we have extended our set of primitive elements so it is a meet semi-lattice, (in fact a full lattice in the case of DLGs), then this intersection element will be in our set Q. Thus though w may be a member of very many sets defined by elements of Q, computationally we can consider just this unique one: the smallest set that we can prove it is in.

The addition of the conjunctive rules increases the power of the formalism to that of Conjunctive Grammars [18], or more precisely to a subclass of the

languages definable by conjunctive grammars. This insight, though it has so far only been applied to the theory of CFG inference, is of more general application, and we think it can potentially be applied to all of the models discussed here.

9 MCFGs

Yoshinaka [19, 16] shows how we can extend this model to the inference of Multiple Context-Free Grammars [20]. We fix a natural number constant p and define the set Q to be a set of tuples of strings $(u_1, \ldots, u_m)$ where $1 \leq m \leq p$. X is then a set of generalised contexts which again are tuples of strings, this time of arity up to $p+1$.

Given a tuple $\mathbf{w} = (u_1, \ldots, u_m)$ we can define the generalised distribution with respect to a language L to be the set
$C_L^m(\mathbf{w}) = \{(v_1, \ldots, v_{m+1}) | v_1 u_1 v_2 \ldots v_m u_m v_{m+1} \in L\}$. The representational assumption of Yoshinaka's algorithm is thus

$$\mathbf{D}(\mathbf{w}) = \{\mathbf{u} | C_L^m(\mathbf{u}) \supseteq C_L^m(\mathbf{w})\} \tag{5}$$

Since the elements are no longer strings, but rather tuples of strings, the ways in which they can be combined are significantly more complex. Rather than one **B** rule, we have a whole family of such rules, each corresponding to a different combination operation.

It is an open question whether the class of DLGs is sufficiently expressive to represent the class of natural languages, or whether it will be necessary to move into the MCFL hierarchy. It might be that even if DLGs are sufficiently expressive, one might still want to use MCFGs because of the slightly richer notion of dependency that they allow, which might permit a more principled modeling of certain movement phenomena in natural languages.

10 Transductions

We now turn our attention to the study of transductions or bilanguages. We assume that we have two alphabets Σ and Δ which may or may not be disjoint, and rather than a language we are interested in *bilanguages* or *transductions* which are subsets of $\Sigma^* \times \Delta^*$; we will write an element of a bilanguage T as an ordered pair (u, v) where $u \in \Sigma^*$ and $v \in \Delta^*$. As defined like this, there is a symmetry but we will often be interested in the cases where L considered as a relation between Σ^* and Δ^* is a function. We say that a transduction T is *functional* if $(u, v), (u, w) \in T$ implies that $v = w$. We say that a transduction is total if for all $u \in \Sigma^*$ there is a $v \in \Delta^*$ such that $(u, v) \in T$.

If we are not interested in functional transductions, then this reduces to a special case of the learnability of multiple context free languages, subclasses of which can be learned directly using results already published [16]. However, as is demonstrated by the well-known OSTIA algorithm [17], if we assume that the data is functional, then we do not need to have membership queries or access

```
1  E ← ∅ ;
2  Q ← InitQ ;
3  X ← InitX ;
4  G ← Make(Q, X, E) ;
5  while (u_i, v_i) is a positive example do
6  |  E ← E ∪ {(u_i, v_i)} ;
7  |  for (u, v) ∈ E do
8  |  |  if not S ⇒*_G (u, v) then
9  |  |  |  Q ← Q ∪ IncreaseQ(E) ;
10 |  X ← X ∪ IncreaseX(w_i) ;
11 |  G = Make(Q, X, E) ;
```

Algorithm 2: Generic meta-algorithm for learning functional transductions from positive data.

to negative evidence, as the positive examples are restricted enough to learn the function.

We will consider now the case where we wish to infer a representation for a total function T. We will define the function $\tau_T : \Sigma^* \to \Delta^*$ as $\tau(u) = v$ where $(u, v) \in T$.

We will start by considering a basic model analogous to that of regular languages. We start by noting that our model above assumed only that the language was a subset of a monoid. Note that we clearly have a natural monoid structure over $\Sigma^* \times \Delta^*$, where $(u, v) \circ (u', v') = (uu', vv')$, and (λ, λ) is the identity. Thus we can now immediatelt lift our analysis to the case where $\mathbf{D}(p)$ is defined to be a subset of $\Sigma^* \times \Delta^*$.

We start by defining our sets of primitive elements Q to be a finite set of pairs $(u, v) \in \Sigma^* \times \Delta^*$, and assume further that $(\lambda, \lambda) \in Q$. We define for a given element $p = (u_p, v_p) \in Q$

$$\mathbf{D}((u_p, v_p)) = \{(u', v') | (u_p u', v_p v') \in T\} \tag{6}$$

We define the following rule schemas:

I $S \to [[(\lambda, \lambda)]]$ which is *a priori*
R $[[(u, v)]] \to (u', v')[[(uu', vv')]]$, also *a priori*
L0 $[[(u, v)]] \to (\lambda, \lambda)$ iff $(u, v) \in T$. This is certain.

The defeasible rule schema will be **E** rules. We will only use positive data here which is sufficient, since if we observe a pair (u, v) then we know that for all $v' \neq v$ that $(u, v') \notin T$, since T is a function. We will therefore have X being a set of pairs that are a subset of T. We will say that an equality rule $[[(u, v)]] \to [[(u', v')]]$ is *incorrect* with respect to X if there is a pair of elements of X of the form $(ux, vy), (u'x, w')$ such that $w' \neq v'y$. Note that these two elements of X need not be distinct, as we shall see below. If it is not incorrect w.r.t X then we say it is correct w.r.t. X. The **E** rule schema is thus:

E $[[(u,v)]] \rightarrow [[(u',v')]]$ iff it is correct w.r.t. X.

If the rule is correct, and $(ux, w), (u'x, w') \in X$, and suppose that $w = vy$, then $(x, y) \in \mathbf{D}(u, v)$ and therefore $(x, y) \in \mathbf{D}(u', v')$ and so $(u'x, v'y) \in T$ and so $w' = v'y$ since it is functional, and therefore it will be correct w.r.t any X On the other hand, if a rule is incorrect, then there must be some $(x, y) \in \mathbf{D}(u, v) \setminus \mathbf{D}(u', v')$, or $\mathbf{D}(u', v') \setminus \mathbf{D}(u, v)$. If we assume w.l.o.g. the first, then we know that $\tau(ux) = vy$. Let $w = vy$ and $w' = \tau(u'x)$; w' cannot be equal to $v'y$ since this would mean that $(x, y) \in \mathbf{D}(u', v')$ which would be a contradiction. Therefore if X contains the two pairs $(ux, \tau(ux)), (u'x, \tau(u'x))$ then this rule will be incorrect w.r.t. X.

We are only interested in those cases where $(u, v)^{-1}T$ is non-empty. This set could be empty, if for example all of the strings that start with u are mapped to strings that start with a, and v starts with b. The algorithm we consider will only add (u, v) when we have observed them as a prefix of a string, and so we will assume in what follows that there is always at least one element of $(u, v)^{-1}T$. There are is a special cases that we should note: when $u = u'$ and $v \neq v'$ clearly these pairs will not be congruent. In this case, let (x, y) be some element of $(u, v)^{-1}T$; then clearly (ux, vy) is a suitable element of X to show that these pairs are not congruent.

We have implicitly defined **Make**; we now define the other subroutines. **IncreaseX** just returns the current data E, and **IncreaseQ** will return the set of all prefixes of E. We say that (u, v) is a prefix of (u', v') if there is some (x, y) such that $(u, v) \circ (x, y) = (u', v')$. The initialisation functions just set X to the empty set and Q to $\{(\lambda, \lambda)\}$. There is one detail we neglect in this informal presentation: we also need to deal with the **L0** rules. Since we are not using an oracle, we set them when we observe the relevant pair (u, v) in the data.

Therefore Algorithm 2 will learn the class of all such transductions with a finite kernel. Let us pause and consider the class of transductions that have a finite kernel; these will clearly be a class of rational functions. These clearly include all subsequential functions, which are those where the underlying automaton is deterministic, and there is a final output function σ. The role of $\sigma(q)$ is played by λ-transitions leading to accepting states. However this clearly also includes non-deterministic transductions. We consider now the classic example of such a transduction (c.f. [21]).

Example 1. Suppose we have $\Sigma = \{a\}, \Delta = \{b, c\}$, and $T = \{(a^{2n}, b^{2n}) | n \geq 0\} \cup \{(a^{2n+1}, c^{2n+1}) | n \geq 0\}$. This is clearly a total function; a^n is mapped to b^n if n is even and to c^n if n is odd.

A kernel for this transduction is the following set Q:

$$\{(\lambda, \lambda), (a, b), (aa, bb), (aaa, bbb), (a, c), (aa, cc), (aaa, ccc)\}$$

We can easily verify that $(a, b) \equiv_T (aaa, bbb)$ and that $(a, c) \equiv_T (aaa, ccc)$. It is easy to convert this to a finite state transducer (FST). Each element of Q corresponds to a state; accepting states are those with a rule of type **L0**. The

initial state is $[[(\lambda, \lambda)]]$. Rules of type **R** such as $[[(u, v)]] \rightarrow (u', v')[[(uu', vv')]]$ are written as a transition $[[(u, v)]] \stackrel{u':v'}{\rightarrow} [[(uu', vv')]]$. Figure 1 shows a minimal set of primitive elements and transitions that define this transduction. The actual output from the algorithm would contain many more states and transitions, but would nonetheless not over-generate.

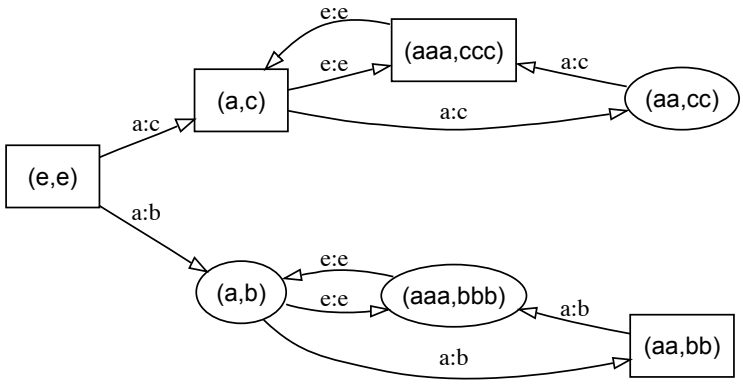

Fig. 1. State diagram for $T = \{(a^{2n}, b^{2n}) | n \geq 0\} \cup \{(a^{2n+1}, c^{2n+1}) | n \geq 0\}$. The "e" stands for the empty string λ; accepting states are drawn as rectangles. We write this as a finite state transducer rather than a rewriting system.

The relation to the OSTIA algorithm is not entirely clear. The OSTIA algorithm infers a class of subsequential transducers; these are deterministic on the input string: thus we can define an equivalence relation of finite index on the set of pairs. If we define the longest common prefix of a set of strings to be *lcp*, the primitives of the OSTIA algorithm are of the form $(u, lcp(\tau(u\Sigma^*)))$, extending τ to sets of strings in the standard way. As can be seen from Figure 1, the output of the algorithm here is not deterministic on the input string. It appears that the class of transductions that can be learned includes all rational functions, but this must remain a conjecture at a moment.

11 Discussion

We have presented a common framework which allows us to see many different models and algorithms, at a suitable level of abstraction, as instances of the same meta-algorithm. In Table 7 we lay out, somewhat crudely the range of representational assumptions of the models that we have looked at in this paper. We

have presented a meta-algorithm quite generally. As a result the specific algorithms are significantly less efficient that they could be. Compare, for example, the elegant algorithmics of the LSTAR or OSTIA algorithms with the very blunt approach taken in this paper. Nonetheless, this rather abstract presentation has allowed us to see that many classic and recent algorithms for GI are variants of the same algorithm. Using these methods allows us to see the range of possible new algorithms and GI techniques that result from combinations of different representational assumptions and sets of rules.

The example of a transduction learning algorithm is meant to show the advantages of this approach – applying this to the problems of learning transductions or functions immediately gives us a new and powerful algorithm for learning regular functions that extends previous results. There is also a natural extension to context-free transductions that we will present elsewhere.

The resulting algorithms are polynomial in the sense that each iteration requires only polynomial time. This is, perhaps, not strict enough a criterion on its own, but in some cases (e.g. [12]) we can get a result under the MAT model.

The general approach we advocate is ultimately very simple – a decision about what each representational element should mean; given this, we can define a set of valid inferences; invalid inferences can be removed based on testing an increasingly large set of experiments. The overall effect is a large and growing family of efficient algorithms for many classic problems in grammatical inference.

Acknowledgments

I am very grateful to Shalom Lappin and Ryo Yoshinaka for comments and corrections on a draft of this paper; any remaining errors are of course my own.

References

1. Gold, E.M.: Language identification in the limit. Information and control **10**(5) (1967) 447 – 474
2. Kearns, M., Valiant, G.: Cryptographic limitations on learning boolean formulae and finite automata. JACM **41**(1) (January 1994) 67–95
3. Angluin, D., Kharitonov, M.: When won't membership queries help? J. Comput. Syst. Sci. **50** (1995) 336–355
4. Clark, A.: Three learnable models for the description of language. In Adrian-Horia Dediu, Henning Fernau, C.M.V., ed.: Language and Automata Theory and Applications, Fourth International Conference, LATA 2010. LNCS, Springer (2010) 16–31
5. Pereira, F., Warren, D.: Parsing as deduction. In: Proceedings of the 21st annual meeting of the Association for Computational Linguistics, Association for Computational Linguistics (1983) 137–144
6. Yoshinaka, R.: Identification in the limit of k, l-substitutable context-free languages. In: Proceedings of the 9th International colloquium on Grammatical Inference, Springer (2008) 266–279

7. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75**(2) (1987) 87–106
8. Dupont, P., Miclet, L., Vidal, E.: What is the search space of the regular inference? In Carrasco, R.C., Oncina, J., eds.: ICGI. Volume 862 of Lecture Notes in Computer Science., Springer (1994) 25–37
9. Sempere, J., Garcia, P.: A Characterization of Even Linear Languages and its Application to the Learning Problem. In: Proceedings of the Second International Colloquium on Grammatical Inference and Applications, Springer-Verlag (1994) 38–44
10. Clark, A., Eyraud, R.: Polynomial identification in the limit of substitutable context-free languages. Journal of Machine Learning Research **8** (Aug 2007) 1725–1745
11. Clark, A.: PAC-learning unambiguous NTS languages. In: Proceedings of the 8th International Colloquium on Grammatical Inference (ICGI). (2006) 59–71
12. Clark, A.: Distributional learning of some context-free languages with a minimally adequate teacher. In: Proceedings of the ICGI, Valencia, Spain (September 2010)
13. Clark, A.: Learning context free grammars with the syntactic concept lattice. In: Proceedings of the ICGI, Valencia, Spain (September 2010)
14. Clark, A., Eyraud, R., Habrard, A.: A polynomial algorithm for the inference of context free languages. In: Proceedings of the International Colloquium on Grammatical Inference, Springer (September 2008) 29–42
15. Clark, A.: A learnable representation for syntax using residuated lattices. In: Proceedings of the 14th Conference on Formal Grammar, Bordeaux, France (2009)
16. Yoshinaka, R.: Polynomial-time identification of multiple context-free languages from positive data and membership queries. In: Proceedings of the International Colloquium on Grammatical Inference. (2010)
17. Oncina, J., García, P., Vidal, E.: Learning subsequential transducers for pattern recognition interpretation tasks. IEEE Transactions on Pattern Analysis and Machine Intelligence **15** (1993) 448–458
18. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics **6**(4) (2001) 519–535
19. Yoshinaka, R.: Learning mildly context-sensitive languages with multidimensional substitutability from positive data. In Gavaldà, R., Lugosi, G., Zeugmann, T., Zilles, S., eds.: ALT. Volume 5809 of Lecture Notes in Computer Science., Springer (2009) 278–292
20. Seki, H., Matsumura, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theoretical Computer Science **88**(2) (1991) 229
21. Mohri, M.: Finite-state transducers in language and speech processing. Computational Linguistics **23**(2) (1997) 269–311