Electronic Notes in Theoretical Computer Science 53 (2001)
URL: `http://www.elsevier.nl/locate/entcs/volume53.html` 14 pages

Learnability of type-logical grammars

Sean A. Fulop [1,2]

Depts. of Linguistics and Computer Science
The University of Chicago
Chicago, U.S.A.

Abstract

A procedure for learning a lexical assignment together with a system of syntactic and semantic categories given a fixed type-logical grammar is briefly described. The logic underlying the grammar can be any cut-free decidable modally enriched extension of the Lambek calculus, but the correspondence between syntactic and semantic categories must be constrained so that no infinite set of categories is ultimately used to generate the language. It is shown that under these conditions various linguistically valuable subsets of the range of the algorithm are classes identifiable in the limit from data consisting of sentences labeled by simply typed lambda calculus meaning terms in normal form. The entire range of the algorithm is shown to be not a learnable class, contrary to a mistaken result reported in a preliminary version of this paper. It is informally argued that, given the right type logic, the learnable classes of grammars include members which generate natural languages, and thus that natural languages are learnable in this way.

1 Introduction

Recent investigations (e.g. [12,13]) have shown the potential for the description of natural language syntax promised by forms of Lambek's syntactic calculus [10] enriched by multiple combination modes and families of unary modalities. Any version of Lambek's calculus in this landscape of logical systems will be called a *type logic* in keeping with current practice. A *type-logical grammar* is then a triple $G = (V_G, I_G, R_G)$ consisting of a vocabulary V_G, a lexical assignment function I_G, and a type logic R_G.

[1] Thanks to my supervisor Ed Stabler, my de facto co-chair Michael Moortgat, and Makoto Kanazawa for his inspiring work. Special appreciation to Christophe Costa Florencio for helping me iron out the mistakes in the preliminary version. This work was supported in part by a UCLA Dissertation Year Fellowship, NSF LIS grant 9720410, an OTS-UCLA exchange grant awarded to Prof. Dr. M. Moortgat, and a Research Incentive grant from the University of Chicago Dept. of Computer Science.

[2] Email: `sfulop@uchicago.edu`

This paper proceeds according to the following:

- Type-logical grammar as a framework for syntactic description is briefly summarized.
- The syntax-semantics connection between type logic and the lambda calculus is outlined, as a generalization of the Curry-Howard morphism.
- We outline a procedure **OUTL** for learning type-logical lexicons from sentences plus lambda terms, which works for any type logic meeting certain conditions.
- It is shown that the entire class of languages generated by the grammars discoverable using **OUTL** is not in general a learnable class, in Gold's [5] sense of identifiability in the limit.
- It is then shown how restricting to certain subclasses of the language class $L(\text{Rng}(\text{OUTL}))$ can give rise to learnability.
- Finally, the ramifications for theoretical psycholinguistics and computational induction of grammars are mentioned.

The focus here is on the learnability results; this paper is part of a larger research project intended to show the feasibility of a particular implementation within the type-logical framework of the psycholinguistic hypothesis that human languages can be learned via *semantic bootstrapping* [14].

2 Type-logical grammar

Definition 2.1 A *type-logical grammar* can be defined as a triple $G = (V_G, I_G, R_G)$ such that:

(i) V_G, the *vocabulary* of G, is a non-empty finite set;

(ii) I_G, the *lexicon* of G, is a function which to each $v \in V_G$ assigns a finite set of types;

(iii) R_G, the *calculus* of G, is a type logic.

The non-associative Lambek calculus NL serves as a "base logic" for the kinds of type logical grammars that the present paper applies to. The systems of inference rules shown below conform to the logical style of Gentzen's [4] *sequent calculus.*

Definition 2.2

(Axiom)
$$A \Rightarrow A$$

(Cut)
$$\frac{\Delta \Rightarrow A \quad \Gamma[A] \Rightarrow C}{\Gamma[\Delta] \Rightarrow C}$$

(/ L)
$$\frac{\Delta \Rightarrow B \quad \Gamma[A] \Rightarrow C}{\Gamma[(A/B \diamond \Delta)] \Rightarrow C}$$

(\ L)
$$\frac{\Delta \Rightarrow B \quad \Gamma[A] \Rightarrow C}{\Gamma[(\Delta \diamond B\backslash A)] \Rightarrow C}$$

(/ R)
$$\frac{(\Gamma \diamond B) \Rightarrow A}{\Gamma \Rightarrow A/B}$$

(\ R)
$$\frac{(B \diamond \Gamma) \Rightarrow A}{\Gamma \Rightarrow B\backslash A}$$

Extensions of NL have been developed in the literature [11,13] which employ unary operators that are analogous to those found in modal logics. Here are the sequent rules of inference governing the unary "modal" operators, using their corresponding structural operator.

($\Diamond$L)
$$\frac{\Gamma[\langle A\rangle] \Rightarrow B}{\Gamma[\Diamond A] \Rightarrow B}$$

($\Diamond$R)
$$\frac{\Gamma \Rightarrow A}{\langle\Gamma\rangle \Rightarrow \Diamond A}$$

($\Box^{\downarrow}$L)
$$\frac{\Gamma[A] \Rightarrow B}{\Gamma[\langle\Box^{\downarrow} A\rangle] \Rightarrow B}$$

($\Box^{\downarrow}$R)
$$\frac{\langle\Gamma\rangle \Rightarrow A}{\Gamma \Rightarrow \Box^{\downarrow} A}$$

A major purpose of introducing modal operators is to license the use of structural rules, in effect restricting their applicability to certain contexts. Typically, extended type logics also employ more than one "family" of slash operators, and these are indexed using subscripts. The different families of slashes are sometimes called *modes of combination.*

A simple example illustrating the use of the modal enrichments is provided by the treatment of English wh-extraction to form a relative clause, taken from Puite and Moot [15]. Take the following lexicon, in which the wh-word has unary modal operators in its assigned type.

(1)
$$\begin{aligned} I_G(\text{agent}) &= n \\ I_G(\text{Mulder}) &= np \\ I_G(\text{liked}) &= (np\backslash s)/np \\ I_G(\text{which}) &= (n\backslash n)/(s/\Diamond_a\Box^{\downarrow}{}_a np) \end{aligned}$$

The following structural rule permits associativity only in the presence of the appropriate structural modal environment.

(Assoc)
$$\frac{\Delta[(\Gamma_1 \diamond \Gamma_2) \diamond \langle\Gamma_3\rangle^a] \Rightarrow C}{\Delta[\Gamma_1 \diamond (\Gamma_2 \diamond \langle\Gamma_3\rangle^a)] \Rightarrow C} \qquad \frac{\Delta[\Gamma_1 \diamond (\Gamma_2 \diamond \langle\Gamma_3\rangle^a)] \Rightarrow C}{\Delta[(\Gamma_1 \diamond \Gamma_2) \diamond \langle\Gamma_3\rangle^a] \Rightarrow C}$$

The figure shows a proof of the complex noun *agent which Mulder liked,* which has a desired constituent structure.

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
np\colon \text{Mulder} \Rightarrow np
\qquad
\dfrac{np\colon \text{which} \Rightarrow np \quad s\colon [\text{Mulder, [liked, which]]} \Rightarrow s}{np \diamond np\backslash s\colon [\text{liked, which}] \Rightarrow s}\,\mathrm{L}\backslash
}{np \diamond ((np\backslash s)/np \diamond np)\colon [\text{Mulder, [liked, which]]} \Rightarrow s}\,\mathrm{L}/
}{np \diamond ((np\backslash s)/np \diamond \langle \Box^{\downarrow}{}_{a} np\rangle^{a})\colon [\text{Mulder, [liked, which]]} \Rightarrow s}\,\mathrm{L}\Box^{\downarrow}{}_{a}
}{(np \diamond (np\backslash s)/np) \diamond \langle \Box^{\downarrow}{}_{a} np\rangle^{a}\colon [[\text{Mulder, liked], which}] \Rightarrow s}\,\textsc{Assoc}
}{(np \diamond (np\backslash s)/np) \diamond \diamond\Box^{\downarrow}{}_{a} np\colon [[\text{Mulder, liked], which}] \Rightarrow s}\,\mathrm{L}\diamond
}{np \diamond (np\backslash s)/np\colon [\text{Mulder, liked}] \Rightarrow s/\diamond\Box^{\downarrow}{}_{a} np}\,\mathrm{R}/
\qquad
\dfrac{n\colon \text{agent} \Rightarrow n \quad n\colon [\text{agent, [which, [Mulder, liked]]]} \Rightarrow n}{n \diamond n\backslash n\colon [\text{agent, [which, [Mulder, liked]]]} \Rightarrow n}\,\mathrm{L}\backslash
}{n \diamond ((n\backslash n)/(s/\diamond\Box^{\downarrow}{}_{a} np) \diamond (np \diamond (np\backslash s)/np))\colon [\text{agent, [which, [Mulder, liked]]]} \Rightarrow n}\,\mathrm{L}/
$$

3 Grammar discovery from semantics

3.1 The syntax-semantics connection

A general algorithm has been developed [3] for learning a syntactic category system for a natural language together with a lexical assignment that determines a completely descriptive grammar. The syntax can in principle be handled using any cut-free decidable multimodal type-logical grammar [11]. The learning data consists of *term-labeled strings,* i.e. sentences annotated by typed lambda calculus meaning recipes with either variable semantic types on the subterms, or with no types on the subterms. The lambda calculus is used in a standard fashion to model the compositional meaning structures of natural language; an example of a term-labeled string is:

(2) $((\mathbf{loves}^{(\mathrm{s}\leftarrow\alpha)\leftarrow\alpha}(\mathbf{Mary}^{\alpha}))(\mathbf{John}^{\alpha}))^{\mathrm{s}}\colon \langle \mathrm{John}, \mathrm{loves}, \mathrm{Mary}\rangle$

alternatively, without subterm types

(3) $((\mathbf{loves}(\mathbf{Mary}))(\mathbf{John}))^{\mathrm{s}}\colon \langle \mathrm{John}, \mathrm{loves}, \mathrm{Mary}\rangle$

The meaning recipes show the basic compositional construction of the sentence meaning in terms of application and abstraction, and can be used as directionally non-specific recipes for the construction of type-logical proofs of the labeled sentence which correspond via a generalized Curry-Howard homomorphism.

In more detail, let us recall that the Curry-Howard formulae-as-types interpretation induces a morphism from lambda terms to proofs, such that a lambda term can be used as a recipe to construct a proof in a logic whose operators correspond to the lambda type operators. Considering the simply typed lambda calculus, the terms can be used to construct valid proofs in the positive implicational propositional logic, whose only logical operator is the implication.

Now, for type logics based on the non-associative Lambek calculus, there is no longer a direct correspondence with the simply typed lambda calculus, because there are more logical operators in the logic than just the implication found in the lambda calculus types. The key difference is that type logics in general will be sensitive to the directional (left-right) relationship among the formulae, and there may also be modal operators and structural rules invoked that have no counterpart in the simply typed lambda calculus. In fact, however, any type logic R_G with the typical pair of slash operators can be shown to induce a fragment Λ_{R_G} of the ordinary typed lambda calculus such that for every sequent $\Gamma \Rightarrow s_G$ in R_G provable by some proof Π there will be a term $M^{s_G} \in \Lambda_{R_G}$ such that M is a *construction modulo direction* of Π by a generalization of the Curry-Howard isomorphism [7] to a homomorphism. The initial idea for this is found in van Benthem [18].

Definition 3.1 The mapping τ from syntactic types in the type logic to semantic types in the lambda calculus is defined as follows. The semantic types that are mapped to in this way are said to be *equivalent modulo direction of application* (emda) to the correponding syntactic types, and vice versa. The function $M(\cdot)$ is used as a schematic, in which M stands for any string of unary modal operators. The subscript i signifies the possibility of having multiple families of slash operators.

$$\tau(M(c)) = c' \quad \text{for corresponding primitive types } c, c';$$
$$\tau(M(A/_iB)) = \tau(M(B\backslash_iA)) = \tau(A) \leftarrow \tau(B)$$

Notice that, in general, a single semantic type is said to be equivalent modulo direction to an infinite number of possible syntactic types.

3.2 A discovery procedure

In order to first discover a *general form* lexicon as an intermediate step, which assigns syntactic types whose primitive types are distinct variables except for the principal type constant 's' which is assigned to sentences, our algorithm is intended to learn from a sample of term-labeled strings such as the above example, whose labels either contain no explicit semantic types on the subterms (they are *unsubtyped*), or are subtyped but contain only such semantic types in which the primitive types are all variables except for the principal type 's' of a sentence. The complete algorithm is charged with learning the system of categories (other than 's') together with the lexical assignment function, for a fixed vocabulary and type logic.

The broad outline of the procedure, called Optimal Unification for Type-Logical grammars (**OUTL**), is as follows:

(i) Given a sample D of unsubtyped term-labeled strings, compute a counterpart sample D' of subtyped term-labeled strings whose terms are typed in a most general way. This can be accomplished using some kind of principle type algorithm, such as the one which is discussed at length in [6].

(ii) Compute the set of general form type-logical lexicons GFTL(D'), in each of which distinct variable primitive types will each occur atomically only once, and such lexicons will generate only the sample D' and not an infinite language. This is accomplished by taking the following steps:
   (a) For each term-labeled string in the sample, determine all proofs in the type logic at hand which can be constructed by using the subtyped lambda term as a construction term modulo direction, and which are also compatible with the word order that is evident in the sentence.
   (b) Non-deterministically select one proof for each term-labeled string in the entire sample; a general form lexicon can then be read off from the

types labeling the words. Repeat this step until all different ways of selecting one proof for each term-labeled string have been exhausted. This will provide all general form lexicons that could generate the learning sample.

(iii) Find all of the *optimal unifications* [1] of each of the lexicons in GFTL(D').

The language class generated by the entire range of OUTL (which assumes a fixed decidable cut-free type logic) is not learnable in Gold's [5] sense, but sufficiently large subclasses are learnable provided that the generalized Curry-Howard correspondence is also finitary in the following sense.

Definition 3.2 The relation of equivalence modulo direction of application (emda) between semantic and syntactic types is said to be *finitary* just when for each semantic type τ, the set $\{T \mid \mathrm{emda}(T, \tau)\}$ of syntactic types equivalent to it is finite, and for each syntactic type T, the set $\{\tau \mid \mathrm{emda}(T, \tau)\}$ of semantic types equivalent to it is finite.

No class of type-logical languages that does not meet this condition is learnable by the above discovery procedure.

The logics found in linguistic applications, including of course the basic nonassociative Lambek calculus but also various extended systems such as the Dutch system presented in [12], either already satisfy the above restrictions, or can be made to by some ad hoc means to force a finitary syntax-semantics correspondence. Note, however, that no type logic enriched with unary modalities can have a finitary emda relation with the simply typed lambda calculus in general. The sequent proof corresponding to a given lambda term might have any number of modal operators tacked onto its consequent type; a indefinite combinatorial explosion of available sequences of modal and structural rules would then have to be explored to find the ones that eventually get rid of all of the modals so that the logical rule that corresponds to some application or abstraction in the lambda term can finally be invoked. To try to quell this explosion, we can stipulate a finitary correspondence by saying that only syntactic types with fewer than some small number k modal operators will be considered in the search for corresponding proofs. This way of meeting the finitary correspondence requirement is mathematically ad hoc, but does not seem linguistically unrealistic. Do we really wish to countenance an infinite number of syntactic categories for a given semantic category? There is probably some definite upper bound on the number of distinct syntactic behaviors that a member of a particular semantic category will be found to have in natural language, and in practice, some rather sophisticated linguistic phenomena can already be handled with a maximum of two modal operators adorning the types.

To demonstrate how the learning turns out, a simple example shows the results of applying the OUTL procedure to two different samples of four annotated sentences, which have the same vocabulary. The procedure settles on a grammar for the same language in each case (the same grammar too, in fact).

$$
\begin{array}{lr@{\,:\ }l}
(4) & (\mathbf{sings}(\mathbf{John}))^s & \langle \text{John}, \text{sings} \rangle \\
 & ((\mathbf{loves}(\mathbf{Mary}))(\mathbf{John}))^s & \langle \text{John}, \text{loves}, \text{Mary} \rangle \\
 & ((\mathbf{loves}(\mathbf{a}(\mathbf{man})))(\mathbf{Mary}))^s & \langle \text{Mary}, \text{loves}, \text{a}, \text{man} \rangle \\
 & ((\mathbf{sees}(\mathbf{John}))(\mathbf{a}(\mathbf{man})))^s & \langle \text{a}, \text{man}, \text{sees}, \text{John} \rangle \\
(5) & (\mathbf{sings}(\mathbf{Mary}))^s & \langle \text{Mary}, \text{sings} \rangle \\
 & ((\mathbf{loves}(\mathbf{John}))(\mathbf{Mary}))^s & \langle \text{Mary}, \text{loves}, \text{John} \rangle \\
 & ((\mathbf{loves}(\mathbf{Mary}))(\mathbf{a}(\mathbf{man})))^s & \langle \text{a}, \text{man}, \text{loves}, \text{Mary} \rangle \\
 & ((\mathbf{sees}(\mathbf{a}(\mathbf{man})))(\mathbf{John}))^s & \langle \text{John}, \text{sees}, \text{a}, \text{man} \rangle
\end{array}
$$

The general form lexicons discovered for the above two samples by exploiting the Curry-Howard homomorphism optimally unify to the same lexicon, presented as an assignment function I_G:

$$
\begin{array}{lr@{\ =\ }l}
(6) & I_G(\text{Mary}) & \alpha \\
 & I_G(\text{sings}) & \alpha\backslash s \\
 & I_G(\text{loves}) & (\alpha\backslash s)/\alpha \\
 & I_G(\text{John}) & \alpha \\
 & I_G(\text{a}) & \alpha/\beta \\
 & I_G(\text{man}) & \beta \\
 & I_G(\text{sees}) & (\alpha\backslash s)/\alpha
\end{array}
$$

It should be noted that this implementation of semantic bootstrapping uses much less information than has frequently been considered in such efforts. The basic requirements of Pinker's [14] proposal for a bootstrapping procedure include the Canonical Structure Realization, through which the syntactic realization of known semantic categories is provided as a part of innate Universal Grammar. This requires the would-be semantic bootstrapper to already know the system of semantic categories as well as their corresponding syntactic ones. An implementation which follows this tack is presented by [16]. Our implementation, in contrast, does not provide complete information about syntactic structure, and it provides no specific information about the particular semantic or syntactic categories which should be used to generate the language.

4 Learnability of optimally unified lexicons

Definition 4.1 [8] Let $\langle \Omega, \mathfrak{S}, L \rangle$ be a grammar frame, consisting of a set Ω of grammars, a set $\mathfrak{S}$ of expressions (sentences), and a function L which maps from the grammars to sets of expressions (i.e. languages). A *learning function* is a partial function φ that maps non-empty finite sequences of sentences to grammars:

$$\varphi : \bigcup_{k \geq 1} \mathfrak{S}^k \to \Omega.$$

$\mathfrak{S}^k$ denotes the set of k-ary sequences of sentences. A *learning algorithm* is one that computes a learning function.

Definition 4.2 Let a grammar frame $\langle \Omega, \mathfrak{S}, L \rangle$ be given, let $\mathcal{G} \subseteq \Omega$. A learning function φ is said to *learn* $\mathcal{G}$ if the following condition holds:
for every language ℓ in $L(\mathcal{G})$,
for every infinite sequence $\langle s_i \rangle_{i \in \mathbb{N}}$ that enumerates the elements of ℓ (i.e. $\{s_i \mid i \in \mathbb{N}\} = \ell$),
there exists some G in $\mathcal{G}$ such that $L(G) = \ell$ and φ converges to G on $\langle s_i \rangle_{i \in \mathbb{N}}$. [8]

4.1 A negative result

Definition 4.3 A class $\mathcal{L}$ of languages is said to *have a limit point* if there exists an infinite sequence $\langle L_n \rangle_{n \in \mathbb{N}}$ of languages in $\mathcal{L}$ such that

$$L_0 \subset L_1 \subset \cdots \subset L_n \subset \cdots$$

(we call this an infinite *ascending chain*) and there is another language L in $\mathcal{L}$ such that

$$L = \bigcup_{n=0}^{\infty} L_n.$$

The language L is said to be a *limit point* of $\mathcal{L}$. [8]

Lemma 4.4 ([8]) *If $L(\mathcal{G})$ has a limit point then $\mathcal{G}$ is not learnable.*

Thus, we can show a class of languages is not learnable by showing it has a limit point.

Theorem 4.5 *The language class $\Lambda L(\mathrm{Rng}(OUTL))$ generated by the range of the $OUTL$ algorithm is not a learnable class of (term-labeled) languages for any type logic.*

Proof. In a fashion analogous to the proof of Kanazawa's [8] Theorem 7.20, we prove that the term-labeled language class $\Lambda L(\mathrm{Rng}(\mathrm{OUTL}))$ has a limit point, even when using just the AB classical type logic (equivalent to the binary combination version of classical categorial grammar) restricted to the forward slash.

(i) Consider the following lexical assignment involving a single vocabulary item:

(7) $I_G(\mathrm{a}) = \{x, x/x, (s/(x/x))/x\}$

This assignment is optimally unified. The term-labeled language generated using AB and I_G consists of the infinite set $\{\text{tls}_n\}$ of term-labeled strings, where the items T_n are defined as follows:

$$(\mathbf{a}^{(s\leftarrow(x\leftarrow x))\leftarrow x}(\underbrace{\mathbf{a}^{x\leftarrow x}(\cdots}_{n \text{ times}}(\mathbf{a}^{x})\cdots)^{x})^{s\leftarrow(x\leftarrow x)}\mathbf{a}^{x\leftarrow x} : \langle \mathrm{a}_{n+3}\rangle_{n\geq 0}$$

in which the types are shown for convenience, so that the manner in which they are derived is easier to see.

(ii) Now, for each $n \geq 0$, define the type A_n by the following recurrence:

$$\begin{aligned} A_0 &\overset{\text{def}}{=} x \\ A_1 &\overset{\text{def}}{=} (s/x)/x \\ A_{n+2} &\overset{\text{def}}{=} (s/A_{n+1})/(s/A_n). \end{aligned} \tag{8}$$

Next, let I_{Gn} name the following assignment for all $n \geq 0$:

$$I_{Gn}(\mathrm{a}) = \{A_0, \ldots, A_{n+1}\}. \tag{9}$$

All lexicons I_{Gn} are optimally unified, and $\Lambda L(G_n) = \{\text{tls}_0, \ldots, \text{tls}_n\}$ for all $n \in \mathbb{N}$, so the grammars G_n define an infinite ascending chain of languages.

(iii) Finally, it can be verified (most easily by running the software) that I_G is in Rng(OUTL), as are all of the I_{Gn}. In fact, $I_G \in \text{OUTL}(\{\text{tls}_0, \text{tls}_1\})$, and $I_{Gn} \in \text{OUTL}(\{\text{tls}_0, \ldots, \text{tls}_n\})$ for each $n \geq 0$.

(iv) Notice that $\Lambda L(G) = \bigcup_{n=0}^{\infty} \Lambda L(G_n)$, making $\Lambda L(G)$ a limit point in the range of OUTL.

□

4.2 Positive results

The following notion, though defined below as in [9], has its roots in Wexler and Hamburger [19].

Definition 4.6 The language $L \in \mathcal{L}$ is said to be an *accumulation point* just when there exists a sequence of finite sets $S_0, S_1, \ldots$ such that

(i) $\forall i \in \mathbb{N}\ S_i \subseteq S_{i+1}$;

(ii) $\bigcup_{i=0}^{\infty} S_i = L$;

(iii) $\forall i \in \mathbb{N}\ \exists L' \in \mathcal{L}(S_i \subseteq L' \ \&\ L' \subset L)$.

The existence of an accumulation point in a language class is both necessary and sufficient for the class to be unlearnable. Thus, we can prove a class learnable by showing it cannot have an accumulation point.

Definition 4.7 A syntactic type A which is a subtype of some type assigned by the lexicon I_G of a type-logical grammar G is said to be *useless* just when there is no proof available in G that uses A other than as a proper subtype.

In fact, it is apparent from the discovery procedure that the algorithm OUTL can never output a lexicon containing useless types.

Definition 4.8 A type-logical grammar is *k-valued* just when its lexicon I_G assigns no more than k types to any one vocabulary element.

Lemma 4.9 *Given any decidable type logic R_G meeting the finitarity condition of Def. 3.2, for any learning sample of term-labeled strings which exhausts the vocabulary V_G, there is a bounded number of distinct (modulo alphabetic variant) k-valued lexicons I_G without useless types for any k.*

Proof. The argument is combinatorial. Any sample of term-labeled strings determines a lexicon of semantic type schemata—a general form semantic lexicon. Clearly from the condition of Def. 3.2, there is some maximum number of distinct 1-valued lexicons corresponding to any such semantic lexicon, so long as there are no useless types permitted. The precise number can in principle be determined in any case, but it will depend on the nature of the semantic lexicon (which varies from sample to sample) and the specifics of the syntax-semantics correspondence (which varies from logic to logic). The same will be true of the 2-valued, 3-valued, etc. lexicons for any k. □

Theorem 4.10 *Given any decidable type logic R_G meeting the finitarity condition of Def. 3.2, no language class generated by a k-valued class of grammars in* Rng(*OUTL*) *can have an accumulation point.*

Proof. Suppose for some k, there were a class of grammars in Rng(OUTL) whose class of generated term-labeled languages has an accumulation point ΛL. By definition, $\Lambda L = \bigcup_{i=0}^{\infty} S_i$ is the infinite union of a weakly ordered chain of sets of term-labeled strings. Using the S_i as learning samples, no discovered I_{Gi} can have the full generating power of I_G which generates ΛL. This is just what it means to be an accumulation point.

Now, let us see that the actual situation contradicts this desideratum. Since the full I_G generating ΛL must be k-valued for some k, ΛL may use just a bounded number of semantic type schemata for each word. This bound will depend on k and on the nature of the type logic R_G. By Lemma 4.9, given any semantic lexicon of type schemata assigned to the vocabulary, there is a determinable number of distinct optimally unified k-valued lexicons I_{Gi} without useless types. For the complete semantic lexicon, which is bounded in extent, one of these I_{Gi} must actually generate ΛL using the assumed type logic R_G. Since the chain $\langle S_i \rangle$ converges to ΛL, all the languages ΛL_i which contain the sets must equal ΛL after a certain point m in the chain. This point m is a function of the number of distinct k-valued lexicons available, which in turn is a function of the assumed type logic R_G and of the semantic lexicon evident from the chain of samples. This proves that the hypothesized accumulation point ΛL cannot actually be one. □

The following is an immediate corollary from the facts about an accumulation point.

Corollary 4.11 *Given any decidable type logic R_G meeting the finitarity condition of Def. 3.2, the k-valued classes of grammars are learnable for all k.*

The preceding proof ultimately shows that the unlearnability of any class of type-logical lexicons depends on the possibility of unbounded lexical ambiguity within the class. That is why learnability is lost when the entire class of lexicons in the range of OUTL is considered, rather than just those with a maximum k types assigned to each word.

Definition 4.12 If D is a finite set of term-labeled strings, let

$$LCTL(D) \overset{\text{def}}{=} \{G \in OUTL(D) \mid \forall G' \in OUTL(D)\ (|I_G| \leq |I_{G'}|)\}.$$

$LCTL(D)$ picks those optimal lexicons that have smallest cardinality. The following now seems to follow as a corollary of Theorem 4.10, though the argument is as yet informal.

Conjecture 4.13 *Given any decidable type logic R_G meeting the finitarity condition of Def. 3.2, the language class $\Lambda L(\mathrm{Rng}(LCTL))$ is learnable.*

The argument here is provided in part by the above remark to the effect that unlearnability of any class of lexicons depends on the possibility of unbounded lexical ambiguity. This is because the accumulation point of the generated language class must ultimately be generated in two ways—once by a possibly bounded lexicon, and again by an "infinite" lexicon that is the limit of the lexicons generating the chain of sets $\langle S_i \rangle$. By selecting just those optimally unified lexicons with least cardinality, the algorithm LCTL ensures that these two ways of generating any language cannot exist, since only one of them will have least cardinality.

In terms of recognizing power, the exact class of languages generated by the class of type-logical grammars covered by these results is not known. Because several type-logical fragments that are covered by these results generate non-context-free natural language fragments (the Dutch fragment from [12,3], for example), one might conjecture that a subclass of the (properly) mildly context-sensitive languages is covered. It is clear, however, that the recursively enumerable languages outside the context-sensitive class are not covered in general, notwithstanding Carpenter's result [2] that the whole class of general multimodal type-logical grammars generate the entire Chomsky Type 0 language class. The reason for this is that Carpenter's result depends crucially on allowing the underlying type logics to be undecidable in general, with the ability to add or delete arbitrary structure during the course of a proof.

Since our type logics are required to be decidable, they do not seem to form a Turing-complete class.

5 Concluding remarks

It has been shown that wide classes of optimally unified type-logical lexicons are identifiable in the limit from term-labeled strings. We conjecture that these classes contain grammars strongly adequate for the description of natural languages. This is justified by noticing that, firstly, the learnable classes include grammars which generate languages beyond the context-free class, and secondly, the optimal unification procedure produces lexicons in which the assignment of distinct categories invariably reflects positive evidence in the learning sample of distinct syntactic behavior. The lexicons learned should thus in the limit assign all and only those syntactic and semantic categories which *function properly* in the language, a term we use to mean category differences exactly reflect syntactic differences. Familiar syntactic categories in linguistic theory such as "Verb" do not function properly in this sense, which is why subcategorization is required.

The above characterization of the present approach to learning grammars, if correct, has ramifications for the computational induction of grammars as well as for psycholinguistic ideas about language learning. It would be possible in principle to induce a precisely adequate grammar (provided one got the universal type logic correct in the first place) from term-labeled strings, which are easier to create based upon modern semantic theories of language than are direct syntactic training databases for grammar induction such as the Penn Treebank. The Treebank induction approach also has the major disadvantage that the parts of speech are already assigned to the words, which stretches even a charitable interpretation of language learning on the psycholinguistic side, and which may propagate damaging assumptions for natural language processing efforts.

References

[1] Buszkowski, W. and G. Penn, *Categorial grammars determined from linguistic data by unification*, Studia Logica **49** (1990), pp. 431–454.

[2] Carpenter, B., *The Turing-completeness of multimodal categorial grammars*, in: J. Gerbrandy, M. Marx, M. de Rijke and Y. Venema, editors, *JFAK: Essays dedicated to Johan van Benthem on the occasion of his 50th birthday*, Institute for Logic, Language, and Computation, University of Amsterdam, 1999 Available on CD-ROM at http://turing.wins.uva.nl.

[3] Fulop, S. A., "On the Logic and Learning of Language," Kluwer, Forthcoming.

[4] Gentzen, G., *Untersuchungen über das logische Schliessen*, Math. Zeitschrift **39** (1934), pp. 176–210, 405–431, english translation in [17].

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

[6] Hindley, J. R., "Basic Simple Type Theory," Cambridge University Press, 1997.

[7] Howard, W. A., *The formulas-as-types notion of construction*, in: J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, New York, 1980 pp. 479–490.

[8] Kanazawa, M., "Learnable Classes of Categorial Grammars," Ph.D. thesis, Stanford University (1994).

[9] Kapur, S., "Computational Learning of Languages," Ph.D. thesis, Cornell University (1991).

[10] Lambek, J., *The mathematics of sentence structure*, American Mathematical Monthly **65** (1958), pp. 154–170.

[11] Moortgat, M., *Categorial type logics*, in: J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, Elsevier, 1997 .

[12] Moortgat, M., *Meaningful patterns*, in: J. Gerbrandy, M. Marx, M. de Rijke and Y. Venema, editors, *JFAK: Essays dedicated to Johan van Benthem on the occasion of his 50th birthday*, Institute for Logic, Language, and Computation, University of Amsterdam, 1999 Available on CD-ROM at http://turing.wins.uva.nl.

[13] Morrill, G. V., "Type Logical Grammar: Categorial Logic of Signs," Kluwer, Dordrecht, 1994.

[14] Pinker, S., "Language Learnability and Language Development," Harvard University Press, Cambridge, MA, 1984.

[15] Puite, Q. and R. Moot, *Proof nets for the multimodal lambek calculus*, Technical Report 1096, University of Utrecht, Dept. of Mathematics (1999).

[16] Siskind, J., *Dispelling myths about language bootstrapping* (1991), manuscript, MIT AI Laboratory.

[17] Szabo, M., "The Collected Papers of Gerhard Gentzen," North-Holland, Amsterdam, 1969.

[18] van Benthem, J., "Language in Action," North-Holland, Amsterdam, 1991.

[19] Wexler, K. and H. Hamburger, *On the insufficiency of surface data for the learning of transformational language*, in: K. J. J. Hintikka, J. M. E. Moravcsik and P. Suppes, editors, *Approaches to Natural Language*, D. Reidel, Dordrecht, 1973 .