Logical Grammars, Logical Theories

Alexander Clark

Department of Computer Science
Royal Holloway, University of London
Egham, TW20 0EX
United Kingdom
alexc@cs.rhul.ac.uk

Abstract. Residuated lattices form one of the theoretical backbones of the Lambek Calculus as the standard free models. They also appear in grammatical inference as the syntactic concept lattice, an algebraic structure canonically defined for every language L based on the lattice of all distributionally definable subsets of strings. Recent results show that it is possible to build representations, such as context-free grammars, based on these lattices, and that these representations will be efficiently learnable using distributional learning. In this paper we discuss the use of these syntactic concept lattices as models of Lambek grammars, and use the tools of algebraic logic to try to link the proof theoretic ideas of the Lambek calculus with the more algebraic approach taken in grammatical inference. We can then reconceive grammars of various types as equational theories of the syntactic concept lattice of the language. We then extend this naturally from models based on concatenation of strings, to ones based on concatenations of discontinuous strings, which takes us from context-free formalisms to mildly context sensitive formalisms (multiple context-free grammars) and Morrill's displacement calculus.

1 Introduction

Logic is concerned with proof; in the logical grammar tradition the role of proof is central and well understood [1]. But logic is also concerned with truth – and what does it mean for a grammar, such as a Lambek grammar, to be true? What can it be true of? This is a problem not of proof theory but of model theory, and while there has been a great deal of work on models for the Lambek calculus [2], these are models for the calculus, not for the grammars. These models are free – the only statements that are true in these models are things that are true of all languages – but this seems to be inappropriate – there are things that are true of some languages and not of others; true about English but not about French or Dyirbal.

A Lambek grammar consists only of a finite set of type assignments: (w, T) where w is a nonempty sequence, typically of length 1, and T is a type. Therefore the question of the truth of the grammar reduces to the question of the truth of the type assignments. The metatheory of Lambek grammar is explained in [3] and [4]. Several things are clear: first the types in the grammar are intended to refer to sets of strings in a given language, and the composite types $A \bullet B$, C/B, $A\backslash C$ and the associated calculus are motivated and justified by the algebraic properties of the associated operations on these sets. The following quotes from [4] lay out the metatheoretical assumptions. First

> Sets of strings of English words will be called *(syntactic) types*.

Secondly, that the operations are 'free':

> When applying this notation to a natural language ... we are thinking of the ... system ... freely generated by the words of the language under concatenation.

Finally, that the starting point for constructing a grammar is to model an existing language. Lambek does not conceive of this as a learning problem, but as a modeling problem in the structuralist tradition. Rather than defining a grammar G *in vacuo* and then showing how that grammar defines a language L — the generative stance — he starts with a language L_*, and constructs a grammar $G(L_*)$, — the descriptive approach — using a non-algorithmic method:

> One can continue playing this game until every English word has been assigned a finite number of types.

One hopes that the grammar defined in this way will define a language which is equal to the original language L_*, a condition Lambek states as follows:

> One may consider the categorial grammar to be *adequate* provided it assigns type S to w if and only if the latter is a well-formed declarative sentence according to some other standard.

Lambek's specific proposals within this metatheoretical program include what are now called Lambek grammars, and later, pregroup grammars. Others have studied the relationship between types and sets of strings; specifically Buszkowski in [5], and the possibilities of developing the informal process of construction of a grammar into an algorithmically well-defined learning procedure [6, 7].

In this paper we will take a different tack, motivated again by a focus on learnability. We are interested in identifying formalisms that are not only descriptively adequate but that also can be learned from the data in a principled way – in Chomskyan terms, in achieving explanatory adequacy. Defining the types as sets of strings seems a reasonable first step. We can in a similar vein consider nonterminal symbols in a context-free grammar to be sets of strings: namely the set of strings that can be derived from that nonterminal. For a CFG G, with a nonterminal N we define as usual $\mathcal{L}(G, N) = \{w \mid N \overset{*}{\Rightarrow}_G w\}$. A symbol NP which represents noun phrases can be considered as the set of strings that can be noun phrases; the start symbol S corresponds to the language L itself.

In Lambek grammars and other categorial grammars, we have the same correspondence, but it is more complex. We can associate with each type, primitive or complex, a set of strings: say $v(T)$, the *value* of that type. There is a complication because the types are no longer atomic elements but have some structure. So a type might be of the form A/B: in this case we can ask, what is the relationship between $v(A/B)$ and $v(A)$ and $v(B)$? In particular what is the relationship between $v(A/B)$ and $v(A)/v(B)$? Clearly this depends on how we define the function v.

More fundamentally, if the primitive types are sets of strings, then what sets of strings are they? We now have a good understanding of how learnability affects the answer to this question, at least in the context of CFGs. Learnable representations are almost invariably based on objective or empiricist representations where the structure of the representation is based on the structure of the data: the language itself. The research program is then to identify structures of various type in the data, and then define representations that exploit algebraic properties of this structure. The classical example of this is the canonical deterministic finite automaton being based on the Myhill-Nerode congruence, which gives rise to the theory of regular grammatical inference [8–10]. When learning CFGs, the sets of strings corresponding to each nonterminal must correspond to distributionally definable sets. In the most basic models we define them to be congruence classes [11] or the sets of strings that can occur in a given context [12]. The syntactic concept lattice (SCL) of a language [13] is defined by looking at the relation between substrings and contexts in languages. This has now given rise to several different learnability results and formalisms [14–16]. The SCL is a residuated lattice. These objects are well known as models of various types of substructural logics [17], and as one of the foundations of the Lambek calculus. Thus a natural question arises as to the precise nature of the relationship between Lambek grammar and grammars of various types based on the SCL. It might be possible, for example, to construct efficient learning algorithms for some subclasses of categorial grammars — [7] showed that some interesting classes are learnable, though [18] showed that there are computational problems with this approach. Conversely, it might be possible to adapt the elegant syntax-semantics interface of categorial and type-logical grammars to these learnable models.

There is an additional connection between Lambek grammars and distributional learning. Learning is in some sense the inverse of generation: learning is an inverse problem. The slash operators $/, \backslash$ are the left and right residuals (a type of inverse) of concatenation and thus have a crucial role in the inference process; they also correspond directly to distributional properties.

More precisely, if we have a set of strings X and a language L, the set L/X which we define as $\{u | \forall v \in X, uv \in L\}$ is exactly the set of strings that has a certain distributional property. Namely, using notation which we present later, it is the set of all strings that can occur in all of the contexts $\{\Box v | v \in X\}$. This is also something noted in [3] where Lambek explicitly defines the motivating operations in distributional terms, as we shall see later.

This paper examines the relationship between the structure of this lattice, and the structure of models of Lambek grammar. Our main claim is that we can better meet the metatheoretical objectives of Lambek's program by basing the grammars we use on the SCL, and that more generally we can look at grammars as being sets of equations that describe algebraic structures associated with the language.

2 Residuated lattices

We will start with some standard definitions. A monoid is a simple associative algebraic structure, $\langle M, \circ, 1\rangle$ where M is a set, $\circ$ is an associative binary operation and 1 is a left and right unit. The standard example is the set of all strings over a finite alphabet Σ under concatenation; which we write Σ^*, with the empty string denoted by λ. This is the free monoid generated by Σ. In what follows we will assume we have some fixed nonempty set Σ. A (formal) language is a subset of Σ^*.

A context is just an ordered pair of strings that we write $l\Box r$. l and r refer to left and right; $\Box$ is a symbol not in Σ. [1] We can combine a context $l\Box r$ with a string u with a wrapping operation that we write $\odot$: so $l\Box r \odot u$ is defined to be lur. We will sometimes write f for a context $l\Box r$. The empty context we write $\Box$. Clearly $\Box \odot x = x$.

We will fix a formal language $L \subseteq \Sigma^*$; this then defines a relation between contexts $l\Box r$ and strings w given by $l\Box r \sim_L w$ iff $lwr \in L$. The distribution of a string w is defined as $C_L(w) = \{l\Box r | lwr \in L\}$. We can define a congruence relation based on distributional equivalence with respect to a language L. We say $u \equiv_L v$ iff $C_L(u) = C_L(v)$. This is an equivalence relation and a congruence; we write $[u]_L = \{v \mid u \equiv_L v\}$. This gives us a second relevant example of a monoid: the *syntactic monoid* which is the quotient monoid $\Sigma^* / \equiv_L$, whose elements are the congruence classes and where the natural concatenation operation, $[u]_L \circ [v]_L = [uv]_L$, is well defined as this is a congruence. This is not free: this has a nontrivial algebraic structure that depends on the language; it can be a group, it can be Abelian, it can be finite or infinite. Indeed this is finite iff the language L is regular.

The second class of structures we are interested in are lattices; which are partially ordered sets with least upper bounds (join, written $\vee$) and greatest lower bounds (meet, written $\wedge$). The classic example is the lattice of all subsets of a given set, where meet is set intersection and join is set union. Given the meet operation we can define the partial order as $X \leq Y$ iff $X = X \wedge Y$.

A residuated lattice is a combination of these two structures — a monoid and a lattice — where the two structures must interact 'nicely'. Formally we say that it is a tuple $\langle M, \circ, 1, \wedge, \vee, /, \backslash\rangle$, where M is a set, $\circ, \wedge, \vee, /, \backslash$ are binary operations and 1 is a constant, such that $\langle M, \circ, 1\rangle$ is a monoid, and $\langle M, \wedge, \vee\rangle$ is a lattice.

[1] In previous work we have written a context as an ordered pair (l, r). This causes confusion when we consider discontinuous strings, and so here we use this less ambiguous notation.

We can state the interaction requirement as the fact that the following three conditions are equivalent:

$$X \circ Y \leq Z \text{ iff } X \leq Z/Y \text{ iff } Y \leq X\backslash Z$$

This is a slightly stronger requirement than mere monotonicity. The classic example of this is the set of all languages over Σ^*, which we denote by 2^{Σ^*}, and where the lattice operations are set intersection and union, and where $\circ$ is defined to be simple concatenation $M \cdot N = \{uv|u \in M, v \in N\}$ and the residuals are defined naturally to be

$$M/N = \{u|\forall v \in N, uv \in M\}$$
$$N\backslash M = \{u|\forall v \in N, vu \in M\}$$

It is easy to verify that these definitions satisfy the axioms of residuated lattices. Note that of course $\circ$ is not commutative. This lattice is distributive, but in general these lattices are not. We will use for the rest of the paper this 'result on top' notation, for consistency with the literature in residuated lattices.

Given the definitions it is easy to see that $(X/Y) \circ Y \leq Y$. Similarly it is easy to establish the following identities, and many others, which will be very familiar to those who know the Lambek calculus.

Composition $(X/Y) \circ (Y/Z) \leq (X/Z)$ and $(Z\backslash Y) \circ (Y\backslash Z) \leq (Z\backslash X)$
Associativity $(Z\backslash X)/Y = Z\backslash(X/Y)$
Lifting $X \leq Y/(X\backslash Y)$ and $X \leq (Y/X)\backslash Y$.

However there are also identities that have no counterparts in the Lambek calculus. These are ones that use the operations $\wedge$ and $\vee$ such as for example the fact that $X \wedge Y \leq X$. These are however valid in the full Lambek calculus, augmented with appropriate inference rules for these connectives.

2.1 Syntactic Concept Lattice

Given a particular language, L, we have a particular monoid, the syntactic monoid of that language. In just the same way, there is a residuated lattice associated with the language that we call the syntactic concept lattice. This is the lattice of all distributionally definable subsets of strings in a language; we will denote this $\mathfrak{B}(L)$.

There are a number of equivalent ways of defining this; we will use a way which emphasizes the links with Galois connections and residuated maps. As noted above, we can consider the relation between strings and contexts defined by a language. Given this relation we can then define two maps from sets of strings to sets of contexts and vice-versa. These are traditionally called 'polar' maps and we will denote them both by the use of $'$. Intuitively, given a set of strings S, we can define the set of contexts S' to be the set of contexts that appear with every element of S. This is sometimes called the *distribution* of the set S in the language L.

$$S' = \{l\Box r \mid \forall w \in S \;\; lwr \in L\} \tag{1}$$

If S consists of a single string w, then S' is just the same as $C_L(w)$. If S contains two strings, then it will be the intersection of the distribution of those two strings. Clearly if $S \subseteq T$ then $S' \supseteq T'$ – as we make the set of strings larger, the intersection of their distributions will only decrease.

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

$$C' = \{w \mid \forall\, l\Box r \in C \;\; lwr \in L\} \tag{2}$$

Consider now for any set of strings S the set of strings S''. It is easy to verify that $S'' \supseteq S$, and furthermore that $(S'')'' = S''$ – in other words, this is a closure operation. We say that the set of strings S is *closed* iff $S = S''$.

Note that for any set of contexts C, C' is a closed set of strings. We can thus define a closed set of strings in two ways: either by picking a set of strings S and closing it to get S'' or by picking a set of contexts C which defines the set C'. We call the former method a *primal* method, and the latter a *dual* method.

It is easy to see that L itself is a closed set of strings. L' will contain the empty context $\Box$. As a result any string in L'' must occur in the context $\Box$ and is thus in L; therefore $L = L''$. Similarly Σ^* is always closed.

We can consider, for a language L, the set of all closed sets of strings of the language. Interestingly, this set is finite if and only if the language is regular. For example, the language $L = \Sigma^*$ only has one closed set of strings, Σ^*. The infinite regular language $L = (ab)^*$ has 7 closed sets of strings. These consist of the four sets $\emptyset, \{\lambda\}, L, \Sigma^*$ together with 3 other sets which are bL, La and bLa.

These closed sets form a lattice – the partial order is given by set inclusion and meet and join are defined as $S \wedge T = S \cap T$ and $S \vee T = (S \cup T)''$. We define a concatenation operation $\circ$ of two closed sets of strings as $S \circ T = (ST)''$; ST is not necessarily a closed set, and so to make the operation well defined we take the closure. Note the difference between this definition and 'normal' set concatenation $S \cdot T$. Given the language $(ab)^*$, if we concatenate the two closed sets of strings La and bL, we get the language $LabL = (ab)^+$. This is not closed: $((ab)^+)'' = (ab)^* = L$. So $La \circ bL = L$ which is not equal to $LabL$.

While the concatenation is different from concatenation in the lattice 2^{Σ^*}, there are two residual operations which are identical. Thus if S and T are closed then S/T and $S\backslash T$ are also closed. We can verify that the closed sets of strings with respect to a language L form a residuated lattice, which we denote $\mathfrak{B}(L)$. See [13] for further details [2]. Note that the natural map from $2^{\Sigma^*} \rightarrow \mathfrak{B}(L)$, given by $h_L(S) = S''$ is a homomorphism of $\circ, \vee$ and 1.

[2] In that paper we present the lattice as a collection of ordered pairs of sets of strings and sets of contexts. In this paper we will consider the mathematically equivalent formulation just as sets of strings.

3 Types should be closed sets of strings

Let us now start to consider the relationship between the SCL, $\mathfrak{B}(L)$, and the system of types that is used by Lambek grammars. This can be framed in a number of different ways, mathematically, conceptually and empirically.

The empirical question is whether, in natural languages, the types correspond to closed sets of strings. Lambek clearly thought so: indeed we find in [3] the following statement:

> We shall assign type n to all expressions which can occur in any context in which all proper names can occur.

Let us look at this statement carefully. Suppose we have a language L and a set of strings N that is the set of proper names. The set of contexts in which all proper names can occur is just the set N' using the polar maps defined earlier. The set of expressions which can occur in any of N' is just N''. So the type n corresponds exactly to the closed set of strings N''; since the start symbol S corresponds to the language L, then all product free types will correspond to closed sets of strings.

The second way of looking at this is as a mathematical claim: is it the case that for all Lambek grammars, the types correspond to closed sets of strings? Here we can give a clear negative answer: it is easy to construct Lambek grammars where the types correspond to quite arbitrary sets of strings. For example consider the language $L = \{a^n b^n c \mid n > 0\}$. The closure of $\{ab\}$ is $\{a^n b^n \mid n > 0\}$, but one could write a grammar where there is a type which generates any finite or infinite context-free subset of that closed set.

Conceptually, we can make additional arguments in favour of this representational assumption. Given that we have exactly this structure why would we impose another structure on it? Using a Lambek grammar implicitly imposes a residuated lattice structure on the language [5]. On grounds of general parsimony, in the absence of a convincing reason to use a different structure, we should use a grammar which follows the existing objectively valid empirically based structure, rather than using some other more arbitrary one: a convincing reason would be if the restriction affected the weak or strong generative capacity of the formalism. We now consider this possibility.

3.1 Weak generative power

Lambek grammars can represent any context-free language. However if we add the restriction that the types must correspond to closed sets of strings, then it is natural to ask whether we lose any weak generative power. The somewhat surprising answer is that we do not.

Suppose we have a context-free language defined by a CFG, G, which we assume is in Greibach normal form. For each nonterminal N we define a set of strings which is the closure of the set of strings in the yield of N: $\{w | N \overset{*}{\Rightarrow}_G w\}''$. And for each terminal symbol a we define a type A which has value $v(A) = \{a\}''$.

For an element of $(V \cup \Sigma)^+$ we extend the type function v using $\circ$, in the obvious way: $v(a\beta) = v(a) \circ v(\beta)$ and $v(N\beta) = v(N) \circ v(\beta)$. We can then prove that if $N \to \alpha$ is a production then $v(N) \supseteq v(\alpha)$ [15]. We can therefore construct a categorial grammar in the standard way since if the rule is $N \to aM_1 \dots M_k$ then $a \in v(N/M_k/\dots/M_1)$, and so we can assign the type $N/M_k/\dots/M_1$ to a. Any derivation of a string with respect to the CFG can be turned into a valid derivation with respect to the Lambek grammar, and, conversely, any string generated by the Lambek grammar will be in the language by the soundness of the Lambek calculus.

3.2 Strong generative capacity

In contrast, from the point of view of strong generative capacity, that is to say in terms of the sets of structural descriptions that our grammar assigns to strings, we do give up some modeling power. In particular we lose the ability to describe distributionally identical strings in different ways. More precisely, we can propose the following principle: If for two letters $a, b \in \Sigma$, the distribution of a is a subset of the distribution of b, then for any strings l, r the set of structural descriptions assigned to lar must be a subset of the set of structural descriptions assigned to lbr, modulo some relabeling to reflect the difference between a and b.

If, for example, two letters are distributionally identical, then in this model, they *must* be treated identically. In a normal Lambek grammar, or context-free grammar, we are free to treat them as being completely different. Consider the following example: a language L which has two different grammars which are weakly equivalent but assign different structural descriptions to the strings. To be concrete, suppose we have $(ab)^+$ which admits the two CFGs, G_1 which has productions $S_1 \to ab$, $S_1 \to abS$ and G_2 which has the productions $S_2 \to ab$ $S_2 \to aTb$, $T \to ba$, $T \to bS_2a$. Consider the language $(c|d)(ab)^+$. We could have a grammar then with two rules $S \to cS_1$ and $S \to dS_2$. This will assign different trees to strings starting with c and strings starting with d even though c and d are in this case completely distributionally identical. This sort of situation is ruled out by this model – if natural languages had this sort of behaviour then the type of distributionally motivated model that we propose here would be clearly incorrect. If on the other hand there is this type of connection between the distributional properties and the structural properties of the language, then this would indicate that we are on the right track.

3.3 Consequences

Let us now proceed on the assumption that we want the types in our grammar to correspond to closed sets of strings. We fix a target language L_*. We start by leaving the Lambek grammar formalism completely unchanged, and making the representational assumption that the types are closed sets of strings. We assume a finite set of primitive types $\mathbf{Pr}$, which we extend to a set of product free types $\mathbf{Tp}(/, \backslash)$ using just the two connectives $/, \backslash$. We assume that we have a function v, a valuation, from these types to closed sets of strings, $v : \mathbf{Tp}(/, \backslash) \to \mathfrak{B}(L_*)$, which is defined on the primitive types and then extended recursively to the complex types using $v(R/T) = v(R)/v(T)$ and $v(R\backslash T) = v(R)\backslash v(T)$. We assume that one primitive type S is the start type such that $v(S) = L_*$.

Since the SCL is a residuated lattice, the Lambek calculus is sound. That is to say, if we have a sequent $\Gamma \to X$ that is derivable in the Lambek calculus then $v(\Gamma) \subseteq v(X)$ where we extend v to $\mathbf{Tp}^+$ using $\circ$. We then make a finite lexicon which is a finite assignment of types to elements of Σ, and we require that it satisfy the obvious compatibility condition that if a has type T then $a \in v(T)$. This is the condition that is called *correctness* in [5].

This gives us a Lambek grammar. Note that since the Lambek calculus is sound, and because of the compatibility condition the type assignment satisfies we have the following trivial lemma, whose proof we omit.

Lemma 1. $L(G) \subseteq L_*$.

Example 1. Suppose $L_* = \{a^n b^n | n > 0\}$. The closed sets of strings include $\{a\}, \{b\}$. We assign type A to $\{a\}$, B to $\{b\}$ and S to L. We then have the infinite set of derived types A/A, S/B etc. each of which refer to closed sets of strings. $v(A/A) = \{\lambda\}$, $v(S/B) = \{a^{i+1}b^i | i \geq 0\}$, $v((S/B)/S) = \{a\}$, and so on. Since $a \in v(S/B)$ and in $v((S/B)/S)$ we assign a the two types S/B and $(S/B)/S$ and to b we assign the type B. This gives us a Lambek grammar. Clearly $ab \in L(G)$ since $S/B, B$ is a valid type assignment and we can prove $S/B, B \to S$. Similarly $aabb \in L(G)$ through the type assignment $(S/B)/S, S/B, B, B$. Indeed $L(G) = L_*$.

This grammar is adequate in Lambek's sense – it defines the correct language. Of course, we needed to manually choose which primitive types we should use. In this case we only needed to use the 'natural' types – namely the closures of the singleton sets of elements of Σ, $A = \{a\}''$ and $B = \{b\}''$, and the special type S. These particularly simple types have a fixed valuation in a given language.

Contrast this to the approach taken by Buszkowski in [5]. There we start with a fixed grammar. For a primitive type T, $v(T)$ is defined to be the set of strings such that there is a valid type assignment so that we can derive T from that type assignment. Given this we then extend it to the set of all product free types. We can then ask whether the type assignment (a, T) satisfies $a \in v(T)$. Here we already have a semantics for the types before we start defining the grammar, and so it is easy to stipulate this correspondence as a condition.

4 Finite Representations of Closed Sets

In the preceding discussion we simply stipulated what the relevant closed sets of strings were. This glosses over an important problem – how to identify or refer to a closed set of strings. In the earlier example we started with some primitive types which referred to the sets $\{a\}''$ which are just the closed sets of strings defined by a single element of Σ, and the language itself, L, which is always a closed set. To avoid confusion, we will use a notational convention from mathematical logic and use $\dot{a}$ for the type (symbol in the metalanguage) to distinguish it from the symbol $a \in \Sigma$ in the object language. We will use only these 'natural' types as our primitive types: one type $\dot{a}$ for each element a of Σ and one special type $\dot{L}$, which refers to L, together with the extra type $\dot{\lambda}$, which denotes the closure of the empty string. In what follows we will eliminate all other primitive types, as they are in a sense arbitrary, and rely only on these more basic ones which have a well defined semantics.[3] We will also augment our set of connectives to include not just the two connectives $/, \backslash$, but also a product $\bullet$, and the two lattice operations $\vee, \wedge$. Our set of types $\mathbf{Tp}_\Sigma$ is thus the closure of these primitive types under the five binary connectives.

Given a language L, we recursively define the value of all types using a valuation v_L, a function from $\mathbf{Tp}_\Sigma \to \mathfrak{B}(L)$, which is defined in Table 1.

Table 1. Interpretation of the basic types and connectives.

Primitive types	Lattice connectives	Monoid connectives
$v_L(\dot{a}) = \{a\}''$	$v_L(R \vee T) = (v_L(R) \cup v_L(T))''$	$v_L(R \bullet T) = (v_L(R) \cdot v_L(T))''$
$v_L(\dot{L}) = L$	$v_L(R \wedge T) = v_L(R) \cap v_L(T)$	$v_L(R/T) = v_L(R)/v_L(T)$
$v_L(\dot{\lambda}) = \{\lambda\}''$		$v_L(R\backslash T) = v_L(R)\backslash v_L(T)$

Note that for any term T, $v_L(T)$ is a closed set of strings, an element of $\mathfrak{B}(L)$.

We are interested in ways that we can use some simple finite expression to define, or refer to, a closed set of strings in a language. There are two different strategies we can use to define closed sets of strings. The first, which we call a *primal* approach, involves specifying a finite set of strings S; this will define the closed set S''. Alternatively, we can pick a finite set of contexts C and use this to define closed set C'. Using these two approaches we can express both the primal and dual approaches using just complex types formed from these 'natural' types. Suppose we have a single string of length n, $w = a_1 \dots a_n$. We will write $\dot{w}$ as an abbreviation for the complex type $\dot{w} = \dot{a_1} \bullet (\dot{a_2} \dots \dot{a_n})$. It is easy to see that $v(\dot{w}) = \{w\}''$, and so we can represent the set of strings $\{w\}''$ using types of the form $\dot{w}$. Similarly suppose we have a set of strings $X = \{u, v\}$. We can refer to the closed set of strings X'' using the type $\dot{X} = \dot{u} \vee \dot{v}$. Again it is easy to see that $v(\dot{X}) = X''$. Similarly for any finite set of strings X we define the associated term $\dot{X}$. There are of course many different terms, by associativity of concatenation and commutativity and associativity of union, which all denote the same closed set of strings – we assume that we fix some particular one according to some arbitrary convention.

In the dual situation we will have a finite set of contexts, C which defines a set of strings C'. Suppose $C = \{l \Box r\}$; we define $\dot{C}$ to be the complex type $(\dot{l} \backslash \dot{L})/\dot{r}$. Suppose $u \in C'$. This means that $lur \in L$. Suppose $l_2 \in \{l\}''$ and

[3] We do not consider here $\top$ and $\bot$ as they seem not to be necessary for syntactic description.

$r_2 \in \{r\}''$ then $l_2 u r_2 \in L$ so $u \in v(\dot{C})$. Indeed $v(\dot{C}) = C'$. If C has more than one element, then we will define a term using $\wedge$: If $C = \{l_1 \Box r_1, \dots, l_k \Box r_k\}$, we define $\dot{C} = ((\dot{l_1} \backslash \dot{L})/\dot{r_1}) \wedge \dots ((\dot{l_k} \backslash \dot{L})/\dot{r_k})$. Again it is straightforward to verify that $v(\dot{C}) = C'$.

Therefore in order to define closed sets of strings using either the primal or dual methods, we need only use the basic natural types that refer to the individual elements of Σ and the language itself. There is no need to use any additional arbitrary primitive types. This comes at a price – we cannot refer to all closed sets of strings in all languages using only finite terms of this form. This restricts the class of languages we can define in this way in a nontrivial way. Contrast this case with that of congruential languages [11], where the restriction to congruence classes causes some limitation in descriptive power, but it is easy to refer to each congruence class (simply by referring to any string in the class). Here, we lose no descriptive power, but cannot in some cases refer to the relevant classes of strings using only a finite description.

5 Grammars as sets of equations

Having eliminated all of the arbitrary symbols, we are left only with symbols that have a clear denotation in our language. We can therefore define statements using these symbols to say things about the language we are modeling that may be either true or false.

We have a set of types or terms, using the primitive symbols and the binary operations, that we call $\mathbf{Tp}_\Sigma$. Each term refers to a particular closed set of strings in a language using the valuation v_L.

We then add one relation symbol, equality, $\doteq$, again using the dot convention to distinguish this as a symbol in the metalanguage. We will consider equations of the form $R \doteq T$ for $R, T \in \mathbf{Tp}_\Sigma$.

Definition 1. *An equation $R \doteq T$ is true in the language L, which we write $L \models R \doteq T$, iff $v_L(R) = v_L(T)$*

Note that we can state inequalities using only the symbol $\doteq$ since $(T \wedge R) \doteq R$ is true iff $v_L(T) \subseteq v_L(R)$. We will use $S \,\dot{\subseteq}\, T$ as a shorthand for such an equation.

Every equality or inequality of this form is either true or false of a particular language. The statement $w \in L$ becomes the equation $\dot{w} \,\dot{\subseteq}\, \dot{L}$, in the sense that $L \models \dot{w} \,\dot{\subseteq}\, \dot{L}$ iff $w \in L$. The statement $u \equiv_L v$ becomes $\dot{u} \doteq \dot{v}$. Clearly there are variety of statements that can be framed in this way, from ones which, for CFGs are polynomially decidable such as the former, to those like the latter, which are undecidable for CFGs. The lexicon in a Lambek grammar consists of type assignments (a, T) where $a \in \Sigma$ and $T \in \mathbf{Tp}_\Sigma$ which in this formalism can be written as equations of the form $\dot{a} \,\dot{\subseteq}\, T$. We do not restrict ourselves only to this sort of equation but consider for the moment any kind of equation.

We can therefore view a grammar as a set of equations that are true of the language, or more strictly true of the syntactic concept lattice of the language. More formally, let $\mathcal{E}$ be a finite set of equation; we say that $L \models \mathcal{E}$ iff $L \models E$

for all $E \in \mathcal{E}$. If this is the case then we say that L is a model for $\mathcal{E}$. Note that there will always be at least one model: the language $L = \Sigma^*$ has a trivial SCL with only one element, and so it satisfies all sets of equations.

Example 2. Consider the grammar of Example 1. We have two nontrivial assignments of the word a: $\dot{L}/B$ and $(\dot{L}/B)/\dot{L}$. These two assignments correspond to the inequalities $\dot{a} \dot{\subseteq} \dot{L}/\dot{b}$ and $\dot{a} \dot{\subseteq} \dot{L}/\dot{b}/\dot{L}$. These correspond to the statements $\{a\}''\{b\}'' \subseteq L$ and $\{a\}''L\{b\}'' \subseteq L$, both of which are true for the language in question.

We can define a semantic notion of entailment: If every language L that satisfies all of the equations in $\mathcal{E}$ also satisfies another equation E then $\mathcal{E} \models E$. In other words, all models of $\mathcal{E}$ are also models of $\mathcal{E} \cup \{E\}$. We could define a language on the basis of this semantic entailment: $\{w \in \Sigma^* \mid \mathcal{E} \models \dot{w} \dot{\subseteq} \dot{L}\}$, but it is more convenient, we think, to use a syntactic notion of entailment using a syntactic calculus. We can code the axioms of the residuated lattices using equations and functions of various arities. We code the primitive types $\dot{a}, \dot{\lambda}, \dot{L}$ as constant symbols, and have a countable set of variables $x, y, z, \ldots$. For example associativity of the monoid is defined as: $x \bullet (y \bullet z) \doteq (x \bullet y) \bullet z$, using implicit universal quantification as in equational logic. We assume a finite equational basis for residuated lattices that we write **RL** and we write $\mathcal{E} \vdash E$ for the syntactic notion of entailment under Birkhoff's equational deductive system using $\mathcal{E}$ and **RL**. This gives us a slightly different definition that we will now use.

Definition 2. *A finite set of equations $\mathcal{E}$ defines a language over Σ, $\mathcal{L}(\mathcal{E}) = \{w \in \Sigma^* \mid \mathcal{E} \vdash \dot{w} \dot{\subseteq} \dot{L}\}$*

This does not entirely coincide with the definition of semantic entailment, in spite of Birkhoff's completeness theorem. This is because the notion of semantic entailment restricts the class of models to lattices that are the SCLs of languages, whereas the equations that we use to define residuated lattices allow all residuated lattices.

Many different formalisms can be cast in this light: they all have weak generative power properly included in the class of conjunctive grammars [19]; a context sensitive formalism that properly includes the CFGs. The use of terms with $\wedge$ takes us out of the class of CFLs [20].

An advantage of framing things in this way is that grammar construction may become a computationally tractable problem, because it can be decomposed into a collection of smaller problems. Each equation is either true or false with respect to the language being modeled. If a grammar overgenerates then this is because at least one equation in the grammar is false. The validity of each equation can be evaluated and tested independently of all others. This means that the learning algorithm only has to deal with a set of independent, local problems rather than one single global problem. As a result for these formalisms we can in general devise computationally efficient learning procedures [21, 22, 11]. However, given the undecidability of this logic in general, we will need to restrict the sorts of equations we use in order to exploit this property efficiently.

5.1 Lambek grammars as equations

We will start with Lambek grammars. We assume that all of the primitive types in the grammar can be defined using equations, as in Section 4, and that the grammar is adequate. If we assign the type T to a word a in a Lambek grammar, then this is equivalent to the inequality $\dot{a}\dot{\subseteq}T$. We assume that all of these type assignments are true. Indeed if we assign to a word a the two types R, T then this is equivalent to the single type assignment, $\dot{a} \dot{\subseteq} R \wedge T$ since if $a \in v(T)$ and $a \in v(R)$ then $a \in v(R) \cap v(T) = v(R \wedge T)$[20]. We can therefore code the lexicon of a Lambek grammar as a set of equations $\mathcal{E}$, one per element of Σ. We can then show that the language defined by the Lambek grammar is exactly equal to $\mathcal{L}(\mathcal{E})$. The two sets of derivations though are not entirely the same for reasons we will explain in the context of CFGs in the next section.

5.2 Context-free grammars

We can also frame context free grammars as a system of equations of this type. Suppose we have some CFG G in, for simplicity, Chomsky normal form. For a nonterminal N we can define the corresponding set of strings as the closure of the set of yields $\{w | N \overset{*}{\Rightarrow}_G w\}''$. We assume that for each nonterminal in G, the corresponding closed set of strings can be expressed as a finite term, which we will write as $\dot{N}$. So $v_L(\dot{N}) = \mathcal{L}(G, N)''$. Clearly S can be represented as the term $\dot{L}$. Then we can convert each production of the form $N \rightarrow PQ$ to an equation which states that the right hand side is a subset of the left hand side: $\dot{P} \bullet \dot{Q} \dot{\subseteq} \dot{N}$. Each production of the form $N \rightarrow a$ is an equation of the form $\dot{a} \dot{\subseteq} \dot{N}$, and an epsilon-production is of the form $\dot{\lambda} \dot{\subseteq} \dot{L}$. Therefore we can define for each production in the grammar an equation, giving a set of equations $\mathcal{E}_G$ such that $S \overset{*}{\Rightarrow}_G w$ iff $\mathcal{E}_G \vdash \dot{w} \dot{\subseteq} \dot{L}$

Note that the derivation process in the grammar is not exactly the same as the proof with respect to the set of equations. For example, if a nonterminal N is represented by a dual term, say $\dot{l} \backslash \dot{L} / \dot{r}$, and we have a production $N \rightarrow a$, then from $\dot{a} \dot{\subseteq} \dot{l} \backslash \dot{L} / \dot{r}$ we can deduce directly using the residuation operations that $\dot{l} \bullet \dot{a} \bullet \dot{r} \dot{\subseteq} \dot{L}$ i.e. that lar is in the language, even if, for example N is not reachable in the grammar. This does not affect the set of strings that are generated, since the equations are all true, and thus we will not overgenerate. Similarly, when we consider nonterminals that are defined primally, N might correspond to the closure of the finite set $\{w_1, \ldots, w_k\}$. In this case we will have a direct proof of $\dot{w}_i \dot{\subseteq} \dot{N}$, which might not correspond to a CFG derivation. Thus it might be desirable to restrict the class of derivations available to exclude these ones which though legal, fail to correspond to proper CFG derivations.

In both the case of CFGs and Lambek grammars, we cannot represent all context free languages in this way, since there are languages where the closed sets that we require cannot be defined using only finite terms of the types we consider here.

5.3 Finite automata

We can however represent all regular languages using finite automata in this manner, since residual languages are closed sets. Recall that the residual languages are defined as $u^{-1}L = \{v | uv \in L\}$, for some $u \in \Sigma^*$. Consider a deterministic finite automaton with state set Q, initial state q_0, transition function δ and set of final states $F \subseteq Q$. For each state in $q \in Q$, assuming they are all reachable, we can find a string w_q such that $\delta(q_0, w_q) = q$; we stipulate that $w_{q_0} = \lambda$.

We can therefore represent a state q by the term $\dot{w}_q \backslash \dot{L}$; under this scheme the initial state is represented by $\dot{\lambda} \backslash \dot{L}$ which will have the same value as $\dot{L}$. If there is a transition from state q to state r labelled with a then this can be represented as the equation: $\dot{w}_q \backslash \dot{L} \dot{\supseteq} \dot{a} \bullet (\dot{w}_r \backslash \dot{L})$. If a state q is in F this is represented by the equation $\dot{w}_q \dot{\subseteq} \dot{L}$ or equivalently $\dot{\lambda} \dot{\subseteq} \dot{w_{q_f}} \backslash \dot{L}$. It is easy to verify that if a word w is accepted by the automaton then, if we denote $\delta(q_0, w) = q_f$, then $\mathcal{E} \vdash \dot{L} \dot{\supseteq} \dot{w} \bullet (\dot{w_{q_f}} \backslash \dot{L})$, and therefore, given that $q_f \in F$ that $\mathcal{E} \vdash \dot{w} \dot{\subseteq} \dot{L}$ iff the automaton accepts w.

5.4 Thue systems

We can get some more insight by considering a particularly limited form of equation. In order to say that a string is in the language, we can use an equation of the form $\dot{w} \dot{\subseteq} \dot{L}$. Clearly if we only have equations of this type, we will not have anything other than a simple finite list of elements of the language. Let us add a second type of rule: equations of the form $\dot{u} \doteq \dot{v}$. This states that $\{u\}'' = \{v\}''$ which implies that $u \equiv_{L_*} v$. A finite set of equations of this type is exactly equivalent to a Thue system: the only connective we use is $\circ$ and the only relation is equality. Recall that a Thue system T is just a finite set of pairs of strings, that we will write as $u \leftrightarrow v$. For example $\{ab \leftrightarrow \lambda\}$ is a simple Thue system. This defines a congruence of Σ^*, which we write $\equiv_T$, as the symmetric transitive closure of $lur \equiv_T lvr$ if $u \leftrightarrow v$ is in T. This is clearly a congruence which is called the Thue congruence defined by T; one of the simplest and earliest form of rewriting systems. This is essentially the same as a finite presentation of a finitely generated monoid: we have a finite set Σ of generators, and a finite set of equations or relations over Σ. The combination of these two types of rules gives us a simple way of defining languages: for example the Dyck language can be defined using the two equations $\dot{a}\dot{b} \doteq \dot{\lambda}$ and $\dot{\lambda} \dot{\subseteq} \dot{L}$.

Note that the congruence defined by the system will not in general be exactly the same as the syntactic congruence of the language defined by the system. If the system of equations defines the language L then clearly if $\mathcal{E} \vdash \dot{u} \doteq v$ then $u \equiv_L v$, but not necessarily in reverse.

Example 3. Consider $\mathcal{E}$ to consist of $\dot{a}\dot{b}\dot{a}\dot{b} \doteq \dot{a}\dot{b}$ and $\dot{a}\dot{b} \dot{\subseteq} \dot{L}$. This defines the language $(ab)^+$ which is regular. This has a syntactic monoid where $aa \equiv_L bb$. But it is not the case that $\mathcal{E} \vdash \dot{a}\dot{a} \doteq \dot{b}\dot{b}$.

5.5 Distributional Lattice Grammars

Distributional lattice grammars (DLGs) are a learnable grammatical formalism explicitly based on a model of the SCL [23]; it is therefore straightforward to convert them into a set of equations that describe it. For reasons of space we will not present them in full; one of their advantages is that they can compactly represent an exponential number of equations using only a polynomial amount of data. The DLG models a partial lattice that considers only a finite set of contexts We assume that we have a set of k contexts, $F = \{l_1\Box r_1, \ldots, l_k\Box r_k\}$, which include the empty context $\Box$. We denote this partial lattice by $\mathfrak{B}(L,F)$; which consist only of the finite collection of closed sets of strings defined by subsets of F, together with a concatenation operation defined by

$$M \circ N = ((MN)' \cap F)'$$

We can finitely represent this partial lattice using the set of contexts, a sufficiently large set of substrings K, and some amount of information about L. In particular we need to know which elements of $F \odot KK$ are in L. There is a map f^* from $\mathfrak{B}(L,F) \to \mathfrak{B}(L)$, which embeds the finite collection of closed sets defined by subsets of F into the full lattice, which satisfies

$$f^*(M \wedge N) = f^*(M) \wedge f^*(N) \tag{3}$$
$$f^*(M \circ N) \supseteq f^*(M) \circ f^*(N) \tag{4}$$

These equations therefore justify the following conversion of relations in the finite lattice into statements about the SCL. We use subsets of F to define closed sets of strings; if $C \subseteq F$ is a set of contexts, we use $\dot{C}$ for the term that we use to refer to the closed set of strings C'. We then have the following types of equations, where X, Y, Z are subsets of F.

$$\begin{array}{rl} \dot{a} \,\dot{\subseteq}\, \dot{X} & \text{if } a \in X' \\ \dot{X} \bullet \dot{Y} \,\dot{\subseteq}\, \dot{Z} & \text{if } X'Y' \subseteq Z' \\ \dot{X} \wedge \dot{Y} \,\dot{\subseteq}\, \dot{Z} & \text{if } X' \cap Y' \subseteq Z' \end{array}$$

The DLG specifies a subset of these equations $\mathcal{E}$ and there is a simple cubic time algorithm for computing for any string w, the maximal set of these contexts C such that $\mathcal{E} \vdash \dot{w} \,\dot{\subseteq}\, \dot{C}$. Language membership is then determined by the presence of the empty context $\Box$, equivalent to $\dot{L}$ in the set C.

6 Discontinuity

So far we have considered continuous individual strings and models based on simple concatenation. There are three related systems: distributional learning based on the relation between contexts and strings; Lambek calculus and Lambek grammars which we can view as the logic of these structures, and context free grammars.

Given the inadequacies of context-free formalisms in general, it is natural to extend this approach to mildly context sensitive formalisms, which are directly or indirectly based on modeling *discontinuous strings*. The formalism most direct analogous to CFGs is the class of multiple context free grammars [24] (MCFGs) and we can think of the displacement calculus [1] as its logic. Distributional learning has also made the same transition [25]. For simplicity, we will just consider the generalisation to strings with one gap (two components); the generalisation to k components is straightforward but requires a more elaborate notation.

We define a 2-word to be a pair of strings (an element of $\Sigma^* \times \Sigma^*$) that we write (v_1, v_2). We define a 2-context to be a string with 2 gaps: which we can write $u_0 \Box u_1 \Box u_2$. We can combine this with a pair of strings to get a string: $u_0 \Box u_1 \Box u_2 \odot (v_1, v_2) = u_0 v_1 u_1 v_2 u_2$. Given a particular language, this defines a relation between 2-contexts and 2-words. We again define the polar maps given sets of 2-contexts C_2 and 2-words S_2.

$$C_2' = \{(v_1, v_2) | \forall f \in C_2, f \odot (v_1, v_2) \in L\}$$

$$S_2' = \{f | \forall (v_1, v_2) \in S_2, f \odot (v_1, v_2) \in L\}$$

Again we say that a set of 2-words, S_2 is closed if $S_2'' = S_2$. The set of all closed sets of 2-words in a language forms a lattice, which we call $\mathfrak{B}^2(L)$. These closed sets have an interesting structure, and are closely related, as one would expect, to the elements of $\mathfrak{B}(L)$, which we will now write $\mathfrak{B}^1(L)$. Indeed suppose X_2 is a closed set of 2-words (an element of $\mathfrak{B}^2(L)$). Suppose $X_2 \supseteq Y_1 \times Z_1$, for sets of strings Y_1, Z_1, then $X_2 \supseteq Y_1'' \times Z_1''$; in other words we can write any element of $\mathfrak{B}^2(L)$ as $\bigcup_i Y_1^i \times Z_1^i$, as a union of products of the order-1 lattice. Therefore there is a natural map from $\mathfrak{B}(L) \times \mathfrak{B}(L)$ to $\mathfrak{B}^2(L)$ given by $h(X, Y) = (X \times Y)''$. Indeed we can consider these elements to define a relation between two closed sets of strings: it may be the case that an element of $\mathfrak{B}^2(L)$ is equal to $X \times Y$ for some $X, Y \in \mathfrak{B}^1(L)$, in which case the relation is trivial and there is no gain in using the order 2 lattice. In the interesting cases of non-context-free languages, this relation in general will be infinite in the sense that it can only be defined as an infinite union of products. In linguistically interesting cases this relation will often correspond to the relation between a displaced constituent (e.g. a *wh*-phrase in English) and the constituent that it has been displaced out of.

We can define a number of concatenation operations, and their associated residuals. For two 2-words, there are a number of ways that they can be concatenated to form another 2-word or a word. For example, (u_1, v_1) and (u_2, v_2) could be combined to form $(u_1 u_2, v_1, v_2), (u_1, u_2 v_1 v_2), (u_1 u_2, v_2 v_1)$ and many others. For brevity we will only consider some of these. Given sets of 2-words X_2, Y_2, we can define a family of concatenation operations of which we just show two: $X_2 \oplus Y_2 = \{(u_1 u_2, v_1 v_2) | (u_1, v_1) \in X_2, (u_2, v_2) \in Y_2\}''$, $X_2 \ominus Y_2 = \{(u_1 u_2, v_2 v_1) | (u_1, v_1) \in X_2, (u_2, v_2) \in Y_2\}''$. We have taken the closure of the resulting set of multiwords, giving in these cases associative binary operations on $\mathfrak{B}^2(L)$. The operation $\ominus$ corresponds to the MCFG production $Z(u_1 v_1, v_2, u_2) := X(u_1, u_2), Y(v_1, v_2)$ in Horn clause notation. Indeed $\{(\lambda, \lambda)\}''$ is a left and right identity for both operations. Unsurprisingly we can again define the left and right residuals of the operations which will give us some more residuated structures.

These operations are

$$X_2/_{\ominus}Y_2 = \{(v_1, v_2) \mid \forall (u_1, u_2) \in Y_2 (v_1u_1, u_2v_2) \in X_2\} \tag{5}$$

$$Y_2\backslash_{\ominus}X_2 = \{(v_1, v_2) \mid \forall (u_1, u_2) \in Y_2 (u_1v_1, v_2u_2) \in X_2\} \tag{6}$$

The sets defined here are closed, and satisfy the equations $X_2 \ominus Y_2 \subseteq Z_2$ iff $X_2 \subseteq Z_2/_{\ominus}Y_2$ iff $Y_2 \subseteq X_2\backslash_{\ominus}Z_2$.

At this level of the hierarchy again we have the same metatheoretical issues. We can represent the tuples of strings directly using a MCFG; the same arguments that we used in the case of CFGs suggest that the nonterminals of dimension 2 should represent closed sets of 2-words. We can view the logic, in this case Morrill's discontinuous calculus, as the equational theory of this algebraic structure. Finally, these can be learned using an extension of the distributional techniques we discussed earlier [25, 26] .

7 Discussion

The standard model theory for the Lambek calculus uses free models of various types. In particular we have the standard models: the free semigroup models 2^{Σ^+} (or 2^{Σ^*} if we allow empty antecedents). It is easy to verify that the Lambek calculus is sound with respect to these free models, and it can also be proved to be complete [2]. Morrill [1] says:

> Since language in time appears to satisfy the cancellation laws, the free semigroup models appear to be the most ontologically committed and therefore scientifically incisive models.

The cancellation laws are of the form $u \bullet v = u \bullet w$ implies $v = w$. Clearly if equality here just means equality of strings, then this law is true, since the strings are part of a free semigroup. But if $=$ refers to linguistic/distributional equality, then this is not true: for example, words can be ambiguous in isolation and unambiguous in context. "the can" might be distributionally identical to "the jug" but "can" and "jug" are very different.

Lambek's stipulation that the algebraic structures must be free is thus, from our point of view partially correct. The strings are generated freely giving us the free monoid Σ^*; but the subsets of these are not – we do not have the free join semilattice but only the lattice of closed sets.

7.1 Proof theory

In general the relation of syntactic entailment is undecidable. Post [27] showed that in general determining whether $x \equiv_T y$ is undecidable with respect to a Thue system. Nonetheless there are subclasses of these sets of equations which have inference procedures that are efficient for certain classes of conclusions.

Recall that in order to parse, we are only interested in proving equations of the form $\dot{w} \dot{\subseteq} \dot{L}$.

Lexicalisation is one solution. By requiring all the equations to be of the form $\dot{a} \dot{\subseteq} T$, we can reduce the problem to one which is decidable since it is, approximately, the equational theory of residuated lattices. This solves that part of the problem. But this is only one solution. For suitable other classes of equations the required proofs can be solved efficiently: indeed in polynomial time, rather than the NP-hardness of the Lambek calculus.

7.2 Conclusion

The main claim of this paper is that since languages have an intrinsic residuated lattice structure, grammatical formalisms that use residuated lattices, such as Lambek grammars should be based on this structure. The crucial consequence of this is that we can then learn these grammars efficiently under various different learning paradigms. We deviate in two important respects from Lambek's metaprogram – we abandon the free models, and we may abandon lexicalisation. Lambek says:

> If we take such a program seriously we are not allowed to state a rule such as $N_s \subseteq S/(N\backslash S)$ which although plausible[4] is neither listed in the dictionary nor derivable from general principles. . . .

We might say that we are not concerned with rules that are 'plausible', but rather with rules that are true or correct in a given language. If they are true, and we can reason efficiently with them using a sound logical calculus, and in addition learn them, then there seems little reason to forbid them.

Pereira [28] in his review of Morrill's 1994 book on logical grammar says:

> Types and allowed type inferences are not arbitrary formal machinery but instead reflect precisely the combinatory possibilities of the underlying prosodic algebra.

Here our prosodic algebra is much simpler than the one Pereira is discussing, but the observation is very apt. Grammars are theories and languages (or algebras associated with them) are models. Substructural logics can be viewed as the equational theories of certain types of lattices; logical grammars then can be viewed as theories of syntactic concept lattices. We can view this as a principled misinterpretation of Chomsky's dictum [29],

> A grammar of the language L is essentially a theory of L.

Of course, Chomsky meant a *scientific* theory. We, in contrast, propose viewing a grammar as a finite set of equations whose deductive closure defines the language: as a logical theory in other words.

This reduces the arbitrariness of grammars – there is a single specific algebraic object that we are trying to model, be it the syntactic concept lattice, the

[4] N_s here refers to a mass noun like 'water'.

syntactic monoid, or some multisorted algebra of discontinuous strings as in Section 6. Many different formalisms can be framed in this way, that depend on the type of algebra that we are modeling. For suitable restricted sets of equations we can perform the relevant inferences efficiently.

Learnability has always been a central concern of modern linguistics, and the uniformity of the syntax semantics interface as conceived in the categorial grammar tradition is very appealing from this perspective. As Pereira says:

> ... uniformity must also go backwards, if the use and meaning of a sign is to be induced from its appearance with other signs of appropriate type.

By basing the logic on a well defined observable structure, we can learn the use cleanly; and perhaps ultimately the meaning as well.

Acknowledgments

I am grateful to Glynn Morrill, Shalom Lappin, Hans Leiss, Ryo Yoshinaka, Zhaohui Luo and Robin Adams for helpful discussions. Any errors are of course my own.

References

1. Morrill, G.: Categorial grammar: Logical syntax, semantics and processing. Oxford University Press (2011)
2. Pentus, M.: Models for the Lambek calculus. Annals of Pure and Applied Logic **75**(1–2) (1995) 179–213
3. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly **65**(3) (1958) 154–170
4. Lambek, J.: Categorial and categorical grammars. In Oehrle, R.T., Bach, E., Wheeler, D., eds.: Categorial grammars and natural language structures. Volume 32. D. Reidel (1988) 297–317
5. Buszkowski, W.: Compatibility of a categorial grammar with an associated category system. Mathematical Logic Quarterly **28**(14-18) (1982) 229–238
6. Buszkowski, W., Penn, G.: Categorial grammars determined from linguistic data by unification. Studia Logica **49**(4) (1990) 431–454
7. Kanazawa, M.: Learnable classes of categorial grammars. PhD thesis, Stanford University (1994)
8. Angluin, D.: Inference of reversible languages. Journal of the ACM **29**(3) (1982) 741–765
9. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75**(2) (1987) 87–106
10. de la Higuera, C.: Grammatical inference: learning automata and grammars. Cambridge University Press (2010)
11. Clark, A.: Distributional learning of some context-free languages with a minimally adequate teacher. In Sempere, J., Garcia, P., eds.: Grammatical Inference: Theoretical Results and Applications. Proceedings of the International Colloquium on Grammatical Inference. Springer (September 2010) 24–37

12. 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
13. Clark, A.: A learnable representation for syntax using residuated lattices. In: Proceedings of the 14th Conference on Formal Grammar, Bordeaux, France (2009)
14. Clark, A.: Learning context free grammars with the syntactic concept lattice. In Sempere, J., Garcia, P., eds.: Grammatical Inference: Theoretical Results and Applications. Proceedings of the International Colloquium on Grammatical Inference. Springer (2010) 38–51
15. Yoshinaka, R.: Towards dual approaches for learning context-free grammars based on syntactic concept lattices. In: Developments in Language Theory. (2011) 429–440
16. Yoshinaka, R.: Integration of the dual approaches in the distributional learning of context-free grammars. In: LATA. (2012) 538–550
17. Galatos, N., Jipsen, P., Kowalski, T., Ono, H.: Residuated lattices: an algebraic glimpse at substructural logics. Elsevier (2007)
18. Costa Florêncio, C.: Learning categorial grammars. PhD thesis, Utrecht University (2003)
19. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics **6**(4) (2001) 519–535
20. Kanazawa, M.: The Lambek calculus enriched with additional connectives. Journal of Logic, Language and Information **1**(2) (1992) 141–171
21. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75**(2) (1987) 87–106
22. Clark, A.: Towards general algorithms for grammatical inference. In: Proceedings of ALT. Springer, Canberra, Australia (October 2010) 11–30 Invited Paper.
23. Clark, A.: Efficient, correct, unsupervised learning of context-sensitive languages. In: Proceedings of the Fourteenth Conference on Computational Natural Language Learning, Uppsala, Sweden, Association for Computational Linguistics (July 2010) 28–37
24. Seki, H., Matsumura, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theoretical Computer Science **88**(2) (1991) 229
25. Yoshinaka, R.: Efficient learning of multiple context-free languages with multidimensional substitutability from positive data. Theoretical Computer Science **412**(19) (2011) 1821 – 1831
26. Yoshinaka, R., Clark, A.: Polynomial time learning of some multiple context-free languages with a minimally adequate teacher. In: Proceedings of the 15th Conference on Formal Grammar, Copenhagen, Denmark (2010)
27. Post, E.: Recursive unsolvability of a problem of Thue. The Journal of Symbolic Logic **12**(1) (1947) 1–11
28. Pereira, F.: Review of Type logical grammar: categorial logic of signs by Glyn Morrill. Computational Linguistics **23**(4) (1997) 629–635
29. Chomsky, N.: Syntactic Structures. Mouton (1957)