

Linguistic Analysis by Computer

Sean A. Fulop

Dept. of Linguistics
University of Chicago

October 24, 2002

We consider what it means to have a grammar for a natural language, and how the problem of making, or learning one, can be attacked using a computer.

Recent investigations (e.g. Moortgat 1999, Morrill 1994) have shown the possibilities for the description of natural language syntax promised by enriched forms of Lambek's syntactic calculus (Lambek 1958).

Any version of Lambek's calculus in this landscape of logical systems will be called a *type logic* in keeping with current practice.

This talk 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 simply typed lambda calculus is outlined, as a generalization of the Curry-Howard morphism.
- We outline a procedure **OUTL** for learning type-logical grammars from sentences plus lambda terms—*term-labeled strings*—which works for any type logic meeting certain conditions.

1 Motivation

There is a long list of grammar formalisms whose weak generating capacity has been proven sufficient to provide the string set of any human language. Type-logical grammar is one of these.

We can also provide a set of tree-structured sentences, if that is desired, and the generating capacity in this regard is also sufficient.

So, it is a fact that there is some type-logical grammar generating every natural language, under the usual idealized construal of what that means.

Yet it is very hard to actually design a grammar for a language in any of these formalisms; as a result, no such grammar has ever been exhibited.

An automated routine that could design a descriptively adequate grammar for a natural language would be of considerable value to linguistic theory, since we could then see what a natural grammar is really like, and perhaps study the properties of language.

The idea for inputting semantically annotated sentences to a learner draws in part upon theoretical psycholinguistics; a leading suggestion in cognitive science has been that humans learn language using some sort of “semantic bootstrapping” (Pinker 1984).

2 Type-logical grammar

The non-associative Lambek calculus NL serves as a “base logic” for the kinds of type logical grammars that the present paper applies to. We present it as a system of inference figure schemata among type-reduction statements.

This is also known as a Gentzen sequent calculus. A proof in the system is any combination of instances of the schemata in tree form, with axiom sequents at the top and the proven sequent at the bottom.

The logic has only two operators $/$, \backslash , and formulae (types) are combined into trees using ‘ \diamond ’, rather than the more customary sets.

1 DEFINITION

(Axiom)

$$A \Rightarrow A$$

(\ L)

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

(Cut)

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

(/ R)

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

(/ L)

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

(\ R)

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

A, B, C, \dots stand for type formulae, Δ, Γ, \dots for trees of type formulae that are also called *G-terms*.

$\Gamma[A]$ either means a G-term Γ containing an occurrence of type A somewhere within, or it means a G-term Γ in which A has replaced an occurrence of something else, the identity of which would be clear from a previous use of the notation $\Gamma[\cdot]$.

The symbol \diamond indicates a binary non-associative combination (tree-building) operation.

A *type-logical grammar* is a triple $G = \langle V_G, I_G, R_G \rangle$ comprising a vocabulary V_G of words, a lexical function I_G assigning words to sets of type formulae, and a type logic R_G . The lexicon below leads to the sentence proof underneath. The words labeling the types in the proof allow us to keep track of the sentence we are dealing with.

$$(1) \quad I_G(\text{Susan}) = \alpha_2$$

$$I_G(\text{sings}) = \alpha_2 \setminus s$$

$$(2) \quad \frac{\alpha_2 : \text{Susan} \Rightarrow \alpha_2 \quad s : [\text{Susan}, \text{sings}] \Rightarrow s}{(\alpha_2 : \text{Susan} \diamond \alpha_2 \setminus s : \text{sings}) \Rightarrow s}$$

3 Grammar discovery from semantic composition

Consider a model for human language in which a single type logic based on NL, but perhaps extending its capabilities, is universal, underlying all languages.

In this event, the fact that languages differ is accounted for by their having different vocabularies and different lexical assignments.

Thanks to the nature of type logics, the type formulae are themselves documents of the syntactic behavior that they exhibit in the language, once the logic they are involved in is known.

Since a type logic for sentence inference can be provided in advance as a universal language faculty, the language learning problem comes down to learning the vocabulary and the type formulae assigned to those items.

The learning data consists of *term-labeled strings*, i.e. sentences annotated by typed lambda calculus meaning recipes:

$$(3) \quad ((\mathbf{loves}^{(s \leftarrow \alpha)} \leftarrow \alpha)(\mathbf{Mary}^\alpha)))(\mathbf{John}^\alpha))^s : \langle \mathbf{John}, \mathbf{loves}, \mathbf{Mary} \rangle$$

Alternatively, without subterm types, one has:

$$(4) \quad ((\mathbf{loves}(\mathbf{Mary})))(\mathbf{John}))^s : \langle \mathbf{John}, \mathbf{loves}, \mathbf{Mary} \rangle$$

The meaning recipes to the left of the colon show the basic compositional construction of the sentence meaning in terms of application (and abstraction), and can be used as partial descriptions of type-logical proofs of the sentence via a generalized Curry-Howard homomorphism.

The bold-face items are atomic terms representing the meanings of the corresponding words.

The morphism from semantic terms to syntactic proofs is induced by defining a mapping τ from syntactic types to semantic types:

$$\tau(c) = c'$$

for corresponding primitive types c, c' ;

$$\tau(A/i:B) = \tau(B \setminus_i A) = \tau(A) \leftarrow \tau(B)$$

Then lambda term application is seen to correspond to the slash-left Gentzen rules, and lambda abstraction corresponds to slash-right rules. A lambda term can thus be used as an underspecified proof recipe, called a *homomorphic construction*.

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

1. Given a sample D of unsubtyped term-labeled strings, compute a counterpart sample D' whose terms are subtyped using variable primitives in a most general way.

For example, given two term-labeled strings **input**

$(\text{singsMary})^s : \langle \text{Mary, sings} \rangle$

$(\text{singsSusan})^s : \langle \text{Susan, sings} \rangle$

we provide **output**

$(\text{sings}^{s \leftarrow \alpha_1} \text{Mary}^{\alpha_1})^s : \langle \text{Mary, sings} \rangle$

$(\text{sings}^{s \leftarrow \alpha_2} \text{Susan}^{\alpha_2})^s : \langle \text{Susan, sings} \rangle$

2. Next, we compute the set of general form type-logical lexicons $\text{GFTL}(D')$ which will generate the sample, in each of which distinct variable primitive types will each occur atomically only once. 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 proof recipe, 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.

Continuing the example, the above data tell us:

(a) $(\mathbf{sings}^{s \leftarrow \alpha_1} \mathbf{Mary}^{\alpha_1})^s$ is a *homomorphic construction* of a proof of:

$$\Gamma_1 : [\mathbf{Mary}, \mathbf{sings}] \Rightarrow s$$

(b) $(\mathbf{sings}^{s \leftarrow \alpha_2} \mathbf{Susan}^{\alpha_2})^s$ is a *homomorphic construction* of a proof of:

$$\Gamma_2 : [\mathbf{Susan}, \mathbf{sings}] \Rightarrow s$$

By reading the lambda term like a recipe for proof-building, the two proofs are derived, respectively:

$$(5) \quad \frac{\alpha_1 : \mathbf{Mary} \Rightarrow \alpha_1 \quad s : [\mathbf{Mary}, \mathbf{sings}] \Rightarrow s}{(\alpha_1 : \mathbf{Mary} \diamond \alpha_1 \setminus s : \mathbf{sings}) \Rightarrow s} (\backslash\mathbf{L})$$

$$(6) \quad \frac{\alpha_2 : \mathbf{Susan} \Rightarrow \alpha_2 \quad s : [\mathbf{Susan}, \mathbf{sings}] \Rightarrow s}{(\alpha_2 : \mathbf{Susan} \diamond \alpha_2 \setminus s : \mathbf{sings}) \Rightarrow s} (/ \mathbf{L})$$

GF_{TL} thus finds just one general form lexicon that is consistent with the learning sample:

$$(7) \quad I_G(\text{Mary}) = \alpha_1$$

$$I_G(\text{sings}) = \alpha_1 \setminus s$$

$$I_G(\text{Susan}) = \alpha_2$$

$$I_G(\text{sings}) = \alpha_2 \setminus s$$

3. Finally, we compute all of the *optimal unifications* of each of the lexicons in $GF_{TL}(D')$.

The notion of an optimal unification of a family of type formulae comes from Buszkowski and Penn (1990), and is a substitution (mapping variables to types, standardly) that has the effect of equating all types that have the same form, and thus in some sense equates all word senses that are equivalent as to usage.

The general form lexicon above can be optimally unified, which results in the following lexicon.

(8) Mary α
sings α/s
Susan α

Another example shows the results of applying the OUTF 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).

(9) (sings(**John**))^s : ⟨John, sings⟩

 (((loves(**Mary**)))(**John**))^s : ⟨John, loves, Mary⟩

 (((loves(**a man**)))(**Mary**))^s : ⟨Mary, loves, a, man⟩

 (((sees(**John**)))(**a man**))^s : ⟨a, man, sees, John⟩

(10) (sings(**Mary**))^s : ⟨Mary, sings⟩

 (((loves(**John**)))(**Mary**))^s : ⟨Mary, loves, John⟩

 (((loves(**Mary**)))(**a man**))^s : ⟨a, man, loves, Mary⟩

 (((sees(**a man**)))(**John**))^s : ⟨John, sees, a, man⟩

$$I_G(\text{John}) = \{\alpha_1, \alpha_2, \alpha_9\}$$

$$I_G(\text{sings}) = \alpha_1 \setminus s$$

$$I_G(\text{loves}) = \{(\alpha_2 \setminus s) / \alpha_3, (\alpha_4 \setminus s) / \alpha_5\}$$

$$I_G(\text{Mary}) = \{\alpha_3, \alpha_4\}$$

$$I_G(a) = \{\alpha_5 / \alpha_6, \alpha_7 / \alpha_8\}$$

$$I_G(\text{man}) = \{\alpha_6, \alpha_8\}$$

$$I_G(\text{sees}) = (\alpha_7 \setminus s) / \alpha_9$$

$$I_G(\text{Mary}) = \{\alpha_1, \alpha_2, \alpha_9\}$$

$$I_G(\text{sings}) = \alpha_1 \setminus s$$

$$I_G(\text{loves}) = \{(\alpha_2 \setminus s) / \alpha_3, (\alpha_7 \setminus s) / \alpha_9\}$$

$$I_G(\text{John}) = \{\alpha_3, \alpha_4\}$$

$$I_G(a) = \{\alpha_5 / \alpha_6, \alpha_7 / \alpha_8\}$$

$$I_G(\text{man}) = \{\alpha_6, \alpha_8\}$$

$$I_G(\text{sees}) = (\alpha_4 \setminus s) / \alpha_5$$

These two optimally unify to the same lexicon:

$$(11) \quad I_G(\text{Mary}) = \alpha$$

$$I_G(\text{sings}) = \alpha \backslash s$$

$$I_G(\text{loves}) = (\alpha \backslash s) / \alpha$$

$$I_G(\text{John}) = \alpha$$

$$I_G(\text{a}) = \alpha / \beta$$

$$I_G(\text{man}) = \beta$$

$$I_G(\text{sees}) = (\alpha \backslash s) / \alpha$$

4 Extensions and ramifications

- The above algorithm extends to enrichments of the simple syntactic calculus, which extend the logic with structural rules and unary operators controlling access to them, à la Linear Logic.
- The logics in a large class of such systems could plausibly generate natural languages.
- The lexical type systems of natural languages could then be learned in the above fashion.
- The optimally unified grammars of this kind are learnable in the limit according to the Gold (1967) criterion.

- The term-labeled languages generated by such grammars are *syntactically consistent*, which means that word senses having identical usages automatically have identical syntactic categories, and conversely words having distinct usages automatically have different syntactic categories.
- The above algorithm is proven to always terminate on any finite sample of term-labeled sentences.
- The algorithm is so inefficient, it cannot be run in reasonable time on any sample requiring an enriched logic, so more work on this is sorely needed.

References

- Buszkowski, Wojciech, and Gerald Penn. 1990. Categorical grammars determined from linguistic data by unification. *Studia Logica* 49:431–454.
- Gold, E. M. 1967. Language identification in the limit. *Information and Control* 10:447–474.
- Grimshaw, Jane. 1979. Complement selection and the lexicon. *Linguistic Inquiry* 10:279–326.
- Grimshaw, Jane. 1981. Form, function, and the language acquisition device. In C. L. Baker and J. J. McCarthy (Eds.), *The Logical Problem of Language Acquisition*. Cambridge, MA: MIT Press.
- Lambek, Joachim. 1958. The mathematics of sentence structure. *American Mathematical Monthly* 65:154–170.

- Moortgat, Michael. 1999. Meaningful patterns. In Jelle Gerbrandy, Maarten Marx, Maarten de Rijke, and Yde Venema (Eds.), *JFAK: Essays dedicated to Johan van Benthem on the occasion of his 50th birthday*. Institute for Logic, Language, and Computation, University of Amsterdam. Available on CD-ROM at <http://turing.wins.uva.nl>.
- Morrill, Glyn V. 1994. *Type Logical Grammar: Categorical Logic of Signs*. Dordrecht: Kluwer.
- Pinker, Steven. 1984. *Language Learnability and Language Development*. Cambridge, MA: Harvard University Press.