A gentle introduction to Type Logical Grammar, the Curry-Howard correspondence, and cut-elimination[1]

Chris Barker *UCSD* <barker@ucsd.edu>

This paper aims to give an introduction to Type Logical Grammar (TLG) for the general linguist who may not have a background in logic or proof theory. It is shorter than book-length treatments such as Hepple 1990, Morrill 1994, Carpenter 1996, Jäger 2001, and Moot 2002, and less technical than surveys such as Moortgat 1997.

What does it mean to use logic as a grammatical formalism? The short answer goes like this: what a logical system does is take a list of assumptions and derive some conclusions from those assumptions. For instance, from the assumptions A and $A \rightarrow B$ (read "A implies B"), the conclusion B follows in most logical systems. Analogously, given a sequence of assumptions consisting of an NP *John* of category np and a verb phrase *left* of category $np \rightarrow s$, a Type-Logical grammar might conclude that *John left* as a whole must be of category s—i.e., a well-formed sentence.

Well, using logic to derive sentences is a nice trick, but what makes this approach appealing? I will concentrate below on two of the more popular answers. First, Type-Logical grammars have a particularly attractive style of semantic compositionality. As explained below, thanks to the Curry-Howard correspondence, each syntactic derivation has a definite, natural, automatically-generated semantic interpretation.

Second, in TLG, certain generalizations about complex linguistic behavior that must be stipulated in most theories (e.g., type lifting, function composition, and more) turn out to be quite literally theorems of the basic system.[2]

This paper will briefly describe the logical heritage of TLG, sketch a simple TLG analysis, convey the spirit of the Curry-Howard result, discuss Gentzen's famous cut-elimination result, and motivate multimodal extensions of Lambek's logic.

---

[1]Some of the expository strategies (such as avoiding discussion of the product connective as long as possible) are due to conversations with David Dowty. Thanks to Christopher Dutchyn, Gerhard Jäger, Brian Weatherson, and the Semantics Reading Group at Brown. Version of October 17, 2003.

[2]Some people love the fact that Type-Logical grammars tend to be radically lexicalized, which means roughly that there are no non-lexical rules that mention specific syntactic categories. I personally do not see the appeal of radical lexicalism, but there are many linguists besides TLG practitioners who do.

1. Gentzen

Inspired by Kant's Constructivism, Brouwer and his students invented Intuitionistic logic. A constructivist refuses to claim that A or B is true unless she can either construct a proof of A or else a proof of B; in general, non-constructive proofs (sometimes called "existence proofs") are not allowed. For instance, merely proving that it is impossible to disprove A does not entitle one to conclude that A must be true; rather, A can only be asserted if you have a positive proof of A. Thus Intuitionistic logic is characterized by rejecting the classical law of double negation (which says that from "not not A" conclude "A").

Gentzen invented two ways of formalizing Intuitionistic logic: Natural Deduction and the sequent calculus. Each format has its advantages. Natural Deduction presentation is easier for humans to use, but sequent presentation makes certain technical results easier to prove (as discussed below). Unfortunately, one obstacle to learning about TLG is that authors often, perhaps typically, present their grammars first in Natural Deduction, then again in sequent format (and then sometimes also in tree format!). In this paper, I will stick to Natural Deduction format (using trees to represent derivations).

In addition to inventing the sequent calculus, another crucial element in Gentzen's approach involves dividing inference rules into two types: introduction rules and elimination rules. We shall see how this is useful from a linguistic point of view immediately below; but first we have to consider linear order.

2. Lambek

Traditional Intuitionistic logics like those studied by Gentzen are not suitable for describing natural language. One salient reason is that they are order-insensitive: if assumptions $A \rightarrow B$ and A allow concluding B, then the same assumptions in reverse order also justify concluding B. From a purely logical point of view, this makes complete sense. However, when we are using logic to describe natural language, we do not want the fact that *John* followed by *left* is of category s to justify concluding that *left* followed by *John* is also an s.[3]

Nowadays, there is a rich branch of logic studying such things as order of assumptions. Girard's (1987) linear logic gives rise to a family of such "resource-sensitive" logics, which linguists may have encountered in the form of glue semantics for LFG. But as long ago as 1958, Joachim

[3]As Brian Weatherson points out, it's even worse: in standard Intuitionistic logic, from A and $A \rightarrow B$, it is possible to conclude A—and we certainly don't want to decide that *John left* should count as an *np*.

Lambek proposed an order-sensitive Gentzen-style logic. His main innovation was to divide up order-insensitive implication ("$A \rightarrow B$") into a lefthanded and a righthanded version: $A\backslash B$ is equivalent to $A \rightarrow B$ under the restriction that the expression that is in category A must be to its left; and B/A is also equivalent to $A \rightarrow B$, under the restriction that the category A expression must be to its right.

As some linguist readers will know, Ajdukiewicz in the 1930's created a grammar formalism thought of today as the original Categorial Grammar, and in which category labels such as $A\backslash B$ have a structure and interpretation highly similar to the Lambek categories. Nevertheless, it makes sense to consider Lambek as the true originator for Type-Logical Grammar, because he introduced his connectives in the form of a Gentzen-style logic. In particular, Lambek provided not only the normal elimination (cancellation) rules that characterize Ajdukiewicz's Categorial Grammar, he also provided Gentzen-stye introduction rules, which we will now examine in some detail.

With left and right hand versions cross-cutting elimination and introduction rules, what in traditional logic is a single rule for implication turns into four distinct logical rules in Lambek's system:[4]

```
(1)     Elimination rules:          Introduction rules:
        Combination (cancellation)  Hypothetical reasoning


        A,A\B                       A,X ==> B
        -------\E                   -----------\I
          B                         X ==> A\B


        B/A,A                       X,A ==> B
        -------/E                   -----------/I
          B                         X ==> B/A
```

The intended interpretation of the elimination rule '$\backslash E$' is as follows: if an expression of category A is immediately followed by an expression of category $A\backslash B$, then it is valid to conclude that the complex expression as a whole is of category B. (Analogously for the right elimination rule '/E', differing only in order.)

[4]This presentation of Lambek simplifies his system fairly drastically; in particular, I have left out the product connective ('$\bullet$') that makes $/, \bullet, \backslash$ a residuated triple. Most other presentations of TLG (with some notable exceptions, such as Hepple 1990 and some papers of Dowty) discuss the important but fairly technical topic of residuation.

The elimination rules all by themselves enable some complete sentences to be derived. If *John* is of category *np*, and *left* is of category $np\backslash s$, then we have:

```
(2)   John       left
      np         np\s
      ------------------\E
            s
```

Simple enough. But it is the introduction rules that gives TLG it distinctive flavor. This is where is gets interesting: the introduction rules allow hypothetical reasoning. For instance, the intended interpretation of the introduction rule $\backslash I$ is as follows: if it is possible to prove that when an expression of category A is immediately followed by a sequence of categories X the result is a proof of B, then it follows that the sequence X alone must have category $A\backslash B$.

Given this interpretation of the introduction rule, we can think of $A\backslash B$ as meaning "If there were only an A to my left, I could prove B". But of course this same interpretation applies to the elimination rule: what $\backslash E$ says is precisely that preceding $A\backslash B$ with an A produces a B. Thus the elimination rule and it corresponding introduction rule are two sides of the same coin: together then characterize the meaning of the connective $\backslash$.

Consider again the derivation in (1) in light of the introduction rules. The derivation in (1) proves that when an expression of type *np* is followed by an expression of type $np\backslash s$, the complex expression has type s. By setting $X = np$ and $A = np\backslash s$, the introduction rule $\backslash I$ tells us that X alone (in this case, the NP) must have type $B/A = s/(np\backslash s)$. So Lambek's inference rules tell us that if *John* has category *np*, it must also have category $s/(np\backslash s)$.

This gives rise to a second derivation for *John left*, namely:

```
(3)   John          left
      s/(np\s)      np\s
      -------------------/E
               s
```

This record of the derivation, however, does not make explicit the hypothetical reasoning that allowed us to conclude that *John* was in the category $s/(np\backslash s)$. When using the introduction rules in a derivation, the convention is to explicitly show how the presence of the expression corresponding to the A in the introduction rules leads to the conclusion B; this is done by inserting the appropriate category into the derivation with a numerical label, which is then mentioned at the point at which

the introduction rule that depends on it is used. The full description of the derivation for *John left*, then, is:

```
(4)   John   ----1      left
      np     np\s       np\s
      -----------\E
           s
        ---------/I,1
        s/(np\s)
        ---------------------/E
                  s
```

We say that the expression labelled 1 is a hypothesis. Careful thought about the interpretation of the introduction rules will reveal that the hypothesis retired by an introduction rule must occur at the edge of that expression, either the left edge in the case of $\backslash I$, or the right edge in the case of $/I$.

Motivating hypothetical reasoning: non-constituent coordination

The derivation in (4) is redundant, in the sense that we had already derived the sentence in question in (2) without using any hypothetical reasoning. In order to provide some sense of how hypothetical reasoning can be helpful to solve linguistic problems, I will give a basic analysis of so-called non-constituent coordination (Steedman, Dowty). Usually, coordination conjoins constituents:

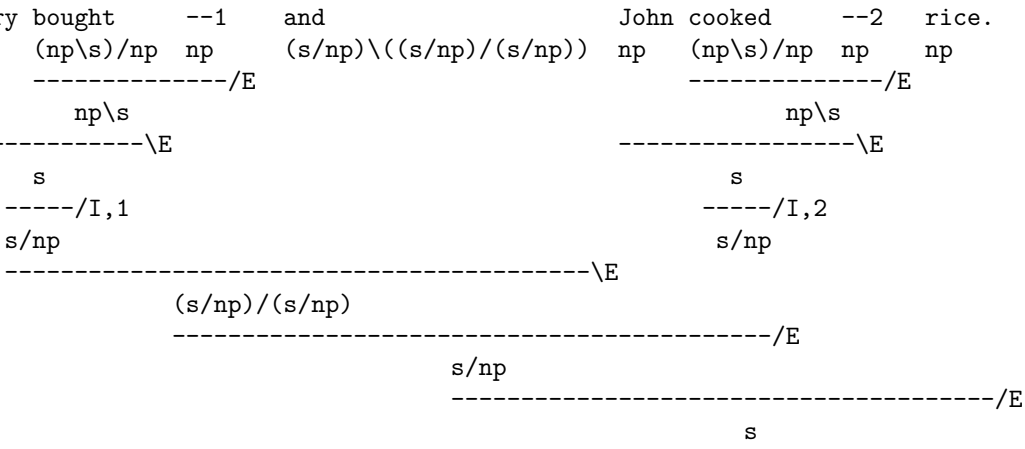
(5) a. [John] and [Mary] left.
    b. John [saw Mary] and [called Tom].
    c. [John left] and [Mary left].

In each case, the bracketed material is a constituent, either an NP, a VP, or an S. But in general, coordination sometimes can involve what at first glance do not seem to be constituents in the normal sense:

(6) a. [Mary bought] and [John cooked] rice.
    b. John ate [rice yesterday] and [beans today].
    c. John gave [a book to Mary] and [a record to Tom].

Here, *Mary bought* does not seem to be a constituent, nor *rice yesterday* or *a book to Mary*. Nevertheless, these coordinate structures are grammatical. In TLG terms, we cannot combine the category of *Mary* (np) with the category of *bought* ($(np\backslash s)/np$) using the elimination rules alone. But with the introduction rules, there is a solution:

(7)



Assume that the conjunction *and* has category $X\backslash(X/X)$ for any category X; here $X = s/np$.

Deriving operators as theorems

We saw above that every expression in category np is also in category $s/(np\backslash s)$. In general, for arbitrary categories A and B, any expression of category A must also be in category $B/(A\backslash B)$. Many linguists stipulate this correspondence as a rule: Partee, Hendriks, Steedman, Jacobson, and others all have a lifting operator; but lifting falls out automatically on the TLG approach merely by contemplating the full nature of what the category A/B means, as characterized by its introduction rule.

TLG also automatically provides a number of theorems besides lifting that other systems must stipulate: argument lowering, function composition, the Geach rule, and more. For instance, in addition to lifting as illustrated in (4), several authors (Steedman, Jacobson) postulate a combinatory rule called the Geach rule, which is a kind of Curried function composition. Here is a derivation of the Geach rule in our simplified Lambek system:[5]

[5]A common misconception among linguists encountering TLG for the first time is that the numerical indicies in the natural deduction format have some theoretical content. In fact, the natural deduction chart merely summarizes a series of application of the four rules given (1), nothing more, nothing less. Thus the numerical indicies are entirely dispensable. This would be a good time, by the way, to mention that I am silently assuming that the connectives in the Lambek grammar presented here are associative, both in the derivation in (7) and in (8); see the end of section 6 for more details.

```
(8)          -----1    ---2
     A/B     B/C       C
             ---------------/E
                    B
     -----------------/E
             A
          --------/I,2
             A/C
           ------------/I,1
           (A/C)/(B/C)
```

In other words, any linguistic analysis that depends on the Geach rule can be reconstructed in TLG without needing to stipulate a new rule, since Geaching falls out automatically from the basic system.

To the extent that many of the operators needed for linguistic analysis come automatically as part of the TLG system, type logical grammars serve to characterize a notion of what makes a natural operator (i.e., any operator that is a theorem of some type-logical system).

3. Automatic meaning: the Curry-Howard correspondence

The Curry-Howard isomorphism maps Intuitionistic logic into a particular version of the simply-typed lambda calculus.[6] In the traditional Curry-Howard correspondence, formulas map onto types, and proofs map onto terms in the lambda calculus. In the context of Type Logical Grammar, formulas correspond to syntactic categories.

Some Curry-Howard correspondences:

(9)

Logic		lambda-calculus		TLG
formulas	==	types	==	syntactic categories
proofs	==	terms (programs)	==	syntactic derivation
			==	semantic composition

Thus a formula such as $(np\backslash s)/np$ corresponds to the syntactic category of a transitive verb. If $(np\backslash s)/np$ doesn't strike you as resembling a formula, remember that $\backslash$ is an order-specific version of the implication arrow $\rightarrow$; if we substitute in the arrow and use letters more commonly

[6] Because we are dealing here with logics that are more restricted than Intuitionistic logic, I will talk about the Curry-Howard correspondence instead of the Curry-Howard isomorphism.

associated with propositions, we get $(p \rightarrow q) \rightarrow p$, which looks much more like a typical formula in the propositional calculus.

It is the fact that the Curry-Howard correspondence maps formulas onto types in the λ-calculus that makes the Lambek calculus a "type logic", which of course is why this type of linguistic analysis is known as Type Logical Grammar (see Jäger 2001:31).

There are two interpretations of proofs in TLG. Focusing on the logical side of the Curry-Howard mapping, proofs constitute a derivation, a demonstration that some sequence of lexical items is a well-formed constituent of some type. On this interpretation, proofs correspond to syntactic derivations.

Alternatively, focusing on the λ-calculus side of the Curry-Howard correspondence, proofs correspond to programs that compute values. On this interpretation, proofs correspond to semantic composition recipes.

Thus in TLG, merely proving that a sentence is syntactically well-formed automatically and completely determines the (non-lexical part of the) meaning of the sentence. That is deeply, wonderfully cool.

So, what exactly is the Curry-Howard correspondence? In our simplified Lambek system, there are only four rules to worry about. As you might expect from the examples above (or from even the most passing acquaintance with any form of categorial grammar), the correspondence maps the formula B/A (and also $A\backslash B$) onto the semantic type B^A (or, as linguists often write, $\langle A, B\rangle$), which is the type of a function from objects of type A into objects of type B. This expresses the sense in which an expression of category B/A semantically is a function expecting an argument of type A and returning a value of type B. For instance, a derivational step corresponding to an elimination rule maps onto an instance of functional application. Thus in the derivation in (2), the meaning of the sentence *John left* turns out to be the function denoted by *left* applied to the argument denoted by *John*: $\mathbf{left}(\mathbf{j})$.

What, then, is the interpretation of hypothetical reasoning? Extending the thinking that revealed the introduction rules as the dual of the elimination rules, what is the dual of functional application?

The answer, of course, is the only other semantic mechanism available in the lambda calculus beside functional application, namely, functional abstraction (i.e., λ-abstraction). In the Natural Deduction derivation-writing style, the hypothesis corresponds to an occurrence of a variable, and the relevant introduction rule corresponds to the lambda form that binds that variable.

For our lifting derivation, then, we have the following mapping:

```
(10) Proof (== Derivation)              Program (== semantic composition)

     John   ----1      left
     np     np\s       np\s            j      P      left
     ------------/E                    --------
          s                              P(j)
       ---------/I,1                    --------
       s/(np\s)                         ^P.P(j)
       ---------------------/E          ------------------
                 s                      [^P.P(j)](left)
```

Since hypothesis 1 in this derivation has category $np\backslash s$, we know that the variable must be over objects that map individuals to truth values. That is why I have chosen P as the variable corresponding to the hypothesis (since linguists traditionally use P as a variable over VP meanings). The elimination rules apply the meaning of the functor category to its argument. The introduction rule abstracts over the hypothesized variable.

Note that after λ-conversion (specifically, β-reduction), the semantic value of the sentence is equivalent to **left**(**j**), the same semantic value we get from the simpler derivation in (1); the next section addresses this equivalence.

4. Cut elimination is λ-reduction.

Many discussions of type-logical grammar mention a technique called "cut elimination". Gentzen first proved a cut-elimination theorem (his *Hauptsatz*, or 'main theorem') for intuitionistic logic; Lambek adapted Gentzen's proof to order-sensitive logics like the one discussed here.

The Hauptsatz has an astonishing variety of important consequences. The one that I will mention here is that it guarantees that if there is any proof based on a certain sequence of formulas, then an equivalent proof that is at least as simple can be found whose length does not exceed a finite bound based on the length of the original sequence. What this means when using a Lambek-style grammar is that it is possible to decide efficiently whether a string is a well-formed sentence.

In order to be able to eliminate cuts, of course, you have to have cuts to begin with. The cut rule is an inference rule that says that it's ok to have sub-proofs (lemmas). Here is what the cut rule looks like for the natural-deduction style Lambek grammar studied here:

```
(11)   X ==> A      Y,A,Z ==> B
       --------------------------Cut
              Y,X,Z ==> B
```

If you have a proof that from the assumptions in X it is possible to derive A (i.e., $X \Rightarrow A$), and that using A it is possible to derive B ($Y, A, Z \Rightarrow B$), then the cut rule says that it is valid to assert that after substituting the assumptions in X for A it is still possible to prove B ($X \Rightarrow B$). Obviously, this is a property that any reasonable logic must have, so virtually every logic has a cut rule, either implicitly or explicitly.

As an example, we can use the cut rule to provide yet a third proof that *John left* is a grammatical sentence. As proved above, $np \Rightarrow s/(np\backslash s)$. By the /E rule, we have $s/(np\backslash s)$, $np\backslash s \Rightarrow s$. Choosing Y empty, we can instantiate the cut rule as follows:

```
(12)    np ==> s/(np\s)    s/(np\s), np\s ==> s
        ---------------------------------------Cut
               np, np\s ==> s
```

In some sense what we are doing is using the theorem that (in general) $A \Rightarrow B/(A\backslash B)$ (what we called "lifting" above) as a lemma. What the cut rule enables us to do is insert a previously proved lemma into a derivation at will, as long as the formula in the conclusion of the lemma matches some formula in the assumptions of the remainder of the derivation.

What the cut elimination theorem says is that any proof that relies on the cut rule can be proved directly without using the cut rule at all. Put another way, adding the cut rule to the logic does not increase the class of theorems that can be proved in the logic. In the official terminology, the cut rule is "admissible". The fact that the cut rule is admissible with respect to the logic above is my justification for not including it.

Eliminating cuts in a natural-deduction logic is trivial: just insert the proof of the lemma ($X \Rightarrow A$) in the position of A in the larger proof ($Y, A, Z \Rightarrow C$). The derivation using cut in (12), then, can be replaced by the equivalent cut-free proof given above in (4).

Because cut-elimination is trivial in a natural-deduction logic, full appreciation of the cut elimination theorem requires considering the sequent calculus. Indeed, the reason Gentzen's 1935 paper presented two equivalent logical systems (natural deduction and the sequent calculus) is precisely because even though the natural deduction format is so much more congenial to human minds, it was only using the sequent calculus that Gentzen could prove a non-trivial version of the cut-elimination theorem.

Gentzen's sequent presentation is provably equivalent to the natural deduction versions as far as the set of theorems that each proves,

but differs in ways that are relevant for cut elimination. I promised that I would use only natural-deduction format, and I won't break that promise. Nevertheless, I will try to convey a sense of one crucial property of the sequent calculus, namely, the sub-formula property, by showing that the natural deduction format does not have this property.

A logical rule has the subformula property if all of the formulas above the line are subformulas of the formulas below the line. For instance, the natural deduction introduction rules have the subformula property, as you can see by examining the $\backslash I$ rule, repeated here:

```
A,X ==> B
-----------\I
X ==> A\B
```

The only items above the line are A, B, and X, each of which appears below the line as well. In addition to A, B and X, the conclusion sequent contains a slash \ that is not part of any of the antecedent formulas, so in fact the antecedents are strictly simpler than the conclusion.

Not surprisingly, the natural deduction introduction rules are identical to the corresponding rules in the sequent presentation (though they are called "right rules" or "rules of proof" when they are part of a sequent system).

The natural deduction elimination rules, in contrast, do not have the subformula property, as you can see by examining the elimination rule $\backslash E$:

```
A,A\B
------\E
B
```

The formula A occurs above the line but not below it. The sequent version of the elimination rules (called "left rules" or "rules of use") look very different; in any case, they have the subformula property.

The reason that the subformula property is relevant here is that it is what guarantees finite termination: since all of the material above the line occurs below it as well, there is a finite number of possible configurations that could have been used as the previous step in the proof. Once each of these alternatives has been tried, the search for a proof is over.

That's all I'm going to say about the sequent calculus. Nevertheless, it is possible to give a very good impression of what cut elimination does indirectly, by exploiting the Curry-Howard correspondence. Not surprisingly, cut-elimination has an equivalent on the lambda side of the Curry-Howard correspondence: λ-reduction.

Consider the Curry-Howard labelling of the derivation in (4) as spelled out in (8): $(\lambda P.P(\mathbf{j}))(\mathbf{left})$. Obviously, this lambda term can be simplified by substituting the argument **left** in place of the occurrence of P in the body of the lambda expression. That is, according to the rules of λ-reduction, $(\lambda P.P(\mathbf{j}))(\mathbf{left})$ reduces to $\mathbf{left}(\mathbf{j})$.

As soon as we notice that the Curry-Howard labelling can be simplified, we know that the proof in (4) can be simplified as well: there must be an equivalent proof with fewer steps—and there is, namely, (2). One other happy consequence of the cut-elimination theorem for the sequent calculus, then, is that any cut-free proof is guaranteed to be as simple as possible (in a certain sense related to the way in which a λ-term that is in normal form and that cannot be further reduced is as simple as possible). That is, the proof in (2) is simpler than the proof in (4); but since the sequent version of the proof in (2) is cut-free, there is no simpler proof. The extra length in (4) comes from the gratuitous lifting of the type of the NP from np to $s/(np\backslash s)$. By hypothesizing $np\backslash s$ when we didn't need to, we have gone out of our way; that is why the cut-elimination theorem is often said to result in proofs "without detours".

Under Curry-Howard, the cut rule corresponds to application of a function to an argument, and cut-elimination corresponds to reducing a functor-argument pair by substituting the argument into the body of the function. Thus Gentzen's Hauptsatz is equivalent to the fact that for any given term in the λ-calculus, a maximally simple equivalent term (i.e., a term in 'normal form') can be found in a finite number of steps (this fact is known as the property of "strong normalization").[7]

Indeed, as you might guess, an equivalent notion of simplification called proof normalization can be defined directly on natural-deduction proofs without converting from sequents or from a Curry-Howard mapping. Gerhard Jäger (personal communication) puts it like this: "if you have a $/E$ directly followed by $/I$ in a natural deduction proof, you can cancel them out against each other, and likewise if $/I$ is directly followed by /E. On the level of lambda-terms, the former corresponds

[7]At this point, you may be wondering about the elimination rules, since they also correspond to functional application. The difference between the elimination rules and the cut rule is that the functional application represented by the elimination rules cannot undergo further reduction. In technical terminology, this is the difference between a redex and an application. That is, even in the simple proof in (2) there is an instance of application in the Curry-Howard labelling: $\mathbf{left}(\mathbf{j})$. This application of the function denoted by *left* to the individual named by *John* corresponds to the one use of the $\backslash E$ elimination rule.

to η-reduction and the latter to β-reduction... The Curry-Howard-correspondence is not just between proofs and terms, it also holds between proof normalization and term normalization."

As a result of the Curry-Howard correspondence, then, there is a deep connection between proof and computation. In the lambda calculus, doing lambda reduction counts as computation. Since the effect of cut elimination corresponds to λ-reduction under the Curry-Howard correspondence, eliminating cuts is the same thing as computing. Lambek's cut elimination theorem says that the Lambek calculus is capable of expressing only computations that terminate in a finite number of steps.

5. Modes and the product connective

Lambek divided implication into forward slash and backward slash to capture the fact that natural language often cares whether an argument is to the right or to the left of its functor (i.e., *John left* but not **left John*). But there are many situations in which natural languages do not care about linear order. As a result, a linguistically realistic grammar must provide controlled access both to order-sensitive and to order-insensitive resources. In TLG, each type of combination regime is called a mode, so a grammar that mixes an order-sensitive mode with an order-insensitive mode is called a multi-modal type logical grammar.[8] As a simple example motivating a multi-modal analysis, let's consider a so-called scrambling language such as Serbo-Croatian.

Scrambling in Serbo-Croatian
Serbo-Croatian allows all six possible orderings for a sentence consisting of a subject, and object, and a predicate.

(13) a. Marko=je ispraznio džepove.
Marko=AUX emptied pockets
'Marko emptied [his] pockets.'

b. Marko=je džepove ispraznio.
c. džepove=je Marko ispraznio.
d. džepove=je ispraznio Marko.
e. ispraznio=je Marko džepove.
f. ispraznio=je džepove Marko.

[8]Incidentally, there is a deep mathematical connection between type-logical modes of combination and modal logic. See, e.g., the discussion in Moortgat 1997.

Here we see all six possible major word orders, all of which are grammatical (given appropriate variation in discourse context).[9]

Yet many (probably all) scrambling languages also have elements that do care about linear order. In Serbo-Croatian, for example, coordinating conjunctions always come in between the expressions they coordinate:

(14) a. ispraznli=su džepove Marko i Jelena.
emptied=AUX pockets Marko and Jelena.
'Marko and Jelena emptied [their] pockets.'

b. *ispraznli=su džepove Marko Jelena i.
c. *ispraznli=su džepove i Marko Jelena.

Any attempt to re-arrange the order of *Marko*, *i*, and *Jelena* results in ungrammaticality unless the conjunction *i* occurs in between the two conjuncts.

Clearly, languages mix order sensitive and order-insensitive elements within a single sentence, and so we need to have fine-grained control over when a grammatical inference requires order-sensitivity and when it does not.

TLG therefore allows for several modes of logical inference (equivalently, several modes of syntactic combination). The default mode, corresponding to the standard Lambek connectives / and \, is order-sensitive; but we can define a separate connective $\rightarrow$ (in honor of traditional implication) for which word-order variations are allowed. (Two modes is not the limit: more complex analyses can have three or more modes.)

Nor is it quite sufficient to declare that $\rightarrow$ merely doesn't care whether its next argument is to the left or to the right, since that will not explain how to get OSV order. The problem is that the verb must combine first with its direct object, which is not adjacent to it, and only then with the subject.

This is where things get a little bit tricky. Following Lambek, in TLG binary connectives always come in sets of three: a left-leaning version (think of \), a right-leaning version (/), and a middle neutral version (in Lambek's grammar, $\bullet$). That means for our scrambling mode, we will have $\leftarrow$, $\rightarrow$, and $\bullet_s$. In the analysis of Serbo-Croatian, we can ignore $\leftarrow$ and concentrate on $\rightarrow$, but will we need to make use of $\bullet_s$ in order to state the rules that govern scrambling.

[9]Ignore the auxiliary clitic *=je*, which attaches to the first constituent in the sentence, whatever that happens to be.

Rather than trying to explain the logical nature of the product connective (let alone its Curry-Howard labelling), I will provide a less than fully general description of the role the product connective plays in this specific analysis, and encourage anyone who is interested to consult the works in the bibliography for a more complete exposition.

Imagine that the product connective is a placeholder. You are free to posit either type of product connective (either $\bullet$, the default connective, or $\bullet_s$, the scrambling connective) between any two categories during a derivation; but once you have chosen a connective, that amounts to a promise that the only elimination rule that will ever be used to bridge the gap marked by the product must come from the same mode as the chosen product connective.

(15)

```
Valid:                          Invalid:

np * np\s                       np  *s  np\s
----------\E                    --------------\E
   s                                 s


np *s np-->s                    np  *  np-->s
------------  -->E              -------------  -->E
     s                               s
```

The idea is that $\bullet$ must eventually turn into either / or \, and $\bullet_s$ must eventually turn into either $\leftarrow$ or $\rightarrow$. The reason that the upper right proof is invalid is that we chose the scrambling mode ($\bullet_s$) above the line, but then applied $\backslash E$, which is from the default mode; similarly, in the lower right derivation, we chose the default mode ($\bullet$) above the line, but then attempted to use an elimination rule ($\rightarrow E$) that belongs to the scrambling mode.

So far as we have said, $\leftarrow$ behaves just like \, and $\rightarrow$ behaves just like /. In order to allow scrambling, we need to explicitly permit it by means of the following postulate:

(16) $A \bullet_s B \vdash B \bullet_s A$ SCRAM

What this postulate says is that you can reverse the order of two adjacent expressions A and B, as long as you promise that the only elimination rule you will every apply across the gap marked $\bullet_s$ is a scrambling-mode elimination rule (i.e., $\rightarrow E$). I've named this postulate SCRAM for 'scrambling', which is why I've used a subscript 's' to indicate the non-default mode.

We can now provide the following toy lexicon for Serbo-Croatian:

(13)

Marko	$np.nom$	Marko
Jelena	$np.nom$	Jelena
dzepove	$np.acc$	pockets
i	$np.nom\backslash(np.nom/np.nom)$	and
isprazniti	$np.acc \rightarrow (np.nom \rightarrow s)$	emptied

This gives rise to the following derivation for the difficult case, namely, OSV:

```
(17)  dzepoveje        Marko              ispraznio
      np.acc      *s   np.nom             (np.acc-->(np.nom-->s))
      ------------------------SCRAM
      np.nom      *s   np.acc
                       ---------------------------------------- -->E
                                    np.nom-->s
      ------------------------------------------ -->E
                        s
```

Because we chose the scrambling mode for the gap inbetween *džepoveje* and *Marko* (witness the presence of the $\bullet_s$), we commit ourselves to using $\rightarrow E$ to combine the leftmost constituent with the remainder when the time comes. But in this derivation, only the scrambling mode is used, so there is no doubt about satisfying the requirements imposed by choosing $\bullet_s$ from the scrambling mode.

Consider next an OSV derivation involving a coordinated subject:

```
(18)

       Marko     i                              Jelena   ispraznio
       np.nom    np.nom\(np.nom/np.nom)         np.nom   emptied
       ---------------------------------\E              np.acc-->(np.nom-->s)
                    np.nom/np.nom
dzepoveje           ------------------------------/E
np.acc       *s                  np.nom
-------------------------------------------------SCRAM
np.nom       *s                  np.acc
                                 ------------------------------------ -->E
                                             np.nom --> s
-------------------------------------------------------- -->E
                        s
```

The conjuncts combine with the coordinate conjunction by means of default-mode slashes, which care about order, but the verb combines with its arguments via the scrambling-mode arrow.

If we try to apply the scrambling postulate inappropriately, the derivation cannot be completed. For instance, consider trying to prove that *i Marko Jelena* is a (nominative) NP:

(19)

```
i                                  Marko          Jelena
np.nom\(np.nom/np.nom)  *s  np.nom                np.nom
-----------------------------------SCRAM
np.nom  *s  np.nom\(np.nom/np.nom)
-----------------------------------\E
         np.nom/np.nom
         ---------------------------------------/E
                        np.nom
```

We can try scrambling the conjunction back into its natural position, but that creates a fatal problem in the location marked with $\bullet_s$: by applying the scrambling postulate, we declare that the only connective that can appear between *i* and *Marko* must be a scrambling-mode connective. But in the very next step in the derivation, we used $\backslash E$, from the default mode, which is illegal.

The net result is that when lexical types use slashes, they can count on their arguments occurring adjacent on the specified side; and when a lexical type uses the arrow, the relevant argument can scramble freely over other arrow arguments. Thus conjuncts must appear rigidly on either side of the coordinating conjunction, but the subject, the object, and the verb can swap places with each other.

The derivation just given glosses over another important detail: I am assuming that both modes are associative. To be explicit, for all of the desired derivations to go through, it is necessary to add the following postulate:

(20) $\quad (A \bullet_s B) \bullet_s C \vdash A \bullet_s (B \bullet_s C)$

For similar reasons, the derivations given earlier in (7) and (9) only go through if we assume that the default mode is associative as well.

The Serbo-Croatian example provides a case in which different word orders are plainly visible. But multi-modal techniques are also needed to provide analyses of phenomena such as quantifier scope displacement and relativization even in (relatively) fixed-word languages such as English.

A short bibliography

Please suggest additions!

Areces, Carlos and Raffaella Bernardi, 200?. Analyzing the core of categorial grammar. Ms, available on the web. The goal of this paper is to elucidate what it means to be a "residuated triple",

and extends this notion to explain how unary modalities work. Therefore is not a basic topic, but I found it helpful at a certain point.

[van Benthem, 1986. *Essays in Logical Semantics*. This book has inspired a number of people, though I haven't read it yet. Explores the consequences of the Curry-Howard result for linguistics.]

Carpenter, Bob, 1996. *Type Logical Semantics*. MIT Press. Introduces the general linguist to TLG. Fairly lengthy, and indispensable for anyone interested in TLG. It has been used as a first textbook in semantics, but I suspect it will be of more use to someone who already knows the basics.

Jäger, Gerhard, 2001. *Anaphora and Type Logical Grammar*. The first chapter is a fairly gentle introduction from first principles with minimal assumptions, and that situates TLG with respect to other related approaches, especially combinatory categorial grammar. See especially chapter 1, pp. 1–53. Does not treat multimodal TLG, but goes much further that this paper on many topics.

Hepple, Mark, 1990. *The Grammar and Processing of Order and Dependency: A Categorial Approach*. PhD dissertation (unpublished), University of Edinburgh, Centre for Cognitive Science, Sept 1990. Respect and concern for the linguistic aspects comes through especially clearly.

Lambek, Joachim, 1958. The mathematics of sentence structure. *American Mathematical Monthly*, **65**: 154–170. The Ur-text for TLG. Charmingly antique. Available in PDF form directly from the journal.

Moortgat, Michael, 1997. Categorial type logics. In Johan van Benthem and Alice ter Meulen, eds., *Handbook of Logic and Language*, chapter 2, pp. 93–178, Elsevier, MIT Press. Dense like pemican, highly mathematical, but lucid. Indispensable.

Moot, Richard (?interface by Dick Oehrle), 2002. Grail on the web. `http://131.211.190.177/grail/`. Allows web access to a parser. You can design grammars, add lexical items, and parse sentences. The results are returned in a LaTeX-typeset PDF document. [See also Moot's 2002 dissertation, *Proof nets for linguistic analysis*, giving some of the theory behind the parser.] An excellent way to get a feel for actually building an analysis in multimodal TLG– fun, and an impressive bit of engineering.

Morrill, Glyn, 1994. *Type Logical Grammar*. Kluwer, Dordrecht. A standard presentation of TLG; highly sophisticated, and fairly difficult. Full of important results.