

Three learnable models for the description of language

Alexander Clark

Department of Computer Science,
Royal Holloway, University of London
Egham TW20 0EX
`alexcl@cs.rhul.ac.uk`

Abstract. Learnability is a vital property of formal grammars: representation classes should be defined in such a way that they are learnable. One way to build learnable representations is by making them objective or empiricist: the structure of the representation should be based on the structure of the language. Rather than defining a function from representation to language we should start by defining a function from the language to the representation: following this strategy gives classes of representations that are easy to learn. We illustrate this approach with three classes, defined in analogy to the lowest three levels of the Chomsky hierarchy. First, we recall the canonical deterministic finite automaton, where the states of the automaton correspond to the right congruence classes of the language. Secondly, we define context free grammars where the non-terminals of the grammar correspond to the syntactic congruence classes, and where the productions are defined by the syntactic monoid; finally we define a residuated lattice structure from the Galois connection between strings and contexts, which we call the syntactic concept lattice, and base a representation on this, which allows us to define a class of languages that includes some non-context free languages, many context-free languages and all regular languages. All three classes are efficiently learnable under suitable learning paradigms.

1 Introduction

“Formal language theory was first developed in the mid 1950’s in an attempt to develop theories of natural language acquisition.”

This statement, in [1], may not be entirely accurate historically, but the point it makes is valid: language theory has its roots in the modelling of learning and of language. The very name of language theory betrays its origins in linguistics. Yet it has moved far from its origins – formal language theory is now an autonomous part of computer science, and only a few papers at the major conferences in Formal Language Theory (FLT) are directly concerned with linguistics. From time to time, the requirements of linguistics impinge on FLT – for example, the discovery of non-context free natural languages [2] inspired much study in

mildly context sensitive grammar formalisms; Minimalist grammars are a direct offshoot of interest in Chomsky's Minimalist Program [3]; and so on.

Learnability is another aspect which has been put to one side. As a model of language acquisition, the original intention was for phrase-structure grammars to be learnable. The PSGs were meant to represent, at a suitable level of abstraction, the linguistic knowledge of the language that the child learner could infer from his or her exposure to the ambient linguistic environment. Chomsky [4] says (p. 172, footnote 15):

The concept of "phrase structure grammar" was explicitly designed to express the richest system that could reasonable be expected to result from the application of Harris-type procedures to a corpus. . . .

The "Harris-type procedures" refer to the methods of distributional learning developed and partially formalised by Zellig Harris [5] following on the ideas of earlier American structuralist linguists. So PSGs in general, and CFGs in particular, were intended, were *designed*, to be learnable by distributional methods — but they weren't. Even the class of regular grammars, equivalent to non-deterministic finite state automata, is not learnable under very easy learning schemes [6]. This is not a problem for distributional methods, but rather a problem with these formalisms. The natural question is therefore whether there are other formalisms, different from the Chomsky hierarchy, that are learnable.

Before we answer this question it is a good idea to be clear about what we mean by learning, and why we think it is so important. Informally learning means that we want to construct our representations for the language from information about the language. From a formal point of view, this means that we wish to define representations, and algorithms for constructing those representations from a source of information about the language, and prove, under a suitable regime, that these algorithms will converge to the right answer. Given a language L , we want to prove that the hypothesised representation will converge to a representation G such that the language defined by G is equal to L . Our focus in this paper is on the computational complexity of learning and so we will assume that we have a very good source of information. We will assume that we have a source of examples of strings from the language, and that additionally the algorithm can query whether a particular string is in the language or not. In the terminology of grammatical inference we have positive data and membership queries. Under some assumptions this is sufficient to learn any class of languages if we neglect computational issues — here we will be considering algorithms that are efficient. For simplicity of exposition we will use a slightly inadequate formalisation: we will merely require that the algorithms use a polynomial amount of computation, in the size of the observed data, at each step of the algorithm.

Of course, not all potential application areas of language theory are linguistic. Linguistics might be considered a special case, where the representations are unknown, as they are cognitive structures and as yet we cannot hope to probe their natures directly. But it is worth considering where the grammars or representations of the languages come from in various other domains. There are broadly speaking two cases: one where we have information about the language,

but not about the representation, and the second is where we have direct information about the representation. In almost all the engineering cases we are interested in, we will have some data that we want to model; in computational biology these might be strings of bases or amino acids, or in other areas they might represent sequences of actions by a robot or a human, or any other sequence of events. But we do not know what the representation is – in none of these cases do we have any direct information about the representation. Indeed, there is no “right” representation, unlike in linguistics. There are merely models that fit the data to a greater or lesser extent.

The only situation where this is not the case is where the language is a programming language or mark up language – in that case, we know what the structure of the language is. In almost all other cases, we will be using languages to represent some sequence of symbols or events, where the representation class is unknown. In these cases, just as in the case of linguistics, learnability is not a tangential property – it is absolutely essential. A representation without some plausible story about how it can be acquired from data is of limited interest. Efficient learning should, in our view, be as important a property of a representation class as efficient parsing.

One way of modelling the data is for a domain expert to construct it by hand. If they are written by a linguist then there are a number of clear desiderata. The grammars should be concise and make it easy for the linguist to express whatever generalisations he or she wishes; it should be easy for humans to reason with consciously, and ideally have a nice graphical representation, perhaps as a tree. Modularity is also a useful property. Following this path we end up with grammatical formalisms being effectively just special-purpose declarative programming languages: DATR [7] and to a lesser extent HPSG [8] are examples of this. There is no reason to think that such formalisms could be or should be learnable. If, on the other hand, we consider the grammar to be the output of a learning process, it need not be particularly convenient for humans to reason about consciously. The properties of a formal grammar are thus radically different, even diametrically opposed depending on whether we envisage them being produced manually or automatically by a process of inference.

2 How

If we accept this argument, then the goal becomes clear – we should construct representations that are intrinsically learnable. Putting this as a slogan: “Put learnability first!” We should design representations from the ground up to be learnable.

Here we present a number of different ways of doing this that follow the same basic strategy: they are objective or “empiricist”. We define the representational primitives of the formalism in a language theoretical way. The basic elements of the formalism, whether they are states in an automaton, or non-terminals in a phrase structure grammar, must have a clear definition in terms of sets of strings, in a way that does not depend on the representation.

Rather than defining a representation, and then defining a function from the representation to the language, we should go backwards. If we are interested first in learnability, then we should start by defining the map from the language to the representation. For example, in a CFG, we define a derivation relation $\xRightarrow{*}$; for each non-terminal N we define the set of strings that can be derived from that non-terminal: $Y(N) = \{w \mid N \xRightarrow{*} w\}$. We then define the language as the set of strings derived from the start symbol or symbols. Thus we define a function from a non-terminal N , to a set of strings $Y(N)$, and thus from the set of context free grammars to the set of context free languages: from G to $L(G)$.

There is however an insurmountable obstacle to going in the reverse direction: $Y(N)$ is almost completely unconstrained. Suppose we have some context free language L , and a grammar G such that $L(G) = L$ and N is a non-terminal in G . What constraints are there on $Y(N)$? We can say literally nothing about this set, other than that it is a context free language. If we restrict the CFG so that all non-terminals are accessible, then it must be a subset of the set of substrings of L , but beyond that we can't say anything. Consider for example the degenerate case when L is Σ^* . There are clearly many CFGs with one non-terminal that define this language, but there are also infinitely many other ones, with non-terminals that correspond to arbitrary context-free subsets of Σ^* . We need some way of identifying some relevant subsets that will correspond to the primitive elements of our representation. We should start by defining some suitable sets of strings; only then can we construct a grammar. If we define some set of strings then the structure of the representation will follow: given an objective definition of the "reference" of the non-terminal or symbol, the derivation process will be fixed.

We will define three representations; we will start with a basic representation that is quite standard, and turns out to be equivalent to a subclass of DFAs; indeed to the canonical automata. In this case we base the representation on the right congruence classes, as used in the Myhill-Nerode theorem. The next step is to define context free grammars where the non-terminals correspond to the syntactic congruence classes of the language; the third and final representation uses what we call the syntactic concepts of the language – elements of a residuated lattice – as the representational primitives, and the resulting formalism can represent some non-context-free languages.

3 Canonical DFA

We will start by considering a classic case [9–11], where the learnability problems are well understood: the case of regular languages. We will end up with a class of representations equivalent to a subclass of deterministic finite automata. We present this standard theory in a slightly non-standard way in order to make the step to the next class of representations as painless as possible.

We will define our notation as needed; we take a finite non-empty alphabet Σ , and the free monoid Σ^* ; we use λ to refer to the empty string. A language is a subset of Σ^* . Let L be some arbitrary language; not necessarily regular,

or even computable. We define the residual language of a given string u as $u^{-1}L = \{w : uw \in L\}$.

Consider the following relation between strings: $u \sim_L v$ iff $u^{-1}L = v^{-1}L$. This is an equivalence relation and additionally a right congruence: if $u \sim_L v$ then for all $w \in \Sigma^*$, $uw \sim_L vw$.

We can consider the equivalence classes under this relation: we will write $[u]^R$ for the congruence class of the string u under this right congruence. It is better to consider these classes not just as sets of strings but as pairs $\langle P, S \rangle$ where P is a congruence class, and S is the residual language of all the strings in P . That is to say we will have elements of the form $\langle [u]^R, u^{-1}L \rangle$. One important such element is $\langle [\lambda]^R, L \rangle$.

Suppose we define a representation that is based on these congruence classes. Let us call these primitive elements of our representation *states*. The state $\langle [\lambda]^R, L \rangle$ we will denote by q_0 . A few elementary observations: if $u \in L$ then every element of $[u]^R$ is also in L . We will call a state $\langle P, S \rangle$ such that $\lambda \in S$ a *final* state. Thus if we can tell for each string in the language which congruence class it is in, then we will have predicted the language. Our representation will be based on this idea: we will try to compute for each string w not just whether it is in L but which congruence class it is in.

We have now taken the first step: we have defined the primitive elements of the representation. We now need to define a derivation of some sort by exploiting the algebraic structure of these classes, in particular the fact that they are right congruence classes. Since it is a right congruence, if we have a string that we know is in the congruence class $[u]^R$ and we append the string v we know that it will be in the class $[uv]^R$. Thus we have a “transition” from the state $[u]^R$ to the state $[uv]^R$ labelled with the string v . It is clear that we can restrict ourselves to the case where $|v| = 1$, i.e. where the transitions are labelled with letters. We now have something that looks very like an automaton. We let Q be the, possibly infinite set of all these states, q_0 the initial state, δ a transition function defined by $\delta([u]^R, a) = [ua]^R$, and let F be the set of final states $\{[u]^R | u \in L\}$. We now define $\mathfrak{R}(L)$ to be this, possibly infinite, representation. We have defined a function from L to $\mathfrak{R}(L)$.

We extend δ recursively in the standard way and then define the function from the representation to the language $L(\mathfrak{R}(L)) = \{w : \delta(q_0, w) \in F\}$. Given this, we have that for any language L , $L(\mathfrak{R}(L)) = L$. In a strict sense, this “representation” is correct. Of course we are interested in those cases where we have a finite representation, and $\mathfrak{R}(L)$ will be finite if and only if L is regular, by the Myhill-Nerode theorem. Thus while we have that for any language this representation is correct, we can only finitely represent the class of regular languages.

It is possible to infer these representations for regular languages, using a number of different techniques depending on the details of the source of information that one has about the language. The recipe is then as follows:

- Define a set of primitive elements language theoretically – in this case the right congruence classes: $[u]^R$.

- Identify a derivation relation of some sort based on the algebraic structure of these elements: $[u]^R \rightarrow^a [v]^R$.
- Construct a representation $\mathfrak{R}(L)$ based on the language and prove that it is correct $L(\mathfrak{R}(L)) = L$.
- Identify the class of languages that can be finitely represented in this way: the class of regular languages. Happily, in this case, it coincides with an existing class of languages.

4 CFGs with congruence classes

We can now move on from representations that are regular to ones that are capable of representing context-free languages. We do this using the idea of distributional learning. These techniques were originally described by structuralist linguists who used them to devise mechanical procedures for discovering the structure of natural languages. As such, they are a reasonable starting point for investigations of learnable representations.

Some notation and basic terminology: we define a context to be a pair of strings (l, r) where $l, r \in \Sigma^*$. We combine a context with a substring so $(l, r) \odot u = lur$; We will sometimes refer to a context (l, r) with a single letter f . A string u occurs in a context (l, r) in a language if $lur \in L$. If L, R are sets of strings then we use (L, R) , (L, r) etc to refer to the obvious sets of contexts: $L \times R$, $L \times \{r\}$ and so on. We define the distribution of a string in a language as the set of all contexts that it can occur in: $C_L(w) = \{(l, r) | lur \in L\}$. We will extend the notation \odot to contexts: $(l, r) \odot (x, y) = (lx, yr)$, so $(f \odot g) \odot u = f \odot (g \odot u)$. We will also use it for sets in the obvious way.

We now define the crucial notion for this approach: two strings u and v are syntactically congruent iff they have the same distribution: $u \equiv_L v$, iff $C_L(u) = C_L(v)$. We write $[u]$ for the congruence class of u . We note here a classic result – the number of congruence classes is finite if and only if the language is regular.

Given the discussion in the previous section we hope that it is obvious what the next step is: our primitive elements will correspond to these congruence classes. Immediately this seems to raise the problem that we will be restricted to regular languages, since we are interested in finite representations, and thus can only represent a finite number of congruence classes. This turns out not to be the case, as we shall see.

Clearly the empty context (λ, λ) has a special significance: this context is in the distribution of a string if and only if that string is in the language; $(\lambda, \lambda) \in C_L(u)$ means that $u \in L$. So if we can predict the congruence class of a string, we will know whether the string is in the language or not. Given this fixed interpretation of these symbols, we can now proceed to determine what the appropriate derivation rules should be. We have mentioned that this is a congruence: this means that for all strings u, v, x, y if $u \equiv_L v$ then $xuy \equiv_L xvy$. This means that if we take any element of $[u]$ say u' and any element of $[v]$ say v' , and concatenate them, then the result $u'v'$ will always be in the same congruence class as $[uv]$. This means that if we want to generate an element of

$[uv]$ we can do this by generating an element from $[u]$ and then generating an element from $[v]$ and then concatenating the results. In other words, we have a context free production $[uv] \rightarrow [u][v]$. Additionally we know that for any string w we can generate an element of $[w]$ just by producing the string w . Given the previous productions, it is clearly sufficient just to have these productions for strings of length 1: i.e. to have productions $[a] \rightarrow a$.

Another way of viewing this is to note that the concatenation of the congruence classes is well defined – or alternatively that since the relation \equiv_L is a monoid congruence, we can use the quotient monoid Σ^* / \equiv_L which is the well known syntactic monoid. The production rules then can be viewed as saying that $X \rightarrow YZ$ is a production if and only if $X = Y \circ Z$, and that $X \rightarrow a$ iff $a \in X$. Here, we can see even more clearly that the structure of the representation is based on the structure of the language – in this case we have a CFG-like formalism that corresponds exactly to the syntactic monoid of the language. We therefore define our representation as follows: $\mathfrak{C}(L)$ consists of the, possibly infinite, set of congruence classes $[u]$ together with a set of productions consisting of $\{[uv] \rightarrow [u], [v]|u, v \in \Sigma^*\}$ and $\{[a] \rightarrow a|a \in \Sigma\}$ and $[\lambda] \rightarrow \lambda$. We identify a set of initial symbols $I = \{[u]|u \in L\}$, and we define derivation exactly as in a context free grammar.

It is then easy to see that given these productions, $[w] \xrightarrow{*} v$ iff $v \in [w]$. Thus the productions are well behaved. We then define $L(\mathfrak{C}(L)) = \{w|\exists N \in I \text{ such that } N \xrightarrow{*} w\}$. We can then prove that $L(\mathfrak{C}(L)) = L$, for any language L .

We have used the two schemas $[uv] \rightarrow [u][v]$ and $[a] \rightarrow a$; these are sufficient. But it is conceivable that we might want to have different schemas – we can have schemas like $[w] \rightarrow w$, $[lwr] \rightarrow l[w]r$, $[aw] \rightarrow a[w]$ or even $[uvw] \rightarrow [u][v][w]$, which will give us finite grammars, linear grammars, regular grammars and so on. All of these schemas maintain the basic invariant that they will only derive strings of the same congruence class.

We thus end up with something that looks something like a context free grammar in Chomsky normal form. It differs in two respects, one trivial and one extremely important. The trivial difference is that we may have more than one start symbol: we wish to maintain the nice map between the representation and the structure.

The second point is that the number of congruence classes will be infinite, if the language is not regular. Consider the language $L_{ab} = \{a^n b^n | n \geq 0\}$. This is a non-regular context free language. It is easy to see that we have an infinite number of congruence classes since a^i is not congruent to a^j unless $i = j$. It appears therefore that we have only achieved another representation for regular languages. We can consider this as the *canonical context free grammar* for a regular language.

Let us suppose we maintain the structure of the representation but only take a finite set of congruence classes V consisting of the classes corresponding to a finite set of strings K : $V = \{[u]|u \in K\}$. We will assume that K contains Σ and λ and finitely many others strings. The binary productions will thus be limited to the finite set $\{[uv] \rightarrow [u][v]| [u], [v], [uv] \in V\}$. This will then give us a finite

representation, which we denote $\mathfrak{C}(L, K)$. We can prove that if we have only a subset of the productions, then $[w] \xrightarrow{*} v$ implies $v \in [w]$, and therefore our representation will always generate a subset of the correct language: $L(\mathfrak{C}(L, K)) \subseteq L$.

The class that we can represent is therefore the set of all languages L such that there is some finite set of strings K such that $\mathfrak{C}(L, K)$ defines the correct language.

$$\mathcal{L}_{\text{CCFG}} = \{L \mid \exists \text{ finite } K \subset \Sigma^* \text{ such that } L(\mathfrak{C}(L, K)) = L\}$$

This class clearly includes the class of regular languages. It also includes some non-regular context free languages. In the case of our example L_{ab} it is sufficient to have the following congruence classes: $[a]$, $[b]$, $[\lambda]$, $[ab]$, $[aab]$, $[abb]$. Not all context free languages can be described in this way. The context free language $\{a^n b^m \mid n < m\}$ is not in $\mathcal{L}_{\text{CCFG}}$, as the language is the union of an infinite number of congruence classes. $\mathcal{L}_{\text{CCFG}}$ is therefore a proper subclass of the class of context free languages; by restricting the non-terminals to correspond exactly to the congruence classes, we lose a bit of representational power, but we gain efficient learnability. Note the very close relationship to the class of NTS languages [12]; indeed we conjecture that these classes may be equal. The first results on learning using this approach [13–15] have shown that various subclasses can be learned under various non-probabilistic and probabilistic paradigms. Note that we have lost the canonical nature of the formalism – there will often be more than one possible minimal choice of K or V . Nonetheless, given that the definition of the primitives is fixed this is not a problem: any sufficiently large set of congruence classes will suffice to define the same language.

4.1 Regular languages

Let us briefly return to the case of regular languages. We know that the set of congruence classes is finite, but we can get some insight into the structure of this set by looking at the proof. Let A be the minimal deterministic finite automaton for a language L . Let Q be the set of states of this automaton; let $n = |Q|$. A string w defines a function from Q to Q : $f_w(q) = \delta(q, w)$. Clearly there are only n^n possible such functions. But if $f_u = f_v$ then $u \equiv_L v$, and so there can be at most n^n possible congruence classes. Indeed Holzer and König [16] show that we can approach this bound. This reveals two things: one that using one non-terminal per congruence class could be an expensive mistake as the number might be prohibitively large, and secondly, that there is often some non-trivial structure to the monoid.

Since these congruence classes correspond to functions from Q to Q it seems reasonable to represent them using some basis functions. Consider the set of partial functions that take $q_i \rightarrow q_j$: there are only n^2 of these. Each congruence class can be represented as a collection of at most n of these that define the image under f of each $q \in Q$.

If we represent each congruence class as a n by n boolean matrix T ; where T_{ij} is 1 iff $f_u : q_i \mapsto q_j$, then the basis functions are the n^2 matrices that have

just a single 1; and we represent each congruence class as a sum of n such basis functions.

Suppose the language is reversible [17]: i.e. $uv, u'v, uv' \in L$ implies $u'v' \in L$. Then for each state q_i we can define a pair of strings l_i, r_i that uniquely pick out that state: in the sense that $\delta(q_0, l_i) = q_i$ and only q_i has the property that $\delta(q_i, r_i)$ is a final state.

Thus we can represent the basis functions using the finite set of contexts (l_i, r_j) that represents the transition function $q_i \rightarrow q_j$. This gives us an important clue how to represent the syntactic monoid: If we have a class that maps $q_i \rightarrow q_j$ and another which maps $q_j \rightarrow q_k$ then the concatenation will map $q_i \rightarrow q_k$. Thus rather than having a very large number of very specific rules that show how individual congruence classes combine, we can have a very much smaller set of more general rules which should be easier to learn. This requires a representation that contains elements that correspond not just to individual congruence classes but to sets of congruence classes.

5 Distributional lattice grammars

The final class of representations that we consider are based on a richer algebraic structure; see [18] for a more detailed exposition. Note that the congruence classes correspond to sets of strings and dually to sets of contexts: a congruence class $[u]$ also defines the distribution $C_L(u)$ and vice versa. It is natural to consider therefore as our primitive elements certain ordered pairs which we write $\langle S, C \rangle$ where S is a subset of Σ^* and C is a subset of $\Sigma^* \times \Sigma^*$. Given a language L we will consider only those pairs that satisfy two conditions: first that $C \odot S$ is a subset of L , and secondly that both of these sets are maximal, while respecting the first condition. If a pair satisfies these conditions, then we call it a *syntactic concept* of the language.

We have chosen to define it in this way to bring out the connection to the Universal automaton [19, 20], which has the same construction but using a prefix-suffix relation, rather than the context substrings relation.

Another way is to consider the Galois connection between the sets of strings and contexts, which give rise to exactly the same sets of concepts. For a given language L we can define two polar maps from sets of strings to sets of contexts and vice versa. Given a set of strings S we can define a set of contexts S' to be the set of contexts that appear with every element of S .

$$S' = \{(l, r) : \forall w \in S \ lwr \in L\} \quad (1)$$

Dually we can define for a set of contexts C the set of strings C' that occur with all of the elements of C

$$C' = \{w : \forall (l, r) \in C \ lwr \in L\} \quad (2)$$

A concept is then an ordered pair $\langle S, C \rangle$ where $S' = C$ and $C' = S$. The most important point here is that these are closure operations in the sense that

$S''' = S'$ and $C''' = C'$. This means that we can construct a concept by taking any set of strings S and computing $\langle S'', S' \rangle$, and similarly by taking any set of contexts C and computing $\langle C', C'' \rangle$. We will write $\mathcal{C}(S)$ for $\langle S'', S' \rangle$. This set of concepts have an interesting and rich algebraic structure, which gives rise to a powerful representation.

We will start by stating some basic properties of the set of concepts. The first point is that there is an inverse relation between the size of the set of strings S and the set of contexts C : the larger that S is the smaller that C is: in the limit there is always a concept where $S = \Sigma^*$; normally this will have $C = \emptyset$. Conversely we will always have an element $\mathcal{C}(\Sigma^* \times \Sigma^*)$. One particularly important concept is $\mathcal{C}(L) = \mathcal{C}((\lambda, \lambda))$: the language itself is one of the concepts.

The most basic structure that this set of concepts has is therefore as a partially ordered set. We can define a partial order on these concepts where:

$$\langle S_1, C_1 \rangle \leq \langle S_2, C_2 \rangle \text{ iff } S_1 \subseteq S_2.$$

$S_1 \subseteq S_2$ iff $C_1 \supseteq C_2$. We can see that $\mathcal{C}(L) = \mathcal{C}(\{(\lambda, \lambda)\})$, and clearly $w \in L$ iff $\mathcal{C}(\{w\}) \leq \mathcal{C}(\{(\lambda, \lambda)\})$.

Given this partial order we can see easily that in fact this forms a complete lattice; which we write $\mathfrak{B}(L)$, called the *syntactic concept lattice*. Here the topmost element is $\top = \mathcal{C}(\Sigma^*)$ bottom is written $\perp = \mathcal{C}(\Sigma^* \times \Sigma^*)$, and the two meet and join operations are defined as $\langle S_x, C_x \rangle \wedge \langle S_y, C_y \rangle$ is defined as $\langle S_x \cap S_y, (S_x \cap S_y)' \rangle$ and \vee dually as $\langle (C_x \cap C_y)', C_x \cap C_y \rangle$.

Figure 1 shows the syntactic concept lattice for the regular language $L = \{(ab)^*\}$. L is infinite, but the lattice $\mathfrak{B}(L)$ is finite and has only 7 concepts.

Monoid structure

Crucially, this lattice structure also has a monoid structure. We can define a binary operation over concepts using the two sets of strings of the concepts: define $\langle S_1, C_1 \rangle \circ \langle S_2, C_2 \rangle = \mathcal{C}(S_1 S_2)$. Note that this operation then forms a monoid, as it is both associative and has a unit $\mathcal{C}(\lambda)$. There is also an interaction between this monoid structure and the lattice structure. It is clearly monotonic in the sense that if $X \leq Y$ then $X \circ Z \leq Y \circ Z$ and so on, but there is a stronger relation. We can define two residual operations as follows:

Definition 1. Suppose $X = \langle S_x, C_x \rangle$ and $Y = \langle S_y, C_y \rangle$ are concepts. Then define the residual $X/Y = \mathcal{C}(C_x \odot (\lambda, S_y))$ and $Y \setminus X = \mathcal{C}(C_x \odot (S_y, \lambda))$

These satisfy the following conditions: $Y \leq X \setminus Z$ iff $X \circ Y \leq Z$ iff $X \leq Z/Y$. That is to say, given an element Z and an element X , $X \setminus Z$ is the largest element which when concatenated to the right of X will give you something that is less than Z .

With these operations the syntactic concept lattice becomes a residuated lattice. This gives some intriguing links to the theory of categorial grammars [21].

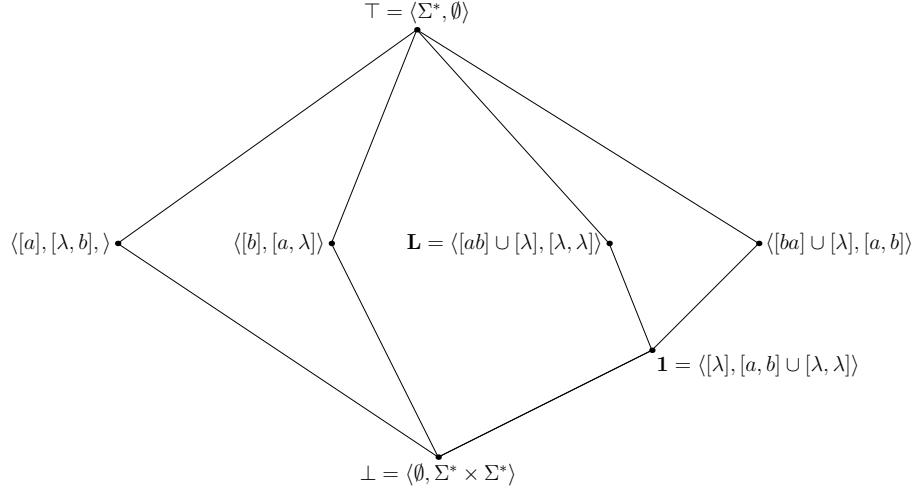


Fig. 1. The Hasse diagram for the syntactic concept lattice for the regular language $L = \{(ab)^*\}$. Each concept (node in the diagram) is an ordered pair of a set of strings, and a set of contexts. We write $[u]$ for the equivalence class of the string u , $[l, r]$ for the equivalence class of the context (l, r) , under equality of

5.1 Maximal elements

One important point of the residual operation is that we can use them to extract maximally general concatenation rules. So, suppose we have some concept Z . We can consider the set of all pairs of concepts (X, Y) such that their concatenation is less than Z .

$$H(Z) = \{(X, Y) \mid X \circ Y \leq Z\} \quad (3)$$

This is clearly a down set, in that if $(X, Y) \in H(Z)$ and $X' \leq X$ and $Y' \leq Y$ then $(X', Y') \in H(Z)$, and so it is natural to consider the maximal elements of this set. We can find these elements using the residuation operations. If $(X, Y) \in H(Z)$ then also we must have $(X, X \setminus Z)$ and $(Z/Y, Y)$ in $H(Z)$. But we can repeat this: we will also have $(Z/(X \setminus Z), X \setminus Z)$ and $(Z/Y, (Z/Y) \setminus Z)$ in $H(Z)$. We can prove that repeating the process further is not necessary – indeed all maximal elements of $H(Z)$ will be of this form.

An example: suppose $L = \{a^n b^n \mid n \geq 0\}$. Consider $H(\mathcal{C}(L))$. Clearly $\mathcal{C}(a) \circ \mathcal{C}(b) = \mathcal{C}(L)$, so $(\mathcal{C}(a), \mathcal{C}(b)) \in H(\mathcal{C}(L))$. Let us identify the two maximal elements that we get by generalising this pair. $\mathcal{C}(L)/\mathcal{C}(b) = \mathcal{C}(aab)$, and $\mathcal{C}(a)\mathcal{C}(L) = \mathcal{C}(abb)$. Repeating the process does not increase these so the two maximal elements above this pair are $(\mathcal{C}(a), \mathcal{C}(abb))$ and $(\mathcal{C}(aab), \mathcal{C}(b))$.

6 Representation

Having defined and examined the syntactic concept lattice, we can now define a representation based on this. Again, since the lattice will be infinite if the language is not regular, we need to consider just a part of it. We will start by considering how we might define a representation given the whole lattice. Given a string w we want to compute whether it is in the language or not. Considering this slightly more generally we want to be able to compute for every string w , the concept of w , $\mathcal{C}(w)$. If $\mathcal{C}(w) \leq \mathcal{C}(L)$, then we know that the string is in the language. If we have the whole lattice then it is quite easy: since $\mathcal{C}(u) \circ \mathcal{C}(v) = \mathcal{C}(uv)$, we can simply take the list of letters that form w and concatenate their concepts. So if $w = a_1 \dots a_n$, then $\mathcal{C}(w) = \mathcal{C}(a_1) \circ \dots \circ \mathcal{C}(a_n)$. It is enough to know the concepts of the letters, and of course of λ and the algebraic operation \circ which is associative and well-behaved. In this case it effectively reduces to computing the syntactic monoid as before. The idea is very simple – we compute a representation of the distribution of a string, by taking the distributions of its parts and combining them.

However, if we have a non-regular language, then we will need to restrict the lattice in some way. We can do this by taking a finite set of contexts $F \subseteq \Sigma^* \times \Sigma^*$, which will include the special context (λ, λ) and constructing a lattice using only these contexts and all strings Σ^* . This give us a finite lattice $\mathfrak{B}(L, F)$, which will have at most $2^{|F|}$ elements. We can think of F as being a set of *features*, where a string w has the feature (context) (l, r) iff $lwr \in L$.

Definition 2. For a language L and a set of context $F \subseteq \Sigma^* \times \Sigma^*$, the partial lattice $\mathfrak{B}(L, F)$ is the lattice of concepts $\langle S, C \rangle$ where $C \subseteq F$, and where $C = S' \cap F$, and $S = C'$.

We can define a concatenation operation as before

$$\langle S_1, C_1 \rangle \circ \langle S_2, C_2 \rangle = \langle ((S_1 S_2)' \cap F)', (S_1 S_2)' \cap F \rangle$$

This is now however no longer a residuated lattice as the \circ operation is no longer associative, there may not be an identity element, nor are the residuation operations well defined. We will now clearly not be able to compute things exactly for every language, but we should still be able to approximate the computation, and for some languages, and for some sets of features the approximation will be accurate.

We note some basic facts about this partial lattice: first it is no longer the case that $\mathcal{C}(u) \circ \mathcal{C}(v) = \mathcal{C}(uv)$. If we take a very long string in the language, we may be able to split it into two parts neither of which have any contexts in F . For example, if we have the language of our running example $a^n b^n$, and a small set of short contexts, we could take the string $a^{100} b^{100}$ and split it into the two strings a^{100} and b^{100} . Neither of these two strings will have any contexts in F , and so $\mathcal{C}(a^{100}) = \mathcal{C}(b^{100}) = \top$; this means that $\mathcal{C}(a^{100}) \circ \mathcal{C}(b^{100}) = \top \circ \top = \top > \mathcal{C}(a^{100} b^{100}) = \mathcal{C}(L)$: they are not equal.

However we can prove that $\mathcal{C}(u) \circ \mathcal{C}(v) \geq \mathcal{C}(uv)$. This means that given some string, w , we can compute an upper bound on the $\mathcal{C}(w)$ quite easily: we will call this upper bound $\phi(w)$. This may not give us exactly the right answer but it will still be useful. If the upper bound is below $\mathcal{C}(L)$ then we definitely know that the string will be in the language: if $\mathcal{C}(w) \leq \phi(w)$ and $\phi(w) \leq \mathcal{C}(L)$, then $\mathcal{C}(w) \leq \mathcal{C}(L)$.

In fact we can compute many different upper bounds: since the operation is no longer associative, the order in which we do the computations matters. Suppose we have some string ab ; our upper bound can just be $\mathcal{C}(a) \circ \mathcal{C}(b)$. But suppose we have a string of length 3: abb . Our upper bound could be $\mathcal{C}(a) \circ (\mathcal{C}(b) \circ \mathcal{C}(b))$ or $(\mathcal{C}(a) \circ \mathcal{C}(b)) \circ \mathcal{C}(b)$. We can now use the lattice structure to help: if $\mathcal{C}(abb) \leq X$ and $\mathcal{C}(abb) \leq Y$ then $\mathcal{C}(abb) \leq X \wedge Y$, since $X \wedge Y$ is a greatest lower bound. So in this case we can have our upper bound as $(\mathcal{C}(a) \circ (\mathcal{C}(b) \circ \mathcal{C}(b))) \wedge ((\mathcal{C}(a) \circ \mathcal{C}(b)) \circ \mathcal{C}(b))$. For longer strings, we have a problem: we cannot compute every possible binary tree and then take the meet over them all, since the number of such trees is exponential in the length.

However we can compute an even tighter bound using a recursive definition that can be computed by an efficient dynamic programming algorithm in $\mathcal{O}(|w|^3)$. For each substring of the word, we compute the lowest possible upper bound and then recursively combine them.

Given a language L and set of contexts F :

Definition 3. we define $\phi : \Sigma^* \rightarrow \mathfrak{B}(L, F)$ recursively by

- $\phi(\lambda) = \mathcal{C}(\lambda)$
- $\phi(a) = \mathcal{C}(a)$ for all $a \in \Sigma$, (i.e. for all w , $|w| = 1$)
- for all w with $|w| > 1$,

$$\phi(w) = \bigwedge_{u,v \in \Sigma^+ : uv=w} \phi(u) \circ \phi(v) \quad (4)$$

We can define the language generated by this representation to be:

$$\hat{L} = L(\mathfrak{B}(L, F)) = \{w | \phi(w) \leq \mathcal{C}((\lambda, \lambda))\} \quad (5)$$

We can show using a simple proof by induction that the computation ϕ will always produce an upper bound.

Lemma 1. For any language L and set of contexts F , for any string w , $\phi(w) \geq \mathcal{C}(w)$

This has the immediate consequence that:

Lemma 2. For any language L and for any set of contexts F , $L(\mathfrak{B}(L, F)) \subseteq L$.

As we increase the set of contexts, we will find that the language defined increase monotonically, until in the infinite limit when $F = \Sigma^* \times \Sigma^*$ we have that $L(\mathfrak{B}(L, \Sigma^* \times \Sigma^*)) = L$. This means that the problem of finding a suitable set of contexts is tractable and we can also define a natural class of languages

as those which have representations as finite lattices. We will call this class of representations the Distributional Lattice Grammars.

The class of languages definable by finite DLGS is denoted \mathcal{L}_{DLG} .

$$\mathcal{L}_{\text{DLG}} = \{L : \exists \text{ a finite set } F \subset \Sigma^* \times \Sigma^* \text{ such that } L(\mathfrak{B}(L, F)) = L\} \quad (6)$$

First we note that \mathcal{L}_{DLG} properly includes $\mathcal{L}_{\text{CCFG}}$. Indeed \mathcal{L}_{DLG} includes some non-context free languages, including ones that are related to the MIX language [22]. \mathcal{L}_{DLG} is a proper subclass of the languages defined by Conjunctive Grammars [23]. \mathcal{L}_{DLG} also includes a much larger set of context free languages than $\mathcal{L}_{\text{CCFG}}$ including some non-deterministic and inherently ambiguous languages.

A problem is that the lattices can be exponentially large. We can however represent them lazily using a limited set of examples, and only compute the concepts in the lattice as they are needed. This allows for efficient learning algorithms. An important future direction for research is to exploit the algebraic structure of the lattice to find more compact representations for these lattices, using maximal elements.

7 Discussion

The importance of reducing the under-determination of the grammar given the language has been noted before: de la Higuera and Fernau [24] argue that learnable representations must have a canonical form, and that equivalence should be computable.

On a historical note, it is important not to neglect the contribution of the Kulagina school, which was initiated by the seminal paper of [25]. The first work on the distributional lattice was by Sestier [26], and subsequent work was developed by Kunze [27]. However the concatenation and residuation properties seem not to have been noted. Important papers that follow this line of research include [28, 29] as well as [30] and [31]. The ideas of basing representations on the congruence classes can be found in these works, but for some reason the rule schema $[uv] \rightarrow [u][v]$ seems not to have been discovered, instead exploration focussed on the linear grammar schema $[lur] \rightarrow l[u]r$, and on the development of contextual grammars [32]. We suspect that there are other related works in the Eastern European literature that we have not yet discovered.

Many other approaches can be recast in this form. For example the context-deterministic languages of [33], are exactly context free languages where the non-terminals have the property that they correspond to concepts, and where additionally the distributions of the non-terminals are disjoint: $\mathcal{C}(M) \vee \mathcal{C}(N) = \top$ for distinct non-terminals N and M .

There are a number of directions for future research that this approach suggests: probabilistic extensions of these algorithms, the development of algorithms for equivalence and decidability, and the use of these approaches for modelling transductions are all natural developments. Another extension would be to consider representations based on the relation $(x, y) \sim (u, v, w)$ iff $uxvyw \in L$. Some steps in this direction have recently been taken by Yoshinaka [34], which lead

naturally to Multiple Context-Free Grammars. There are numerous language theoretic problems that need to be explored in the use of these models.

Finally, representations such as these, which are both efficiently learnable and capable of representing mildly context-sensitive languages seem to be good candidates for models of human linguistic competence.

Acknowledgements

I am very grateful to Rémi Eyraud and Amaury Habrard.

References

1. Harrison, M.A.: Introduction to Formal Language Theory. Addison Wesley (1978)
2. Shieber, S.: Evidence against the context-freeness of natural language. *Linguistics and Philosophy* **8** (1985) 333–343
3. Chomsky, N.: The Minimalist Program. MIT Press (1995)
4. Chomsky, N.: Language and mind. 3rd edn. Cambridge Univ Pr (2006)
5. Harris, Z.: Distributional structure. In Fodor, J.A., Katz, J.J., eds.: *The Structure of Language*. Prentice-Hall (1954) 33–49
6. Angluin, D., Kharitonov, M.: When won't membership queries help? *J. Comput. Syst. Sci.* **50** (1995) 336–355
7. Evans, R., Gazdar, G.: DATR: A language for lexical knowledge representation. *Computational Linguistics* **22**(2) (1996) 167–216
8. Pollard, C., Sag, I.: *Head Driven Phrase Structure Grammar*. University of Chicago Press (1994)
9. Gold, E.M.: Complexity of automaton identification from given data. *Information and Control* **37**(3) (1978) 302 – 320
10. Carrasco, R.C., Oncina, J.: Learning deterministic regular grammars from stochastic samples in polynomial time. *Theoretical Informatics and Applications* **33**(1) (1999) 1–20
11. Clark, A., Thollard, F.: PAC-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research* **5** (May 2004) 473–497
12. Sénizergues, G.: The equivalence and inclusion problems for NTS languages. *J. Comput. Syst. Sci.* **31**(3) (1985) 303–331
13. Clark, A., Eyraud, R.: Polynomial identification in the limit of substitutable context-free languages. *Journal of Machine Learning Research* **8** (Aug 2007) 1725–1745
14. Clark, A.: PAC-learning unambiguous NTS languages. In: *Proceedings of the 8th International Colloquium on Grammatical Inference (ICGI)*. (2006) 59–71
15. Clark, A., Eyraud, R., Habrard, A.: A polynomial algorithm for the inference of context free languages. In: *Proceedings of International Colloquium on Grammatical Inference*, Springer (September 2008) 29–42
16. Holzer, M., König, B.: On deterministic finite automata and syntactic monoid size. In: *Proc. Developments in Language Theory 2002*. (2002)
17. Angluin, D.: Inference of reversible languages. *Communications of the ACM* **29** (1982) 741–765
18. Clark, A.: A learnable representation for syntax using residuated lattices. In: *Proceedings of the 14th Conference on Formal Grammar, Bordeaux, France* (2009)

19. Conway, J.: Regular algebra and finite machines. Chapman and Hall, London (1971)
20. Lombardy, S., Sakarovitch, J.: The universal automaton. In Grädel, E., J., F., Wilke, T., eds.: Logic and Automata: History and Perspectives. Amsterdam Univ Pr (2008) 457–494
21. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly **65**(3) (1958) 154–170
22. Boullier, P.: Chinese Numbers, MIX, Scrambling, and Range Concatenation Grammars. Proceedings of the 9th Conference of the European Chapter of the Association for Computational Linguistics (EACL 99) (1999) 8–12
23. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics **6**(4) (2001) 519–535
24. Fernau, H., de la Higuera, C.: Grammar induction: An invitation for formal language theorists. Grammars **7** (2004) 45–55
25. Kulagina, O.S.: One method of defining grammatical concepts on the basis of set theory. Problemy Kiberneticy **1** (1958) 203–214 (in Russian).
26. Sestier, A.: Contribution à une théorie ensembliste des classifications linguistiques. In: Premier Congrès de l'Association Française de Calcul, Grenoble (1960) 293–305
27. Kunze, J.: Versuch eines objektivierten Grammatikmodells I, II. Z. Zeitschrift Phonetik Sprachwiss. Kommunikat **20-21** (1967–1968)
28. Kříž, B.: Generalized grammatical categories in the sense of Kunze. Archivum Mathematicum **17**(3) (1981) 151–158
29. Drášil, M.: A grammatical inference for C -finite languages. Archivum Mathematicum **25**(2) (1989) 163–173
30. Novotny, M.: On some constructions of grammars for linear languages. International Journal of Computer Mathematics **17**(1) (1985) 65–77
31. Martinek, P.: On a Construction of Context-free Grammars. Fundamenta Informaticae **44**(3) (2000) 245–264
32. Păun, G.: Marcus contextual grammars. Kluwer Academic Pub (1997)
33. Shirakawa, H., Yokomori, T.: Polynomial-time MAT Learning of C -Deterministic Context-free Grammars. Transactions of the information processing society of Japan **34** (1993) 380–390
34. 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