Prefab Glue, v2

Avery D Andrews
ANU, August 2007
light revisions, June 2008

Here I will develop a formulation of LFG's 'glue semantics' that I hope might be more accessible to syntacticians than the standard ones are. Before starting out, I want to emphasize the point that the 'intuitionistic implicational linear logic' (IILL) that glue is based on is a mathematical system that can be presented in many superficially different but essentially equivalent ways; the motivation for the present version is accessibility to people with certain kinds of backgrounds, rather than an attempt to make any kind of empirical claim.

The basic idea of this approach is to treat meaning-constructors as introducing prefab pieces of logical forms, which are then assembled by means of some simple rules. Technically, this is a version of 'proof nets', the format for IILL proofs that was devised by Girard (the inventor of linear logic) as the preferred format for proofs, for those logics where they work well. But the beginner is not supposed to have to know anything about this in order to master basic use of the system. We will however have to spend some time on the standard format of meaning-constructors, and its relationship to the present approach.

The first two sections provide a basic account of glue for people with a limited formal background; the later sections are more demanding, and may require additional reading.

1 Types and Formulas

We start with some material about the relationship between 'semantic types' and formulas, such as would be covered in a standard introduction to formal semantics. In this tradition, semantic types may be thought of as ontological categories that limit the ways in which concepts can coherently be combined.[1] In linguistics, it is common to recognize at least two basic types, entities, which we'll symbolize as e, and 'propositions', which we'll symbolize as p (t is the more common choice; here we follow Pollard (to appear)). There might be many more types (plausible additions based on recent work in semantics would include *ev* for events, *loc* for locations, *pth* for paths, and *deg* for degrees of magnitude.), or only one (Partee 2006), which is effectively none. But for slightly technical reasons, which we'll consider later, LFG glue seems to need at least the distinction between e and p, so we'll assume these two.

From the basic types, we can construct an infinite variety of additional types by means of 'type-constructors', the most important of which is the 'implicational', which combines two types a and b to produce the type $a \rightarrow b$.[2] This is the type of something

[1] See Casadio (1988) for a discussion of the historical background of type theory.

[2] In systems where there is only one type-constructor, this is often omitted, and $<a, b>$ is written

which, if you combine it with something of type a, then you get something of type b (hence the name, 'implicational'). A very simple example is the type of intransitive verbs, such as *yell* or *stumble*. These, if combined with something designating an entity (*Bill*, *the pony*) produce a proposition, so they are of type $e \rightarrow p$. Another simple one is negation: since a negative turns a proposition into another proposition (true where the first proposition is false, false where the first proposition is true), its type will be $p \rightarrow p$.

What about transitive verbs, such as *likes*? Here we use an idea developed independently by the logicians M. Schönfinkel and H.B. Curry (Schönfinkel first), and introduced into linguistics by Richard Montague (following Lambek (1965) and various other sources), that in a sentence such as *Bert likes Ernie*, one can regard the traditional 'predicate' or contemporary 'VP' *likes Ernie* as being of type $e \rightarrow p$ (this is old, maybe even Aristotle — traditional grammar at any rate), and therefore (the new bit) *likes* as something that takes something of type e as argument and produces as output something of type $e \rightarrow p$. That is, its type is $e \rightarrow (e \rightarrow p)$. Similarly, a verb such as *tell* in *John told Mary that she was sick* would be of type $p \rightarrow (e \rightarrow (e \rightarrow p))$.

The parenthesis-nests associated with multi-argument verbs can get confusing, so people follow a convention of omitting rightmost pairs of parentheses, so that the two types above become $e \rightarrow e \rightarrow p$ and $p \rightarrow e \rightarrow e \rightarrow p$. Another issue is that the exact order of the arguments is 'logically arbitrary'. That is, there's no reason in terms of pure formal semantics why we should supply the p argument of *tell* first, and then the others. However a long tradition of work on the 'thematic role hierarchy' in linguistics provides evidence for a non-arbitrary arrangement of the arguments, so that the Theme (thing conveyed) would be innermost, then Recipient, and finally Agent (see Marantz (1984) for arguments that 'order of application' is linguistically significant, although it might perhaps be better regarded as a hierarchy of tightnes of structural bonding, 'earlier application' being tighter bonding).

One of the things the type system can be made to do is to control the syntax of a simple artificial language for describing the application of (typed) functions to their arguments. Such a language has at least one construction, called '(function) application', in which a function is applied to an argument. The most popular practical syntax for application is to put the function first, then the argument, in parentheses, e.g. $f(a)$, *Likes*(*Ernie*)(*Bert*), etc.

Beginners might appreciate a walkthrough of the later expression. Since *Likes* is of type $e \rightarrow e \rightarrow p$, it is a function which applies to something of type e (to which we suppose *Ernie* belongs), so we write *Likes*(*Ernie*). But this is itself a function of type $e \rightarrow p$, so we can put it in front of *Bert* in parentheses to get *Likes*(*Ernie*)(*Bert*) of type p, a 'logical form' of *Bert likes Ernie*.

The syntax rule that's being used here can be stated like this:

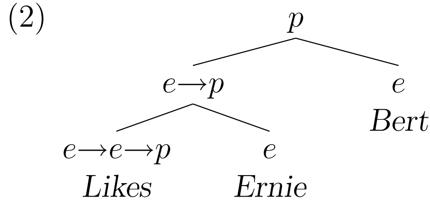
---

rather than $a \rightarrow b$.

(1) If f is an expression of type $A{\rightarrow}B$, and a is an expression of type A, then $f(a)$ is an expression of type B (symbols for types represented in caps to avoid confusion with symbols for expressions).

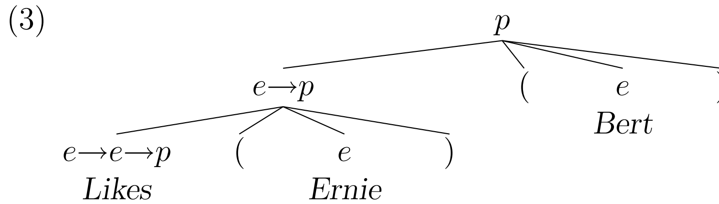
This introduces another way of thinking about types, as grammatical categories for an 'ideal language' which is supposed to closely follow the structure of concepts. It is obviously difficult to distinguish between an ontological system (classification of real things) and a grammar for a language that's supposed to represent the structure of these things, and for our purposes, there is no point in trying to do this; we can remain happily unclear about which of these things we're actually trying to do. With this new 'ideal language' conception in mind, rule (1) lets us produce structure trees in an obvious manner:

(2)



There isn't any real point in including the parentheses in these trees, since the tree-structure itself conveys the relevant information, but if we wanted to do this anyway, the result would be:

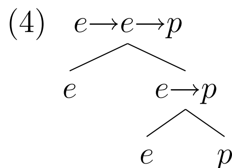
(3)



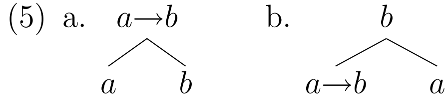
Although we have phrase-structure-like trees for the expressions produced by (1), the rule is not a context-free grammar rule, since it applies to an infinite number of grammatical categories, combining $e{\rightarrow}e{\rightarrow}p$ with e to produce $e{\rightarrow}p$, as well as e with the latter to produce p, etc., etc.

Semantic types also have an obvious tree-structure; for example the tree-form of the type of *Likes* would be:

(4)



The relationship between these is that a type-tree piece of the form (a) below corresponds to a structure-tree piece of type (b):

(5) a. 

Two applications of this reorganization will derive (2) from (4).

This reorganization will play an important role in our handling of meaning-constructors.

A final point is that the ideal language view of types motivates the notion of the 'logical form' of a sentence, which is supposed to be some kind of representation that reflects the structure of the concepts it involves more faithfully than the overt grammatical form. Although this term is widely used, it doesn't have any real definition; rather different linguistic traditions use the term in distinct but related ways. Here, we will use it to refer to a level of representation that indicates in an especially uniform and simple way how meanings are to be combined, and would therefore serve as a reasonable basis for providing a formal definition of entailment and other meaning-based properties and relations.

So there is a first introduction to the some of the basics of type theory. The above can be made more complicated by adding more basic types, and more type-constructors, one of which 'product', will be introduced eventually. And in various other ways. But now we move on to say more about how these things are connected to lexical items, so that they can interact with grammar.

2 Meaning Constructors

In LFG glue, the meanings of words and perhaps certain constructions (Asudeh and Crouch 2002) are presented as 'meaning constructors'. These consist of two components, a 'meaning-side', normally written on the left, and a 'glue-side', normally written on the right, separated by a colon. Here is an example, in uninstantiated form:

(6) $\mathit{Sneeze} : (\uparrow \mathrm{SUBJ})_e \rightarrow \uparrow_p$

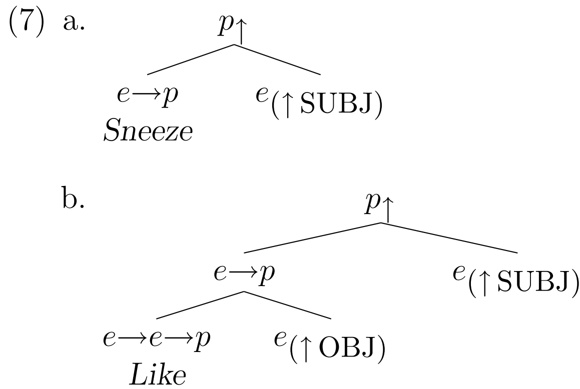
Here the meaning-side is *Sneeze*, which can be regarded as either an abbreviation or promissory note for a specification of the actual meaning of the word 'sneeze' (which, as mentioned above, we can regard as a function that converts an entity-meaning into a proposition-meaning). It is sometimes useful to notate the type of the meaning-side as a superscript: $\mathit{Sneeze}^{e\rightarrow p}$, but we won't normally do this here, since the type is specified in the other component of the constructor, the 'glue side'.

This appears after the colon; in addition to specifying the semantic type of the meaning, it also specifies how the grammatical structure constrains the ways in which the meaning can be combined with the meanings of the other words in the sentence. It is conventionally regarded as an expression in some kind of linear logic, but can equally well be regarded as being a semantic type as discussed above, with some syntactic information added. On the basis of the former interpretation, the 'linear implication'

symbol $\multimap$ appears in most of the literature instead of the $\rightarrow$ found in (6) above.[3] Other differences between (6) and the standard notations will be discussed below.

In the notation of (6), it is assumed that the semantic type information on the glue-side specifies the semantic type of the meaning-side in the obvious way, so that this is $e \rightarrow p$. There is a tradition of using uninformative lambda-abstractions on the meaning-side,[4] so that that of (6) would often be represented as something like $\lambda x.Sneeze(x)$, but I don't see any point in doing this in cases where the meaning-side doesn't actually say anything substantive about the meaning.

One of the basic ideas of prefab glue is to convert the meaning-constructors from the type-tree format to the structure-tree format before assembling them, so that assembly is a matter of putting together pieces of logical forms into a complete structure. In light of the previous section, the prefab forms of the meaning-constructors for *Sneeze* and *Like* would be:



We'll discuss the details of this transmogrification of the meaning-constructors later; what we'll do now is discuss certain aspects of format of (7).

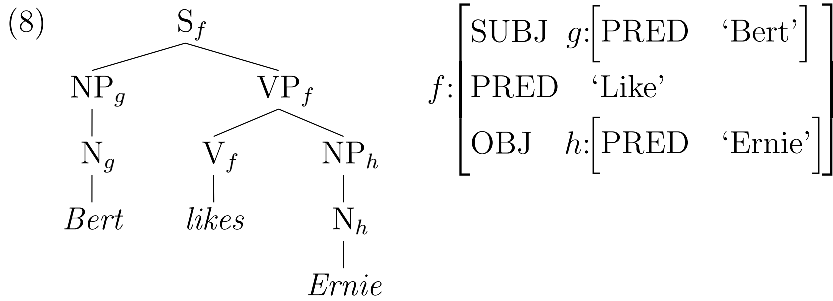
These meaning-constructors in structure-tree format are binary-branching trees, where every node has a semantic type, the leftmost terminals have a meaning, and all nodes of a basic (not implicational) type have an f-structure designator in the uninstantiated form of the constructor, which will become a specification of an f-structure correspondent in the completed structure. Soon, an additional attribute of 'polarity' will be added, which is important for controlling assembly.

But first we'll consider how instantiation works. For a c-structure such as that of, say, *Bert likes Ernie*, the usual workings of LFG produce a c-structure-f-structure pair like this, where the ϕ correspondence from c-structure to f-structure is indicated by co-labelling:

[3] But $\rightarrow$ is used in Lev (2007).

[4] More precisely, less than fully informative, since the lambda-expression does indicate that the meaning-side is of an implicational type, although not which one.

(8)



Meaning-constructors are interpreted as pieces of structure in an additional structural level, which I'll call 'glue structure', although a more descriptive name might be 'semantic compositional structure'. Glue-structure can be regarded in various ways:

(9) a. A binary-branching tree that shows intuitively how the meanings of the words (and perhaps some constructions) in the sentence are to be assembled.

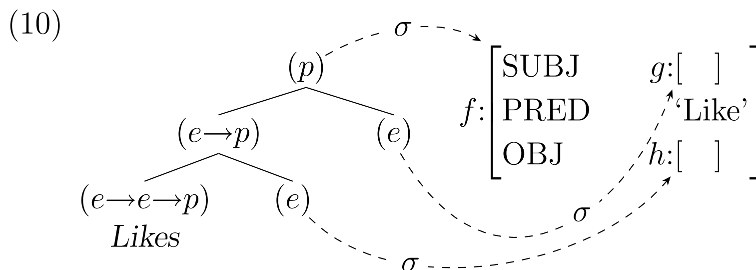
b. A 'linear lambda-term' showing the same thing in a formal way.

c. A 'linear logic proof' in 'Natural Deduction tree format' showing the same thing in a superficially different but actually identical way.

The observation that the latter two conceptions are equivalent is one aspect of the famous 'Curry-Howard Isomorphism'.

A meaming-constructor in the format of (7) specifies a piece of glue-structure, in much the same way as a c-structure rule specifies a piece of c-structure, together with a correspondence to f-structure. The constructor (7b) would be introduced under the V-node of (8), and its functional designators would be interpreted in the usual way (constructively) as referring to pieces of the f-structure, referred to via the ϕ-correspondence from c-structure. So, in (7b), $\uparrow$ means 'the ϕ-correspondent of the V', that is, f, while ($\uparrow$ SUBJ) means 'the value of SUBJ in f', that is, g. We can therefore convert (7b) to instantiated form by replacing the designators with f-structure labels, or, in a more graphic notation, links:

(10)



We essentially follow Kaplan (1987) in using σ as the name of the glue-structure-to-f-structure correspondence, although Kaplan had the opposite directionality (incorrectly, it is claimed here). This use of σ is fundamentally different from what is normally found in the glue literature.

3 Assembly

Assembly divides into two levels of difficulty: easy, when all the arguments are of basic semantic type such as e or p, and hard, when one is of non-basic type, such as property ($e{\rightarrow}p$). Both depend on an attribute of 'polarity', which was devised by the Polish logician Jaśkowski (1963), for reasons that linguists probably don't need to know anything about.

Here are the polarity rules, formulated to work in structure-tree format:

(11) Polarity Rules:

a. The polarities are 'positive' and 'negative', each considered the opposite of the other.

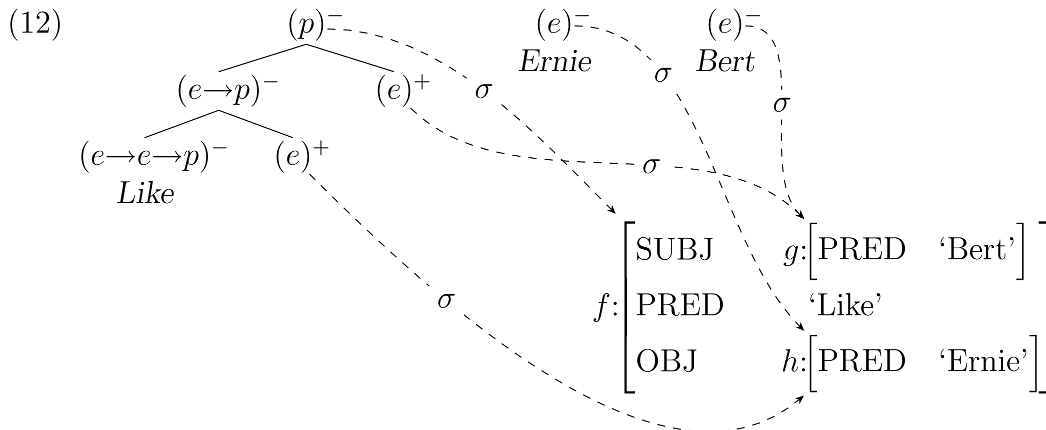
b. Assign negative polarity to each node with a meaning (which will also be a leftmost terminal).

c. Assign the polarity of its left daughter to a branching node

d. Assign the opposite polarity of its mother and left daughter to the right-daughter of a branching node.

Adding constructors for the subject and object of our sentence, the glue-structure pieces and f-structure become:

(12)



Note that as a result of the polarity assignments, the nodes representing the final resultant meanings produced by the constructors have negative polarity, while the arguments, or intuitive inputs, have positive. This seems perverse for meaning-assembly, but makes good sense in the logical tradition from which the ideas emerged. It is probably not worth changing it in linguistics.[5]

We are now ready to formulate the assembly rules, which come in two groups, first, some basic conditions for adding to the structure things called 'axiom links', which

[5]Although Andrews (2004, 2007) did reverse them, possibly following Gupta and Lamping (1998).

can be thought of as lines for plugging outputs into inputs, and then, a more subtle condition called the 'Correctness Criterion', which is needed for ruling out various kinds of bad configurations produced by the basic rules.

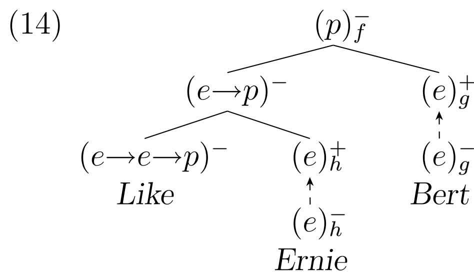
These are:

(13) Connect negative to positive nodes with axiom links subject to these conditions:

1. The linked nodes are of atomic type.
2. The linking is in non-overlapping pairs with one negative left over.
3. The leftover is of type p, and σ-corresponds to the entire f-structure.
4. The semantic types of the linked nodes match.
5. The σ-correspondents of the linked nodes match.

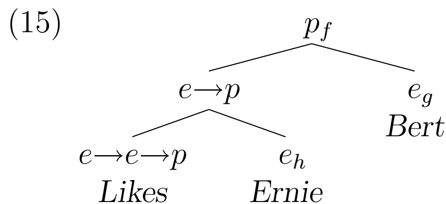
Following these rules, there is only one way to combine the constructors of (13), where we represent the axiom-links with dashed arrows oriented from negative to positive, and retire the σ-arrows in favor of f-structure labels:

(14)



(14) bears an obviously close relationship to the structure tree (2), and this relationship becomes identity if we 'contract' the axiom links, that is, merge axiom-linked nodes into one:

(15)



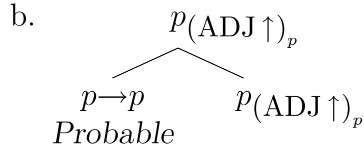
The polarities are erased because they are no longer consistent. This step of axiom-contraction is obviously reversible, so the contracted and uncontracted forms of the structure can be regarded as essentially equivalent.

Unfortunately, however, the rules of (13) are not quite enough. Consider the constructor for *probably*, given in type-expression format in (a), structure-tree-piece format in (b):[6]

[6]Although the designator (ADJ $\uparrow$) is in principle functionally uncertain, this uncertainty will not arise in practice unless adjuncts are multi-attached, which, to my knowledge, doesn't happen.

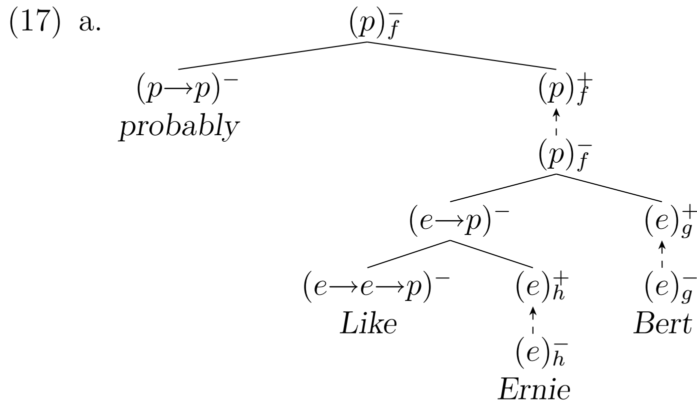
(16) a. *Probable* : $(\text{ADJ}\uparrow)_p \rightarrow (\text{ADJ}\uparrow)_p$

b.

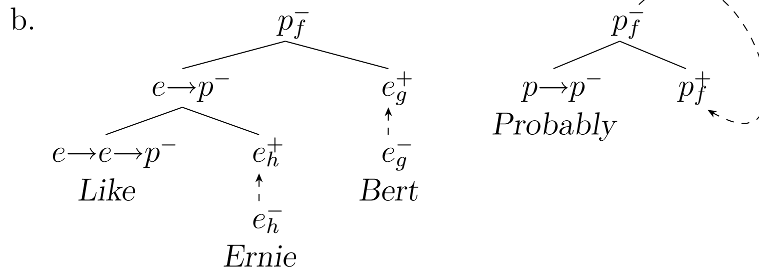


In the structure of a sentence such as *Bert probably likes Ernie*, the ↑ in this constructor would be instantiated to f, and there would then be two ways to assemble the constructors, the sensible way (a), and the perverse way (b):

(17) a.



b.



First note that the sensible (a) shows how reversing the directionality of σ resolves an issue that has been causing trouble for semantics in LFG for quite some time.

It is a property of sentence-adverbials and various other constructions that they seem to involve two levels of semantic structure associated with one level of f-structure, which cannot be described with a function going from the latter to the former. This has, I think, resulted in a certain amount of unclarity in glue about how 'meaning assignments' work. In the 'old glue' format, for example, there were what appeared to be non-monotonic, destructive reassignments of meaning, such as this:

(18) $\forall X((\text{ADJ}\uparrow) \rightsquigarrow_p X \multimap (\text{ADJ}\uparrow) \rightsquigarrow_p \mathit{Probable}(X))$

The symbol $\rightsquigarrow_p$ was read 'assigns a meaning of type p'; this constructor works by taking whatever is currently assigned to (ADJ $\uparrow$) as meaning of type p, and replacing it by that thing embedded as argument of *Probable*. By reversing the direction of σ, we eliminate this source of apparent non-monotonicity (I take no position on whether or not it is real non-monotonicity), and better assimilate glue to the correspondence architecture of LFG.

Now, turning to the non-sensible (b), a reasonable way to exclude it would be to require that the glue structure be connected (that is, between any two nodes, there is a path of links), but to motivate the final formulation we need to consider an additional configuration, positive-polarity implications. These arise from predicates that take 'properties', that is, expressions of type $e{\rightarrow}p$, as their arguments. One classic example is quantified NPs, such as *every dog*, which, under the generalized quantifier analysis of Barwise and Cooper (1981), will be of type $(e{\rightarrow}p){\rightarrow}p$. To remind non-formal semanticists of the motivation for this, note that we can regard *every dog* as something that is true of a property such as *barks* (type $e{\rightarrow}p$) under some circumstances (every dog is indeed barking), and false under others (there is some dog that is not barking). So, taking a property to a proposition is what the type $(e{\rightarrow}p){\rightarrow}p$ does.

From this semantic type, the reorganization principle (5) will produce:

(19)
$(p)^-$
$((e{\rightarrow}p){\rightarrow}p)^-$ $(e{\rightarrow}p)^+$
Everybody

If we blindly applied (5) to the positive implication, we wouldn't get a very useful result.

To get an idea of what we want, consider a slightly more complex logical form in which a quantifier might be used:[7]

(20) $\mathit{Everybody}(\lambda x.\mathit{Like}(x)(\mathit{Bert}))$

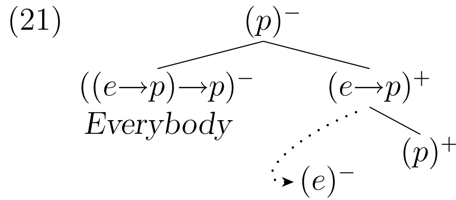
This is supposed to say that the property 'Bert likes them' applies to everybody.

This suggests that we want positive implications to be treated as lambda-abstractions, with the antecedent pointing to the bound variable, the consequent to the formula abstracted over.

We will represent this situation by introducing a new type of node, with one regular daughter, constituting the formula abstracted over, and one 'pseudo-daughter', constituting the variable-binding, and represented by a curved, dotted arrow:

[7] Here we use logical forms that are formulated strictly in terms of function-argument structure. We can get more conventional-looking results such as $\mathit{Every}(x, \mathit{Person}(x), \mathit{Like}(\mathit{Bert}, x))$ via β-reduction (discussed briefly below, or any any introduction to formal semantics), with meaning-sides such as $\lambda P.\mathit{Every}(x, \mathit{Person}(x), P(x))$.

(21)



The negative polarity of the pseudo-daughter means that it needs to plug into a positive of matching type and σ-correspondent, and the positive polarity of the daughter means that something must plug into it, as one would expect of a variable-binder and a place where a formula is supposed to be inserted.

However, we have strayed somewhat from the path of coherent exposition, because we originally introduced polarity as a decoration on trees, and now we're making tree-production dependent on polarity. To get straightened out again, we need to formulate rules for building trees with polarity from glue-sides (which are semantic types with σ-corresondence information added).

We start by making a negative node with the meaning-side as meaning and glue-side as type formula, and then projecting the structure-tree as follows (it doesn't matter whether instantiated or not):

(22) a. Start: Create a node with negative polarity, the meaning-side of the constructor as its meaning, and the semantic type of the constructor as its semantic type.

b. Negative Implication: If the semantic type of a negative node n is an implication, create a new node m with negative polarity, the consequent of the implication as its semantic type, and set n as the left-daughter of m. Then create another new node r with positive polarity, and the antecedent of the semantic type of n as its semantic type, and set this as the right-daughter of m (neither mother nor right-daughter get a meaning).

c. Positive Implication: If the semantic type of a positive node p is an implication, then create a new negative node l as its 'pseudo daughter' (connected to p in the graphic representation by a curved, dotted line), with antecedent of semantic type of p as its semantic type, and create another new positive node r as its (true) daughter, with the consequent of semantic type of p as its semantic type (neither left- nor right-daughter get a meaning).

d. σ-correspondence: If a node n is of atomic semantic type, it is σ-linked to the f-structure whose label appears in the glue-side literal that it represents.

It should be easy to see how this sytem of rules applies to our examples so far; some more complex cases will be discussed later.

For a treatment of quantifiers, we also need to deal with the grammatical information, most problematically, the treatment of scope. The standard approach in glue has

been to use linear universal quantification to in effect say that the two p-nodes can be σ-linked to any f-structure, as long as its the same one:

(23) *Everybody* : $\forall H((\uparrow_e \multimap H_p) \multimap H_p)$

The intent of this formulation is that in a sentence such as *everybody seems to like Ernie*, with the f-structure (24),

(24)
$$f:\left[\begin{array}{ll} \text{SUBJ} & g:\left[\text{PRED} \quad \text{`Everybody'}\right] \\ \text{PRED} & \text{`Seem} \\ \text{XCOMP} & h:\left[\begin{array}{ll} \text{SUBJ} & g:[\ \ ] \\ \text{PRED} & \text{`Like'} \\ \text{OBJ} & i:\left[\text{PRED} \quad \text{`Ernie'}\right] \end{array}\right] \end{array}\right]$$

the variable H can be identified with any of the f-structures, and the conditions on glue-proofs (in our present context, the Correctness Criterion) will rule out all possibilities but f and h, giving rise to the wide and narrow scope readings, respectively.

But treating quantifiers in this way makes the formalism more complex,[8] and I propose to follow the suggestion of Lev (2007) and treat the scope variation as caused by variable instantiation of f-designators rather than by variables in the glue.

So the proposed meaning-constructor for *everybody* will be:[9]

(25) *Everybody* : $(\uparrow_e \multimap \%H_p) \multimap \%H_p$
$\%H = (\text{GF}^* \uparrow)$

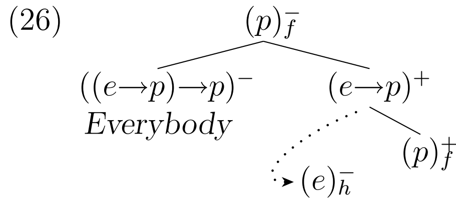
In a sentence such as *Bert likes everybody*, the instantiated form of the *everybody* constructor might be:

[8]See Kokkonidis (2008) and section 6 below for some discussion, although the present approach is formally even simpler than Kokkonidis'.

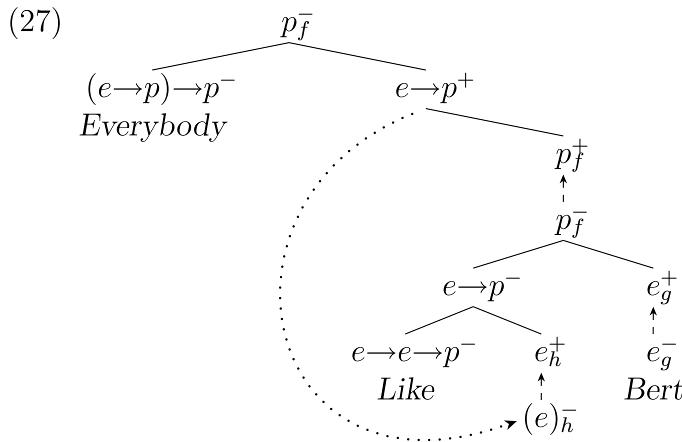
[9]Actually, for simple cases, this will work:

Everybody : $(\uparrow_e \multimap (\text{GF}^* \uparrow)_p) \multimap (\text{GF}^* \uparrow)_p$

with the Correctness Criterion blocking various bad results. But I don't know how to prove that it will always work. Because the f-designators in meaning-constructors are interpreted constructively, they need to be explicitly assigned to some GF in order to designate part of the syntactically constructed f-structure rather than some novel structure constructed only by them.

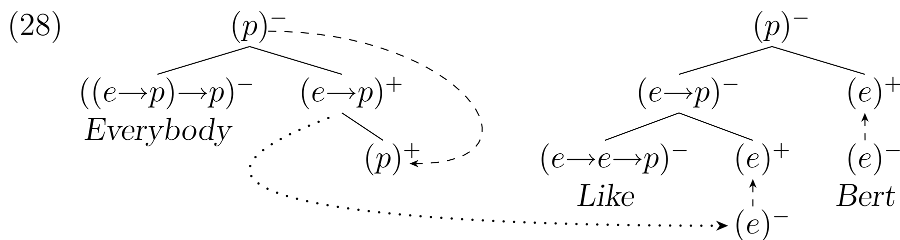


This can combine with the other constructors that the sentence will produce to give the sensible assembly (27):



This is a good representation of the syntax tree of (20), using the dotted pseudo-daughter link to represent variable-binding.

but it is also possible to produce an assembly that obeys the rules of (13), but doesn't represent a logical form (or a valid linear logic proof):



To rule this out, we need the already-mentioned 'Correctness Criterion', which has a remarkably large number of equivalent formulations.

The one we will use here is essentially due to Lamarche (1994), although this paper is very hard to follow (at least to me; I have not managed to understand much of its contents); more accessible discussions of the ideas appear in de Groote (1999) and Moot (2002:94-95). It depends on the notion of 'dynamic graph', which is constituted by the axiom-links, oriented from negative to positive, and the solid-line links in the structure-format meaning-constructors, oriented upwards, or, more formally:

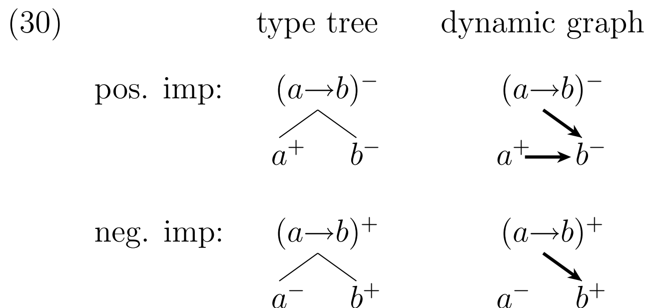
(29) a. for a negative implication, the dynamic graph links run from the nodes labelled with the implication and the antecedent to the consequent

b. for a positive implication, the dynamic graph link runs from the node labelled with the consequent to the node labelled with the implication (but not from the node labelled with the antecedent).

The dynamic graph is then just our structure-format, with the pseudo-daughter links removed, and an direction imposed on the links.

We'll try to make the idea more vivid by showing the relationship between the type-tree format and the dynamic graph format without rearranging the nodes, but depicting the dynamic links with heavy arrows:

(30)



Observe that in the structure-tree format, the left mother-daughter link in the type-tree is discarded for negative implications, but retained as the pseudo-daughter in for the positive ones (but discarded for the dynamic graph).

We now formulate the criterion:

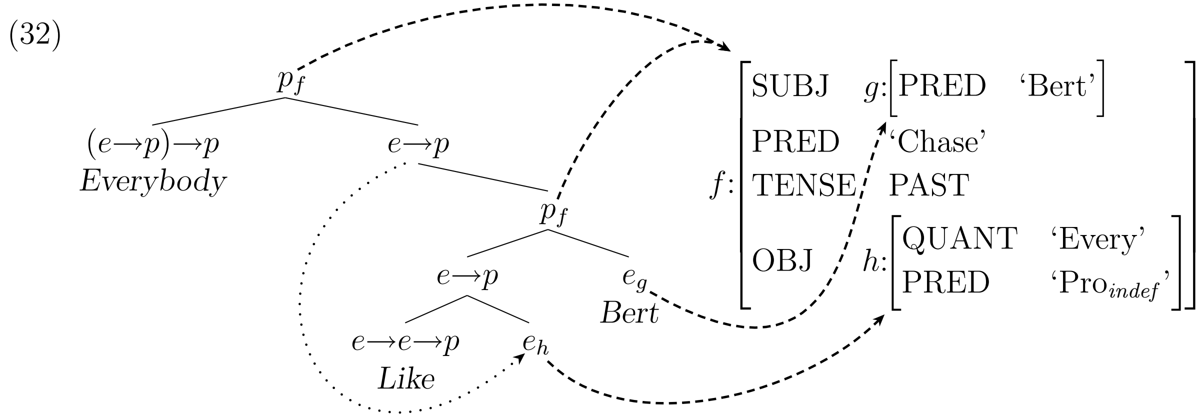
(31) **Correctness Criterion:** The dynamic graph must be (a) rooted and acyclic, and (b) every path to the root that starts at the target of a dotted link must pass through the source of that link.

(27) clearly satisfies (31), while (28) clearly doesn't, on two counts, absence of a root, and failure of every path starting at the target of a dotted link to pass through its source. In the next section, we'll see an example that passes (a) but fails (b).

Condition (a) of the criterion is formulated a bit more generally than it has to be for glue-structures containing only implication; for these, it would suffice and be more intuitive to just say that the dynamic graph must form a tree (which would constitute a reasonable parse-tree for a formula). The formulation given covers the more complex case of 'tensors', which we discuss in a later section. Condition (b) amounts to the requirement that variables be properly bound. You might be able to assemble the constructors for *everybody thinks that Bert sneezed* into an configuration that passes (a) but not (b).

Quantifiers as well as sentence-adverbials exemplify the motivation for having σ run from the glue-structure to the f-structure rather than in the opposite direction, since σ is a function in the former direction but not the latter. We illustrate this by drawing the correspondence-links for (27):

(32)



We now have a basic glue system covering the ground of the ‘core fragment’ of Dalrymple et al. (1999). Next, we will discuss some examples.

4 Some Examples

In this section, I will illustrate the system with some sample analyses. First, scope ambiguity with two quantifiers, as in:

(33) Everybody loves somebody

Instantiated constructors might be:

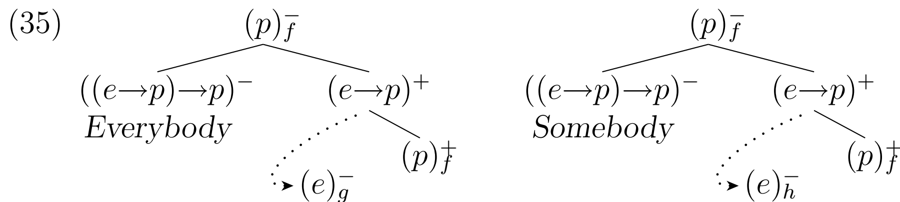
(34) *Everybody* : $(g_e{\to}f_p){\to}f_p$

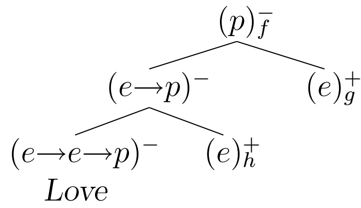
Somebody : $(h_e{\to}f_p){\to}f_p$

Love : $h_e{\to}g_e{\to}f_p$

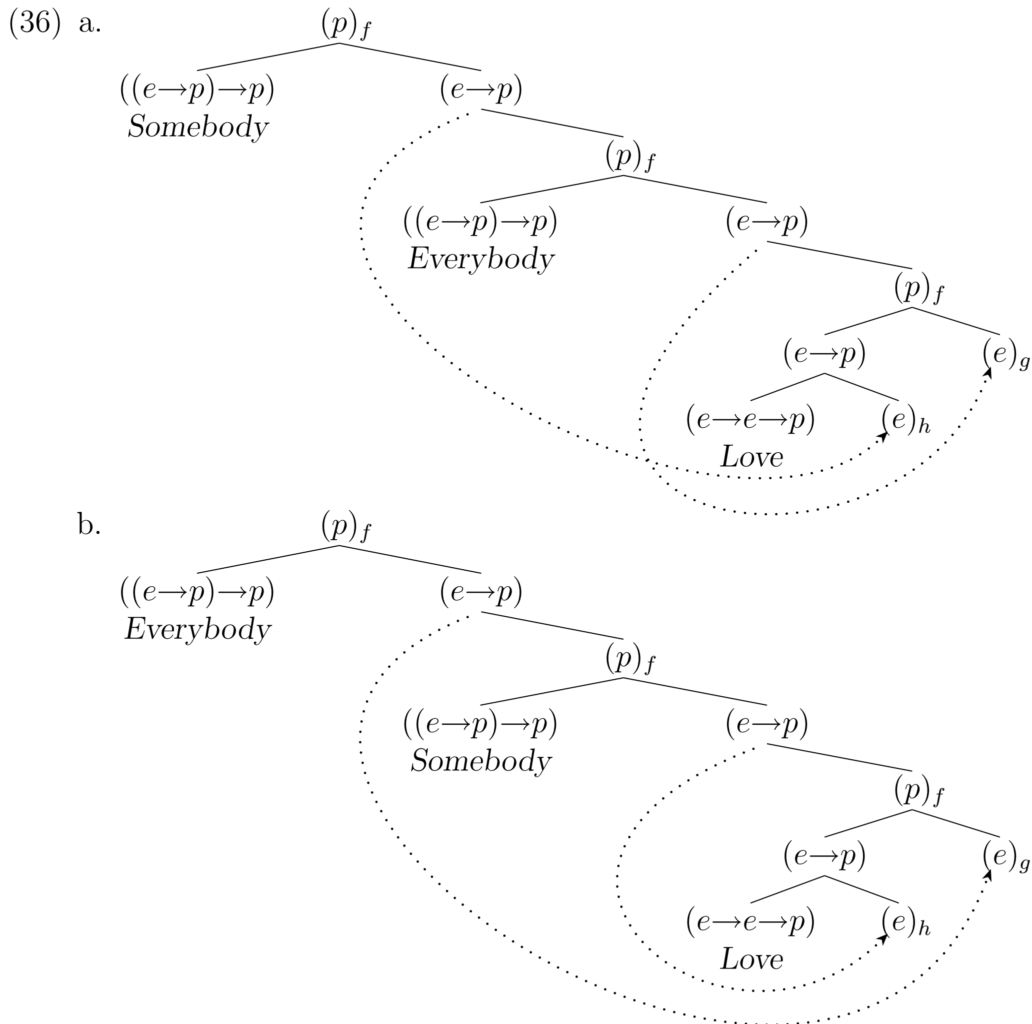
In structure-tree format, these become:

(35)





There are two ways of putting these together so as to satisfy the Correctness Criterion:



It would be a good exercise to re-render one of these in uncontracted form.

In this case the scope variation is produced by alternate arrangements of glue nodes corresponding to the same f-structure. As we discussed earlier in connection with *everybody seems to like Ernie*, scope ambiguity can also be produced by variant instantion of the $(\uparrow \mathrm{GF}^*)$ f-designator in (25). This works out nicely for *seem*, but, as discussed by Asudeh (2002), there appears to be the possibility of a problem with verbs like *try*, which also having sharing of the complement subject with the matrix subject,

but no ambiguity. But Asudeh shows how the potential ambiguity is suppressed, due ultimately to the fact that the subject is an argument of *try*.

If the f-structure for *everybody seemed to sneeze* was (37), then the the instantiated constructors would be(38):

(37)
$$f{:}\begin{bmatrix} \text{SUBJ} & g{:}\begin{bmatrix}\text{PRED} & \text{`Everybody'}\end{bmatrix} \\ \text{PRED} & \text{`try(SUBJ, XCOMP)'} \\ \text{XCOMP} & h{:}\begin{bmatrix}\text{SUBJ} & g{:}[\ ] \\ \text{PRED} & \text{`Sneeze(SUBJ)'}\end{bmatrix}\end{bmatrix}$$

(38)
$$\begin{aligned} \mathit{Everybody} &: (g_e{\rightarrow}f_p/h_p){\rightarrow}f_p/h_p \\ \lambda P.\lambda x.\mathit{Try}(P(x))(x) &: (g_e{\rightarrow}h_p){\rightarrow}g_e{\rightarrow}f_p \\ \mathit{Sneeze} &: g_e{\rightarrow}h_p \end{aligned}$$

We specify some internal structure for the meaning-side of *Try*, because it provides some useful information about the interpretation.

If we choose f as the scope, everything works out, giving the correct wide-scope reading. But with h as the scope, there is no solution. Intuitively, the best we can do is:

(39)

p_f
- $e{\rightarrow}p$
  - $(e{\rightarrow}p){\rightarrow}e{\rightarrow}p$ *Try*
  - $(e{\rightarrow}p)$ (dotted to $(e)^-_g$)
    - p_h
      - $(e{\rightarrow}p){\rightarrow}p$ *Everybody*
      - $(e{\rightarrow}p)$ (dotted to $(e)^-_g$)
        - p_h
          - $e{\rightarrow}p$ *Sneeze*
          - $(e)^+_g$
- $(e)^+_g$

Here the dynamic graph is connected and acyclic, but, because of the upper $(e)^+_g$, there's no way to hook up the variable binders to argument positions that will satisfy (b) of the Correctness Criterion.

Our final example will be an approach to anaphora that is probaby wrong for ordinary bound anaphora, but quite well-motivated for reciprocals (Lev 2007:ch. 13). The problem with bound anaphora in a sentence such as *Everybody*$_i$ *loves their*$_i$ *mother*

(singular epicene *their*) is that we need to in effect get the single variable-binder provided by *Everybody* to function semantically as if it bound two argument positions, subject of *loves*, and possessor *mother* = possessor of the object of *loves*. The most oft-used method to get this effect in glue is to use tensors, which we'll discuss in a later section, but an alternative is the following meaning-constructor, discussed by Lev (2007:253-260).[10] To formulate the constructor, we'll use the ANT(ECEDENT) GF introduced in Dalrymple (1993), which takes as value the antecedent of a pronoun. So the f-structure for our example sentence would be:

(40)
$$f:\begin{bmatrix} \text{SUBJ} & g:\begin{bmatrix}\text{PRED} & \text{`Everybody'}\end{bmatrix} \\ \text{PRED} & \text{`Loves(SUBJ, OBJ)'} \\ \text{OBJ} & h:\begin{bmatrix} \text{PRED} & \text{`Mother(POSS)'} \\ \text{POSS} & i:\begin{bmatrix}\text{PRED} & \text{`Pro'} \\ \text{ANT} & g:[\ \ ]\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

The constructor in uninstantiated form, accompanied by an iofu equation, is:

(41)
$$\lambda P.\lambda x.P(x)(x) \ : \ ((\uparrow \text{ANT})_e {\rightarrow} \uparrow_e {\rightarrow} \%H_p) {\rightarrow} (\uparrow \text{ANT})_e {\rightarrow} \%H_p$$
$$\%H \ = \ (\text{GF}^* \uparrow)$$

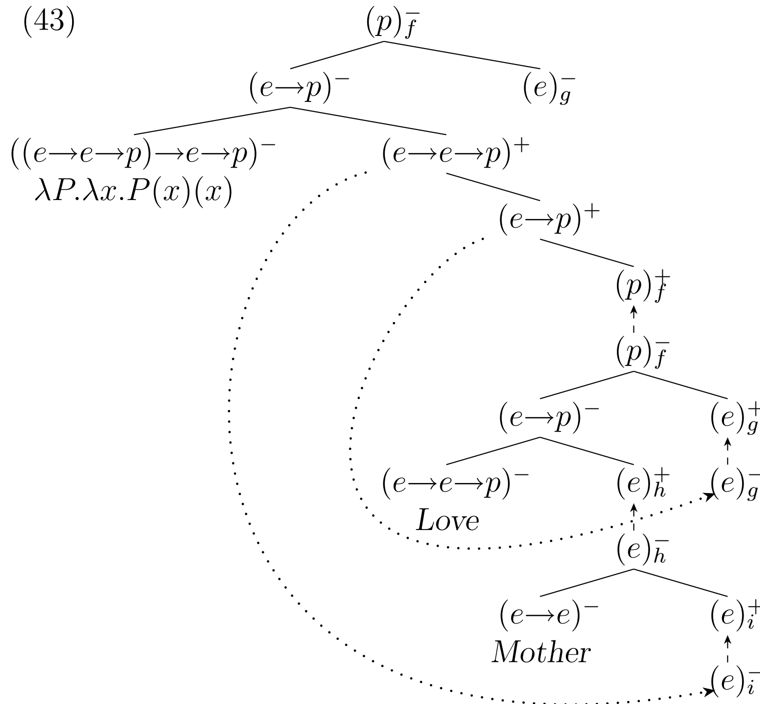
The instantiated version of this in structure-tree format would be:

(42)
$(p)^-_f$
- $(e{\rightarrow}p)^-$
  - $((e{\rightarrow}e{\rightarrow}p){\rightarrow}e{\rightarrow}p)^-$ $\lambda P.\lambda x.P(x)(x)$
  - $(e{\rightarrow}e{\rightarrow}p)^+$
    - $(e)^-_i$
    - $(e{\rightarrow}p)^+$
      - $(e)^-_g$
      - $(p)^+$
- $(e)^-_g$

From the shape of this, it is hopefully evident that it binds two variables in a proposition, one at the position of the pronoun, and another at that of the antecedent, and creates a one-place-predicate:

[10] Originally in Lev (2006), which appears to no longer be available.

(43)



To get something that looks like a conventional logical form for the sentence, we need to do β-reduction with the material inside the meaning-terms, that is, substitute the *Love* substructure for the variable P within the meaning-side of the pronoun:

(44) $\lambda x.\mathit{Love}(\mathit{Mother}(x))(x)$

(44) fails to be a linear lambda-term because one lambda is binding two positions. This is something that the linear logic system we've been using can't produce directly, because it can only use each assumption once.

An obvious question is why not to relax this assumption (which can be done in various ways, for example, in linear logic, by using 'exponentials'). A twofold answer is (a) this makes the computations considerably more difficult, because assumptions can't be discarded upon use, but must be kept available for possible later re-use (b) it seems to be linguistically just wrong. For example, in a sentence such as *Ernie didn't sneeze*, we don't want to use the negative constructor twice so as to get a positive assertion.

Therefore, instead, we continue to use linear logic for our syntactic combinations, and put something more powerful inside the meaning-sides of the constructors, so that one could say that the more powerful system is lexicalized, in that its resources are only used as specifically directed by certain lexical items.[11] I personally would like to know more about the kind of relationship between logical systems that is involved here, but that's not a topic we need to consider further in a basic exposition of glue.

[11] But see Jäger (2005) for a constrained loosening of the syntactic logic, within the framework of Type-Logical Grammar.

This is one of several possibilities for anaphora that have been considered in glue, similar to proposals made in Type Logical Grammar and related approaches, such as Jacobson (1999). Lev (2007) discusses various problems with it, such as spurious ambiguities (multiple structures corresponding to one intuitively available reading), but also uses the basic idea for a well-motivated analysis of reciprocals.

The last example I'll look at is the Montogovian analysis of intensional verbs such as *seek*. An initial point is that it's rather likely that this analysis is just wrong (Deal 2007), but it is nonetheless useful to illustrate certain features of glue.

The goal of the analysis is to account for the wide and narrow scope readings of:

(45) Bert seeks somebody

without postulating a biclausal syntactic analysis. Montague's solution was to assign this syntactic type to the verb (details adjusted):

(46) $((e{\rightarrow}p){\rightarrow}p){\rightarrow}e{\rightarrow}p$

The narrow scope reading is can be gotten by applying this directly to a quantifier, the wide scope via type-raising.

As discussed by Dalrymple et al. (1997),this form of solution can be carried over nicely to glue. We can use the following meaning-constructor for *seek*:

(47) $\mathit{Seek} : (((\uparrow \mathrm{OBJ})_e{\rightarrow}\uparrow_p){\rightarrow}\uparrow_p){\rightarrow}(\uparrow \mathrm{SUBJ})_e{\rightarrow}\uparrow_p$

This affords a situation that we haven't seen before, an implicational argument whose antecedent is itself an implication. If the f-structure of (45) is (48), the instantiated form of (47) will be (49):

(48)
$$f{:}\begin{bmatrix} \mathrm{SUBJ} & g{:}\begin{bmatrix}\mathrm{PRED} & \text{‘Bert’}\end{bmatrix} \\ \mathrm{PRED} & \text{‘Seek(SUBJ, OBJ)’} \\ \mathrm{OBJ} & \begin{bmatrix}\mathrm{PRED} & \text{‘Somebody’}\end{bmatrix} \end{bmatrix}$$

(49)

$(p)^-_f$

— $(e{\rightarrow}p)^-$

—— $(((e{\rightarrow}p){\rightarrow}p){\rightarrow}e{\rightarrow}p)^-$ *Seek*

—— $((e{\rightarrow}p){\rightarrow}p)^+$

——— $(p)^+_f$

——— (dotted arrow to) $(e{\rightarrow}p)^-$

— $(e)^+_f$

$(p)_-$

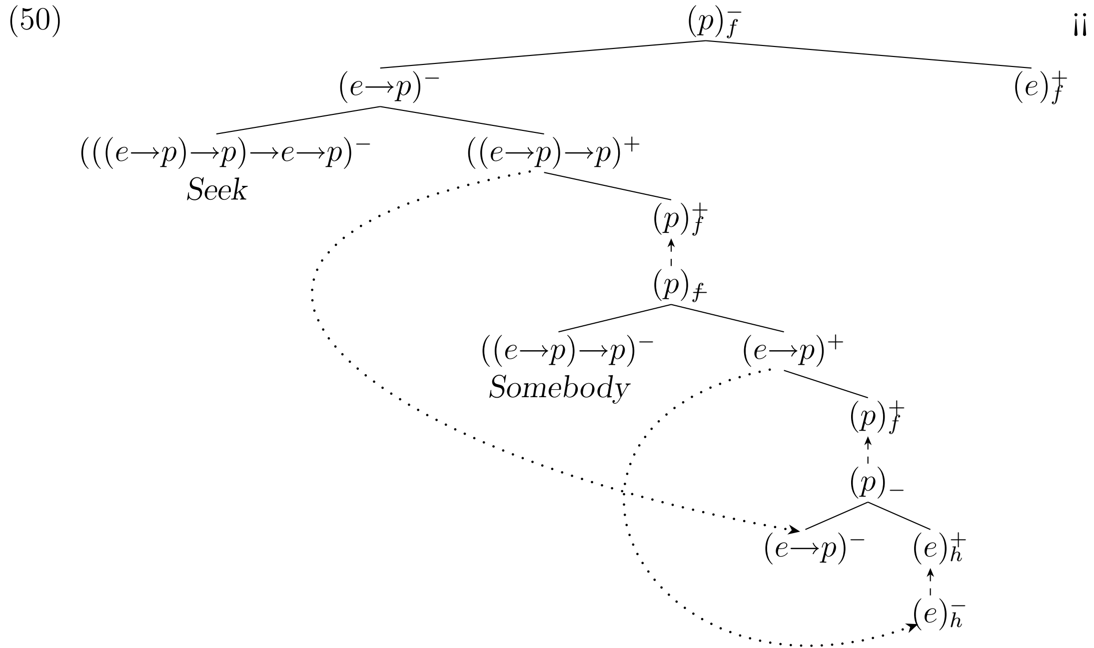
— $(e{\rightarrow}p)^-$

— $(e)^+_h$

We now see that if we allow 'third order' arguments, then the structure-tree format isn't really based on trees; this constructor can be seen as a pair of trees related in a non-tree-like manner by the variable-binding/pseudo-daughter relation.

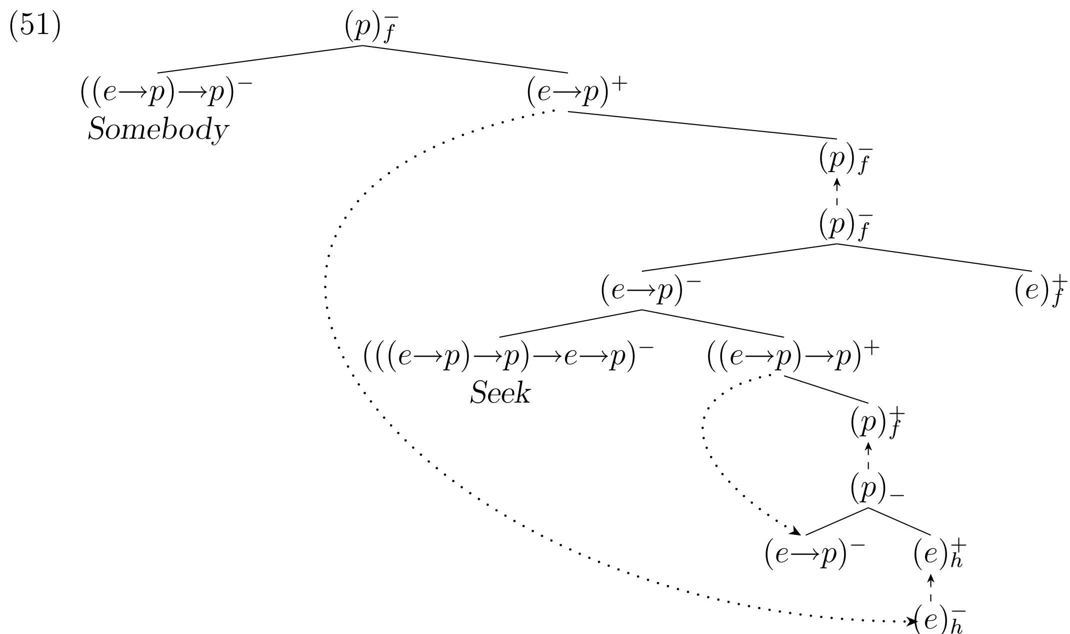
Now the positive $(e)^+_h$ of the lower tree (consitituting a binder for a variable of type $e{\rightarrow}p$) can accept input from a quantifier, and the positive $(p)^+_f$ can send output back to the quantifier, so that this is a Correct sub-assembly:

(50)



For the wide-scope reading, we just plug the $(p)^-_f$ of the lower tree into the $(p)^+_f$ of the upper one, and bind the lower tree's $(e)^+_h$ with the quantifier in the usual manner:

(51)



One might think that some scope-variation in the *seek* constructor would be needed to produce the two readings, but this is not in fact the case, as revealed by the fact that the two structures above satisfy both the basic rules for axiom-linking, and the Correctness Criterion. One way to understand this is to think of the f-structure-location not as being meaning-bearing elements in themselves, but as 'drop off points' for transactions involving meaning, imparting a degree of flexibility to the relationship between meanings and f-structures. This seems to me to be the basic intuition behind de Groote's (1999) algebraic formulation of the correctness criterion, and its elaboration in Perrier (1999).

This concludes our collection of examples. Now we relate the prefab glue system to the other main formulations.

5 Natural Deduction Proofs

Most recent work on glue has been formulated in terms of tree-style natural deduction, as presented for example in Asudeh (2004), which the reader might have to spend some time with in order to understand this section. This system uses the two rules of Implication Elimination (traditional Modus Ponens) and Implication Introduction (traditional Hypothetical Deduction) to produce from the initial meaning-constructors a proof that there is a meaning of type p associated with the entire f-structure. These rules are used to construct proofs in a system of 'linear labelled deduction', 'linear' referring to the fact that each premise is used once and once only (a restriction that goes very nicely with the tree format), 'labelled' to the fact that the objects over which deductions happen are pairs consisting of a 'proof term' representing the meaning, and a formula.

Deduction steps operate on both the formulas and the terms; the two rules are standardly formulated like this:

(52)

$$\frac{f : A \multimap B \qquad a : A}{f(a) : B} \multimap\text{-elim} \qquad\qquad \frac{\begin{array}{c}[x : A]^i \\ \vdots \\ b : B\end{array}}{\lambda x.B} \multimap\text{-intr}^i$$

Here we use the linear implication connective $\multimap$ standardly employed in the literature, rather than the '$\rightarrow$' we have been using so far. The rules derive a new formula and a new proof-term from pre-existing ones,

The elimination rule corresponds in a fairly obvious way (albeit upside-down) to our negative-polarity structure-tree nodes with two daughters, where one daughter represents a function, the other its argment, and the mother the result. In logic, the functional input is often called the 'major premise', the argument, the 'minor premise'. The introduction rule corresponds to our positive-polarity structure-tree nodes with

a daughter and a pseudo-daughter; the pseudo-daughter is a hypothesis that is 'dis-carged' by this step. Clause (b) of the correctness criterion represents the structural relation that must exist between a hypothesis and the step where it is discharged, with superscripts (assigned uniquely) marking the long-distance relationship that we have notated with the pseudo-daughter links.

For the present 'propositional' version of glue,[12] the atomic formulas can be taken as ordered pairs consisting of an f-structure label and a semantic type. If quantification is used for scope-variation, the simplest interpretation is the 'first order' glue of Kokkonidis (2008), where the type information is a predicate, applying to the f-structure label as (universally quantifiable) argument.

The rules are used to assemble a tree whose leaves (conventionally written at the top) are the original meaning-constructors, together with the assumptions that will be 'discharged' by the $\multimap$, and the root the final conclusion. For example, the deduction corresponding to (32) is:

(53)

$$\dfrac{\mathit{Everybody} \qquad \dfrac{\dfrac{\dfrac{\mathit{Like} : h_e \multimap g_e \multimap f_p \qquad [x : h_e]^1}{\mathit{Like}(x) : g_e \multimap f_p}\multimap\text{-elim} \qquad \mathit{Bert} : g_e}{\mathit{Like}(x)(\mathit{Bert}) : f_p}\multimap\text{-elim}}{\lambda x.\mathit{Like}(x)(\mathit{Bert}) : h_e \multimap f_p}\multimap\text{-elim}^1}{\mathit{Everybody}(\lambda x.\mathit{Like}(x)(\mathit{Bert})) : f_p}\multimap\text{-elim}$$

This is obviously very similar to (32), but written root-down (logicians' style) rather than root-up (linguists' style), and using co-superscripting instead of pseudo-daughter links.

But there are a number of differences between this and our present approach that are worth attending to. The first is that the deductive conception doesn't foreground the idea of a glue-structure-f-structure correspondence, even though it is latent, and used in certain glue-work, such as for example Crouch and van Genabith (1999) and Asudeh and Crouch (2002). Rather the correspondence is implicit in the structure of the formulas in the deduction, whose atomic sentences can be thought of as pairs consisting of an f-structure label and a semantic type. The f-structure labels give us a correspondence from the atomic subsentences in the formulas of the proof to the f-structure, but this is not a general correspondence from the proof-tree nodes as such to the f-structure.

In fact, although every structure-tree in our format represents a proof, the reverse isn't true. If a constructor is of non-atomic type, it's possible for another constructor to apply to it directly, as in this example:

[12] Propositional in that the glue-system doesn't use quantifiers (that is, linear logic glue quantifiers).

(54)

$$\frac{\mathit{Everybody} : (g_e \multimap f_p) \multimap f_p \qquad \mathit{Sneezed} : g_e \multimap f_p}{\mathit{Everybody}(\mathit{Sneezed}) : f_p} \multimap\text{-elim}$$

This proof doesn't fit into our present scheme because it doesn't contain any steps/nodes whose formulas are of atomic type, so the way we have set up σ won't work.

Fortunately, there is an alternative form of (54) that does correspond to a tree in our structure-format:

(55)

$$\frac{\mathit{Everybody} \qquad \dfrac{\dfrac{\mathit{Sneezed} : g_e \multimap f_p \qquad [x : g_e]^1}{\mathit{Sneezed}(x) : f_p} \multimap\text{-elim}}{\lambda x.\mathit{Sneezed}(x) : g_e \multimap f_p} \multimap\text{-intr}^1}{\mathit{Everybody}(\lambda x.\mathit{Sneezed}(x)) : f_p} \multimap\text{-elim}$$

The last step of this tree has the same general shape as (54), but the proof-term assigned to the minor premise on the right has a slightly different shape (but the formula is the same), and there is material above this step.

And more fortunately, the fact that some proofs lack structure-format counterparts doesn't actually matter, due to the relationships of equivalence between proofs studied in proof-theory, whereby (54) can be regarded as a mere variant presentation of (53) rather than an essentially different proof.

To grasp these equivalences, we look first at the corresponding ones in the lambda-calculus, β- and η-equivalence:

(56) a. $\lambda x.P(x) \equiv_\eta P$

b. $\lambda x.\phi(\alpha) \equiv_\beta [\alpha/x]\phi$, where $[\alpha/x]\phi$ means ϕ, with all instances of x replaced by α, subject to tricky conditions on variables (which can be avoided by using a unique variable for each lambda).

These equivalences hold by the usual understanding of a lambda-abstract as a function: for example if P is something that can apply to x (that is x is of type A, and P is of type $A \rightarrow B$), applying P to x and then abstracting over x shouldn't effect any change in the meaning.

Corresponding to η-equivalence of lambda-terms is a similar and indeed equivalent relationship between proofs:

(57)

$$\frac{\dfrac{f : A \multimap B \qquad [x : A]^i}{f(x) : B} \multimap\text{-elim}}{\lambda x.f(x) : A \multimap B} \multimap\text{-intr}^i \qquad \equiv_\eta \qquad f : A \multimap B$$

Looking just at the formulas, not the proof terms, the phenomenon illustrated in (57) will be familiar to anyone who has tried to find formal or semiformal logic proofs, where frequently one goes through a hopeful-looking sequence of steps, simply to wind up having derived one of the premises one started with. In words, the idea is that if you have an implication $A \multimap B$, it is pointless to use Modus Ponens to derive B with a temporary hypothesis A, and then discharge the hypothesis. Then taking the proof-terms under consideration, we can see how this flavor of pointlessness is parallel to applying a function to a variable, then abstracting over the variable to get the original function back.

The β-equivalence relation for proofs is more complex:

(58)
$$\dfrac{\dfrac{\begin{array}{c}[x:A]^i \\ \vdots\ \delta \\ b:B\end{array}}{\lambda x.b : A \multimap B}\multimap\text{-intr}^1 \qquad a:A}{(\lambda x.b)(a):B}\multimap\text{-elim} \qquad \equiv_\beta \qquad \begin{array}{c} a:A \\ \vdots\ [a/x]\delta \\ b:B\end{array}$$

Here δ represents the sequence of steps used to derive $b : B$ (including any additional premises that might be required), and $[a/x]\delta$ represents δ with every instance of x replaced by an instance of a.[13] (58) then says that if you can derive B from the hypothesis A with steps $[a/x]\delta$, it is pointless to derive $A \multimap B$ by deriving B with δ, then discharging A, and finally using Modus Ponens again to get B. . We finally observe that the terms at the conclusion of both derivations are β-equivalent, illustrating the essential equivalence of the deductive and lambda-calculus versions of β-conversion. For more on this, see Girard et al. (1989).[14]

One of the basic ideas of proof-theory is to find the 'Real Proofs', i.e. reasons why arguments are valid, abstracting over various 'bureaucratic' details of formulation that are fundamentally meaningless, but essential to have a working formalism. β- and η-equivalence are part of this project, the proofs that they equate being seen as trivial variants of the same thing. The deeper goals of proof-theory are beyond the scope of comment by linguists, but the project of identifying interesting and useful equivalence classes of proofs needs no justification, and those picked out by these two equivalences turn out to be linguistically useful.

One point is that if they are applied from left to right, so that they serve as rules for making formulas shorter (in which case they are called 'reductions'), they have the extremely important 'Church-Rosser' property, which means that if you keep performing the reductions until neither can apply, you get the same result for any given formula or proof (this is just like lowest terms for fractions, as taught in grade school—the rules

[13] Subject to the usual restrictions on variable-binding.

[14] The earlier parts of which are quite accessible, even to people without much logic background.

for reducing fractions are also Church-Rosser). This is called the 'β-η normal form' for that formula/proof. β-η equivalent formulas will have the same β-η-normal form. See a textbook on lambda-calculus such as Hindley and Seldin (1986) for more on normal forms.

One can also consider only β-equivalence/reduction, in which case the equivalence classes will be a bit finer: P and $\lambda x.P(x)$ are in the same β-η-equivalence class, with P as the normal form, but different β-equivalence classes, each being the normal form in their class. Our structure-format trees are β-normal, because their manner of production doesn't provide for applying an argument to a derived function (this requires an additional kind of link, called a 'cut link'). But they aren't η-normal, as we have seen in our discussion of *everybody sneezed*. Rather, they are 'η-expanded', but only up to a point.

If we start with a predicate P of type $a{\rightarrow}b$, and 'η-expand' (applying the equivalence from right to left), we get $\lambda x.P(x)$, and we can this again to get $\lambda y.(\lambda x.P(x))(y)$ and so on (using a different variable for each lambda, to avoid some tricky aspects of variable-management). But this formula is subject to β-reduction, which the earlier one was not. So we have a class of formulas which are as 'η-expanded as they can be', while still being β-normal. These have the structural propery that every subexpression of functional type is explicitly applied to an appropriate argument, a property quite reminiscent of the frequent preference in theories of generative grammar for explicitly representing all arguments of a predicate in 'underlying form', even when they aren't realized overtly.

η-expansion up to β-normality is also essential to support our conception of a simple correspondence from glue-structure to f-structure, by assuring the existence of nodes/proof-steps/subexpressions of atomic type, which have a naturally determined f-structure correspondent. These linguistic considerations are also supported by proof-theory, because each β-η-equivalence class of formulas has a single one of this type, which we can derive by first finding the β-normal form, and then η-expanding to the maximum consistent with β-normality.

So the conclusion of this rather long tale is that our structure-tree format contains one representative of each β-η-equivalence class of proofs/lambda-expressions, and this representative has a property that has long been conjectured to obtain for underlying structures in linguistics, which can be described as overtly representing all predicate-argument relations. Furthermore, it supports the distinct and somewhat more recent idea of a correspondence-based syntactic theory (the idea itself is rather old, going back at least to George Lakoff's ideas about 'global rules', although the LFG formalism as developed by Ron Kaplan was, it seems to me, the first workable proposal). So ideas from lingistics and proof-theory come together reasonably nicely, in this instance.

There is one more kind of equivalence on lambda-expressions that we should mention, α-equivalence, which holds between formulas that can be converted each into the other by a replacement of variables (as usual, subject to tricky constraints). Our use of

pseudo-daughter links eliminates this issue, and it is clearly the case that each of our structures corresponds to an α-equivalence class of proofs or conventional lambda-expressions. This is not a problem; indeed, it is an advantage, since people frequently want to consider all α-equivalent formulas as being the same thing.

6 Proofs and Terms

The next issue we'll consider is the relation between the proofs and the proof-terms. The short answer is that for the natural deduction system of IILL, this relationship is essentially identity: the proof in tree format has exactly the structure of the lambda-term. For some more complex systems, such as those with tensors, this claim isn't so straightforward, but is ultimately defensible.[15] So in a natural deduction system, there is no real point in constructing the proof-term in addition to the proof-tree, so we omit this step in the prefab glue presentation.

However, proof-terms can be useful if there are certain steps in the proof that we wish to discard. In standard glue, this happens with the use of universal quantification for quantifier scope. In such a system, the $\beta\eta$-normal form of the proof for *everybody sneezed* would be:

(59)
$$\dfrac{\dfrac{\mathit{Everybody} : \forall H (g_e \multimap H_p) \multimap f_p}{\mathit{Everybody} : (g_e \multimap f_p) \multimap f_p}\,UI \qquad \mathit{Sneezed} : g_e \multimap f_p}{\mathit{Everybody}(\mathit{Sneezed})}\multimap\text{-elim}$$

In the UI step, the quantifier is stripped off and its variable replaced by a freely chosen f-structure label, but nothing happens in the proof-term, contrary to the treatment in sources such as Girard et al. (1989:81).

A more general perspective on this is that we can regard proof-terms, among other things, as a sort of template for defining functions over the proofs, with the standard proof-term assignment system being the identity function. Standard 'new glue' then would involve a function that in effect erases the UI steps, something we can avoid by treating scope-variation with variable instantiation. A further motivation for proof-terms in glue is the fact that early glue analyses weren't formulated with Natural Deduction, but in the Gentzen Sequent format, whose derivation trees do not look like lambda-terms, but proofs that one would want to regard as equivalent have the same proof-term. For more on this, see for Crouch and van Genabith (2000).

We can now see that our structure-tree format can be regarded as pregenerating in the lexicon pieces of proofs in Natural Deduction format, and then sticking them together into a complete proof with the axiom-linking rules. But we can also regard it as an alternate formulation (one of many) of linear logic proofs, called proof-nets, to which we now turn.

[15]Via 'commuting conversions' as in Mackie et al. (1993).

7 Proof Nets

Proof-nets were devised by Girard as the preferred format for proofs in cases where they worked well, and our structure-format is in fact a variant format for proof-nets. They were historically derived from the 'one sided sequent calculus', as described in Troelstra (1992), Crouch and van Genabith (2000), Moot (2002) and many other sources. However it can also be regarded as a technique for doing semantic assembly on the basis of the unmodified type-trees. To present the background we need to introduce the notion of a sequent. This is a technique for presenting a putatively true fact of logical consequence by putting a collection of assumptions to the left of a 'turnstyle' symbol, and one or more conclusions to the right. In a valid sequent, one of the conclusions must follow from all of the conclusions, so that $A.B, C \vdash D, E$ means that we can conclude either D or E from A, B and C. For semantic assembly, we are interested only in 'intuitionistic' sequents, which have only a single conclusion.[16]

For semantic assembly, our assumption sequents are the glue-sides of the constructors, construed as formulas of implicational linear logic, and the conclusion is an atomic formula specifying p as its semantic type and the entire f-structure as its syntactic information. The sequent is valid if it has a proof, which can be formulated in various ways, including the construction of a proof-net. This is done by connecting the atomic formulas of the assumptions and the conclusion subject to our previous rules (involving polarity assigment) and the Correctness Criterion.

The polarity rules are as follows:

(60) a. each formula on the left of the turnstyle is negative

b. the formula on the right of the turnstyle is positive

c. The consequent of an implication has the same polarity as the whole implication

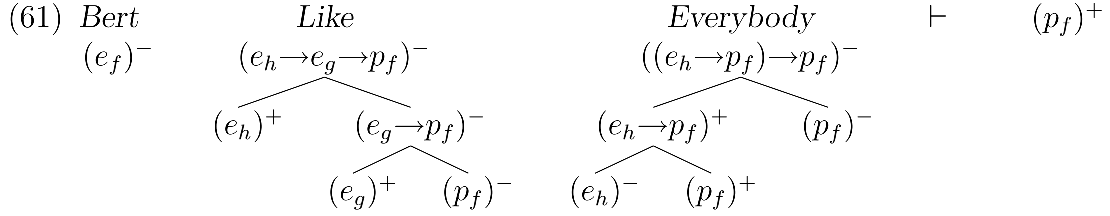
d. The antecedent of an implication has the opposite polarity as the whole implication.

These rules will assign polarities to all (instances of) subformulas in the sequent, down to the atomic formulas, and have the same effect as our previous polarity rules, except that they are defined over type-trees rather than structure-trees.

Logicians draw the type-trees with the roots at the bottom, but we'll illustrate a polarity-assignment with the roots up, in linguists' style:[17]

[16]Restricting sequents in this way has a very interesting effect on the power of the logic, which we won't look into here.

[17]In many presentations, the negative nodes are labelled with the 'tensor' symbol $\otimes$, and the positive ones with the 'par' symbol $\parr$, but this has to do with the relationship between proof nets and the one-sided sequent calculus, and can be ignored here.

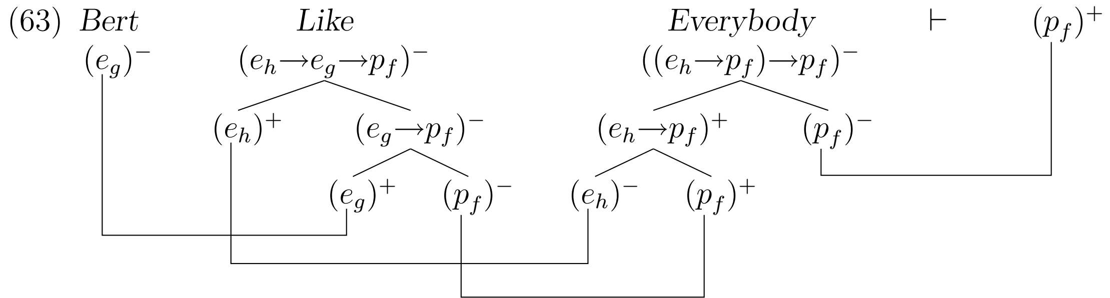


A somewhat more substantive difference from what we have done so far is the addition of a single positive polarity formula to the meaning-constructors; this can be understood as an explicit statment of what kind of result we are looking for, a more general and formally better approach than our previous method of building this specification into the axiom-linking rules. For example, in some contexts, we might be looking for a meaning of type e rather than of type p, which can now be specified by adding a positive of type e rather than p. The premises and the conclusion together constitute what is called a 'frame' in Type-Logical Grammar.

Now we add the proof-net links, but thanks to the additional positive, the rules can be slightly simplified:

(62) Link positive and negative nodes atomic nodes of the same syntactic location and semantic type, in non-overlapping pairs, with none left over.

We could introduce the additional positive into the structure-tree format as well; it would appear at the top of the tree, and the previously unconnected negative would plug into it. The axiom-linked version of (61) is:



Now the Correctness Criterion can be defined in many ways, including in terms of the Dynamic Graph, which is defined as the result of the same link-reorganization as we use to get the solid links of the structure-tree format, But instead of formulating clause (b) the Correctness Criterion in terms of pseudo-daughters, we do so in terms of the links connecting positive implications to their antecedents:

(64) Every path to the root in the dynamic graph that starts at the antecedent of a positive implication must pass through the consequent of that implication (or, equivalently, the entire implication).

It should be evident that our 'structure-tree' format is simply proof-nets with a different organization of the graph-links, trivial because it can be described locally to each implication.

Although the proof-net format involves minimal change from the semantic type-trees, the structure-tree/dynamic graph format expresses what is involved in defining functions over the semantic assembly. This is an essential aspect of glue, because the results of assembly are not supposed to be meanings as such, but rather instructions for assembling meanings of the words and perhaps constructions of the utterance into a meaning for the whole. These instructions can be thought of as schemes for labelling the nodes of the proof-net, starting with the root notes of the glue-sides of the meaning-constructors, which are labelled by their meaning-sides, with values being assigned in the order prescribed by the dynamic graph. de Groote (1999) defines a 'minimal labelling' scheme that provides the mininum information needed to assess correctness, while Perrier (1999) describes a 'maximal labelling' with lambda-terms that describes the structure of the proof-net, and is equivalent to the transformation from type-trees to structure-trees described here, but applied after assembly rather than before. Perrier's rules are used in Andrews (2004), although with the polarities reversed.

8 Tensors and Anaphora

For some initial motivation for tensors, consider the situation of somebody who is unconvinced by the usual reasons (e.g. Marantz (1984)) for thinking that the arguments of multi-place predicates are curried, and instead prefers the more usual treatment found in introductory logic, where the subject and object of a transitive verbs are in effect packed into a 'complex subject' in the form of an ordered pair, to which the verb then applies as a predicate.[18] We can implement such an analysis in our type-theory by introducing an additional type-constructor, 'product', or 'pairing', usually symbolized with '$\times$', so that if a is a type and b is a type, then $a \times b$ is a type (alongside of $a{\rightarrow}b$), the type of pairs whose first component is a and whose second is b. We can then use a type such as $e \times e{\rightarrow}p$ (assuming the convention that $\times$ binds tighter than $\rightarrow$) for transitive verbs, rather than $e{\rightarrow}e{\rightarrow}p$. As long as our verb meanings can be seen as a kind of function, which appears to be the case for all approaches to semantics known to me,[19] this new type will do the same job as the old one, in terms of brute capacity to describe semantic composition; our preference for the curried version is due to its interesting alignment with further properties of language.

To use product types in glue, we need to introduce the 'tensor' connective of linear logic (multiplicative conjunction), usually symbolized as '$\otimes$'. A meaning-constructor

[18]This can be seen as implicit in notations such as $\mathit{Like}(x, y)$, with the parentheses around x and y packing them into a pair.

[19]The actual requirement is even weaker: the meanings must be inhabitants of a Cartesian Closed Category.

for a transitive verb using the tensor might be (assuming the convention that $\otimes$ binds tighter than $\multimap$):

(65) $\mathit{Kill}\ :\ (\uparrow \mathrm{SUBJ})_e \otimes (\uparrow \mathrm{OBJ})_e \multimap\ \uparrow_p$

What this says, intuitively, is ‘form a pair from the meanings of the subject and object, and present that to the meaning of the verb as argument’.

We can extend glue-structures/proof-nets to include tensors by adding tensor nodes in addition to the implication nodes we already have, with the polarity-assignent convention that a tensor and its components all get the same polarity. For positive tensors, the appropriate orientation in the tree-format is tensor over components (and the dynamic graph links are from each of the components to the tensor). So the tree-representation of (65) would then be (f-structure information omitted):

(66)
$(p)^-$
$(e \otimes e \multimap\ p)^-$ $(e \otimes e)^+$
Kill $(e)^+$ $(e)^+$

It is hopefully evident that the capacity of this structure to fit into proof-nets is exactly the same as that of its curried counterparts (two of them, since either argument could be composed first), since it has two inputs of type e, and one output of type p (adding the f-structure information wouldn’t change this).

But we’ll require some additional machinery if we want to do work in the meaning-side, for example lexical decomposition. Suppose we want to express the idea that *kill* means *cause to die* (or some more empirically tenable version of this proposal). Our *Kill*-meaning here applies to a pair of type $e \times e$, but for the decomposition we need to dissect the pair into its two components, and feed each to a different predicate. This can be done with the ‘projections’ $\pi_1^{e,e}$ and $\pi_2^{e,e}$, which extract the first and second members, respectively, of something of type $e \times e$. This looks superficially like a transgression beyond the bounds of linear logic, but, (a), things aren’t quite that simple, as we will discuss later (b) it doesn’t really matter, because within the meaning-side we’re not supposed to be bound by the restrictions of linearity. However, with these notations, we need to be a bit fussier about notating types of variables, which can be done by superscripts, written in the position where the variable is bound. Then the decomposed meaning-side might be:

(67) $\lambda p^{e\times e}.\mathit{Cause}(\pi_1^{e,e}(p), \mathit{Die}(\pi_2^{e,e}(p)))$

(Note that p here is mnemonic for ‘pair’, not ‘proposition’.)

The use of input (positive) tensors for arguments has been a persistent minority option in glue (and the norm in the ‘old glue’ notation, where the projections weren’t needed

to build the meanings), although I think it could be weakly challenged on the basis of requiring more arbitrary decisions than the curried alternative.

A more interesting question arises when we consider the fact that if we have input tensors, we might also expect to have output (negative polarity) tensors. Asudeh (2004) discusses various uses to which these might be put, but the only worked out and persistently popular example is anaphora.

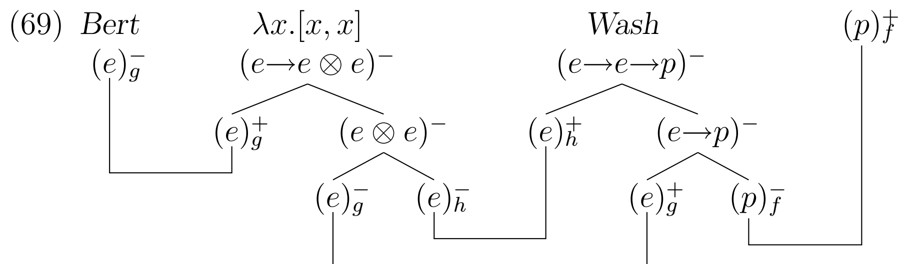
Amongst the considerations militating against output tensors (and therefore, presumably, input tensors as well) is that fact that they create a fair amount of essentially trivial but nevertheless annoying mess in the formalism, which we will be looking at shortly. Another, perhaps more subtle one, is that their use appears to be highly disciplined by Universal Grammar: output-tensors can't just send their outputs to anywhere, but only act in limited ways, such as returning the meaning of antecedent of an anaphor to where it was taken from, plus a copy to the location of the anaphor itself. A need for a lot of stipulative UG discipline on the use of tensors suggests that perhaps they shouldn't be there at all.

But the superficial annoyances do not appear to be associated with genuine mathematical difficulties, indeed, the contrary appears to be the case, due to the 'adjunction' relationship connecting the product and implicational type constructors (Mints 1981).[20] Furthermore, the counterparts to the tensors in systems very closely related to glue (such as the products in closed cartesian categories, and the dot-composites in categorial grammar) play a conceptually central role in those systems, so that their absence from glue might be seen as peculiar. So, without endorsing any specific position on the desireability of including tensors, we present here a discussion of their use in an analysis of anaphora.

This analysis was formulated in the 'old glue' format by Dalrymple et al. (1997), and used in new glue by Asudeh (2004). The idea here is to use a negative tensor as output of the pronoun constructor, which is then:

(68) $\lambda x.[x, x] \;:\; (\uparrow \mathrm{ANT})_e \multimap (\uparrow \mathrm{ANT})_e \otimes \uparrow_e$

Suitably instantiated, this can be combined with other constructors into a conventional (modulo being upside-down) format proof-net like this:

(69) 

[20] I wouldn't claim to understand anything in this paper other than that it makes this point.

This satisfies the correctness criterion.

But an issue arises when we try to convert it to the prefab structure format, since assembling *John* and *Himself* yields a configuration that we haven't produced rules for dealing with:

(70)
$$(g_e \otimes h_e)^-$$
$$(g_e \rightarrow g_e \otimes h_e)^- \quad (g_e)^+$$
Himself
$$(g_e)^-$$
John

The problem of course being the composite output at the top.

The dynamic graph links for a negative tensor run from the tensor to its components, Therefore, for full notational consistency, we ought to render this as an upward-branching configuration, with axiom-links issueing forth from the leaves. But due to the apparently strong substantive limitations on the use of negative tensors in syntax, there is no practical necessity to do this, and it is notationally sufficient to run links from the two components of this output to the two inputs of *Washed* (that's what the proof-net does), but the result doesn't much resemble standard conceptions as to what a logical form would look like:

(71)
$$(f_p)^-$$
$$(g_e \rightarrow f_p)^- \quad (g_e)^+$$
$$(h_e \rightarrow g_e \rightarrow f_p)^- \quad (h_e)^+$$
Washed
$$(g_e \otimes h_e)^-$$
$$(g_e \rightarrow g_e \otimes h_e)^- \quad (g_e)^+$$
Himself
$$(g_e)^-$$
John

Although (71) doesn't have the usual tree-structure of a logical form without representation of variable-binding, it isn't too hard to come up with a reasonable way of reading it. The idea is that the axiom-links from the first and second components of the output tensor are treated as being like first and second projections, so that (71) is read as if it were:

(72) $\mathit{Washed}(\pi_1(\mathit{Himself}(\mathit{John})))(\pi_2(\mathit{Himself}(\mathit{John})))$

The main difference being that in a linear format such as conventional lambda-calculus notation, we need to use two copies, rather than two links to a shared substructure. But there is clearly no reason why underlying linguistic representations ought to be conveniently linearizable.

This analysis shares with the previous treatment of anaphora the property that the logical forms produced by glue don't represent an analysis of coreference (as opposed to some other kind of meaning-operation applied to an NP), until we take into account the internal structure of the meaning-side of the pronoun constructor (reflexive, in this case). As far as glue itself is concerned, it simply says that the subject and object of the sentence are first and second components of something produced as output by something that applies to the subject (or, more generally, antecedent of the pronoun). The linear logic of glue is not capable of copying the reference of the antecedent so as to supply these copies to different positions; this requires the intervention of an additional and less constrained logic that can manipulate the internals of the meaning-sides.

This treatment of anaphora shares with the previous tensor-free one the property that the analysis can't be seen as one of *anaphora* as opposed to some other kind of dependency as long as we remain strictly within glue, since we need to access the inside of the meaning-terms and go beyond linear logic in order to describe the copying/multiple-use of resources that is the essential feature of anaphora.

Projections aren't normally used with tensors in LFG glue (although they are Type-Logical Grammar), and might be thought to be incompatible with linear logic, but this is not exactly the case. To explain why, we need to look at the proof-term builder that is standardly used for tensors in glue, the '**let**' constructor, as discussed in Asudeh (2004). This is introduced by a rule of ⊗-elim in Natural Deduction:

(73)

$$\dfrac{\begin{matrix} \\ \vdots \\ z : A \otimes B \end{matrix} \qquad \begin{matrix} [x : A]^i \quad [x : B]^j \\ \vdots \\ c : C \end{matrix}}{\textbf{let } z \textbf{ be } x \times y \textbf{ in } c : C} \otimes\text{-elim}^{i,j}$$

to produce results like:

(74) $\textbf{let } \mathit{Himself}(\mathit{John}) \textbf{ be } x \otimes y \textbf{ in } \mathit{Washed}(y)(x)$

The idea of **let** is that it undergoes a β-reduction such that if the first argument is a product, the two components are simultanously substituted in the third argument for the pair of variables specified in the second. That is:

(75) $\textbf{let } [a, b] \textbf{ be } x \otimes y \textbf{ in } c \quad \Rightarrow_\beta \quad c[a/x, b/y]$

The effect is that if we can get access to the internal structure of *Himself* as $\lambda x.[x.x]$, then the desired reading will be delivered by this sequence of reductions:

(76) **let** $(\lambda x.[x.x])(John)$ **be** $x \otimes y$ **in** $Washed(y)(x) \Rightarrow_\beta$
let $[John, John]$ **be** $x \otimes y$ **in** $Washed(y)(x) \Rightarrow_\beta$
$Washed(John)(John)$

The reason that projections are OK after all is that the proofs that get the same proof-terms using the projections are those that are interconvertible under the standard 'commuting conversions' as presented in for Mackie et al. (1993) or Benton et al. (1992) (Melliès (2008) provides useful perspective on some of the issues). Some early glue-workers have told me that the use of **let** as opposed to projections was not a considered decision (perhaps however motivated by the fact that **let** appears to prevail in the linear logic literature), and I am aware of no explicit discussion of the reasons for using projections in the type-logical grammar literature, either.

One aspect of the choice seems to be that projections fit more naturally to the use of proof-nets, while the **let**-constructors more closely follow the grain of Natural Deduction. That there is a problem with **let** and proof-nets can be seen from the fact in an adverbial example such as:

(77) John probably washed himself

we have a single proof-net/prefab glue structure, but have to choose between two different natural deductions/**let**-terms:

(78) a. *Probable*(**let** *Self*(*John*) **be** $x \times y$ **in** *Washed*$(y)(x)$)

b. **let** *Self*(*John*) **be** $x \times y$ **in** *Probable*(*Washed*$(y)(x)$)

We could reformulate ⊗-elimto use projections rather than **let**:

(79)

$$\dfrac{\begin{matrix} \vdots \\ z : A \otimes B \end{matrix} \qquad \begin{matrix} [x:A]^i \quad [x:B]^j \\ \vdots \\ c : C \end{matrix}}{[\pi^1(z)/x, \pi^2(z)/y]c : C} \otimes\text{-elim}^{i,j}$$

This would eliminate the difference in proof-terms, but they would then no longer reflect the 'raw' structure of the proofs, before consideration of the commuting conversions.

So we see that natural deductions discriminate proofs a bit more finely than is desireable, but in a way that can be fixed with commuting conversions. On the other hand the Gentzen Sequent format used in most early glue work discriminates proofs too finely (see Crouch and van Genabith (2000) and Restall (in preparation) for useful discussions of this format), a problem that proof-nets were invented to deal with. For the ⊸ -⊗ fragment of linear logic that is useful for meaning-assembly, these appear to discriminate proofs just right. But this issue doesn't arise at all until we include tensors in the fragment.

9 The Semantic Projection

An aspect of glue that we haven't discussed yet is the 'semantic projection', notated with σ, but functioning quite differently from the σ used here. Conventional σ runs from f-structure to an additional level of attribute-value structure, where a small number of semantically relevant attributes are stored. The semantic projection has been criticized on various grounds by Andrews (2004) and Kokkonidis (2008), but an argument for it has been provided by Asudeh (2005).

Amongst the attributes stored on the semantic projections are VAR and RESTR, used in the analysis of common nouns. Their meaning-constructors look like this:

(80) $\mathit{Monster} \ : \ (\uparrow_\sigma VAR)_e \multimap (\uparrow_\sigma RESTR)_p$

These annotations are supposed to add to the f-structure correspondent of the noun σ-correspondent like this:

(81) $\begin{bmatrix}\text{PRED} & \text{`Monster'}\end{bmatrix} \cdots\cdots \sigma \cdots\cdots\!\!\rightarrow \begin{bmatrix}\text{VAR} & [\ ] \\ \text{RESTR} & [\ ]\end{bmatrix}$

The implicit (and in the standard formulation, unnamed) correspondence from the atomic formulas on the glue-sides of the meaning-constructors then goes to this new 'semantic projetion' rather than directly to the f-structures.

One problem with this projection is a general lack of empirical motivation, except for Asudeh's argument, which we'll discuss later. Andrews and Manning (1999) argue that projections should be recognized on the basis of differential attribute-sharing behavior. For example, the μ-projection proposed by Butt et al. (1996) and Butt et al. (1999) to hold certain morphological attributes associated with verbs is different from f-structure because, unlike f-structure, it is not shared between the different verbs of Romance auxiliary and complex predicate constructions. There is however no evidence that the conventional σ-projection shares differently than f-structure.

Rather it seems to serve primarily as a place to stash attributes that are seen as useful or necessary for semantic interpretation, but seem intuitively unnatural to place in f-structure, even if doing this doesn't cause any actual problem. VAR and RESTR are good examples of such attributes, but the problem is that they themselves appear to be technically unnecessary, but merely make the constructors for quantifiers seem more comfortable (universal quantifier left implicit):

(82) $\mathit{Every} \ : \ ((\uparrow_\sigma VAR)_e \multimap (\uparrow_\sigma RESTR)_p) \multimap \uparrow_{\sigma e} \multimap H_p$

The VAR and RESTR attributes serve to keep the material associated with the noun cleanly separated from that of the scope. But they are unnecessary: these two constructors have the same actual functionality as (82) and (80) (the semantic projection could be added here with no effect on this point):

(83) $\textit{Monster} \ : \ \uparrow_e \multimap \uparrow_p$

$\textit{Every} \ : \ (\uparrow_e \multimap \uparrow_p) \multimap \uparrow_e \multimap H_p$

On assembly, the correctness criterion will require that whatever the first $\uparrow_e$ feeds into must itself feed into the $\uparrow_p$, rather than the H_p. So the extra attributes don't actually do anything, although they might make people feel more comfortable.

Another, rather technical, problem communicated to me by Miltiadis Kokkonidis (originally to Mary Dalrymple from a currently unkown source) is that on the set-theoretic interpretation of the LFG formalism as presented for example by Kaplan (1995), the semantic projection won't work at all, since most of the attribute-values are empty, and will be represented as the empty set, and will therefore be identical (this problem can be avoided by using the graph-theoretic formalism as proposed for example by Kuhn (2003)).

Against these considerations is Asudeh's (2006) observation that although the implicit arguments of relational nouns such as *neighbor* seem to function as bound anaphors for the purposes of quantification:

(84) Every resident complained about a neighbor

they don't act as if they introduced resumptive pronouns in relative clause-constructions. Asudeh provides an analysis in which relational nouns take their implicit arguments as values of a ARG attribute on the semantic projection, in such a way that this argument can be bound by a quantifier, but not accessible to a 'resource manager' as used in Asudeh's (2004) theory of resumption.

The arguments against the semantic projection are from non-necessity rather than impossibility, so we need to have a counter to Asudeh's argument in order to dispense with it. Our present analysis with 'inverted' σ in fact provides one. We can provide the following constructors and side-equation for a relational noun such as *neighbor*:

(85) $\textit{Neighbor} \ : \ \%H_e \multimap \uparrow_e$

$\lambda x.[x, x] \ : \ (\%\mathrm{H}\,\textit{ANT})_e \multimap (\%\mathrm{H}\,\textit{ANT})_e \otimes \%H_e$

$(\%H\ \mathrm{ANT}) = (\mathrm{GF}^* \uparrow)$

%H will here construct a 'free-floating' piece of f-structure which bears no GF to the rest of the syntactic structure, but is connected to it by an ANT attribute. This allows the implicit argument of a relational noun to be bound by a quantifier, without letting it serve as a resumptive pronoun.

This should not be regarded as settling the issue fully; for one thing, it is not part of a comprehensive analysis of all of the known properties of implict arguments. But it does suffice to show that so far, the semantic projection is under-motivated.

The analysis proposed here relies an a point which is usually implicit in the glue-literature, which is that the attribute-value designators in meaning-constructors are interpreted constructively, and don't identify values without an explicit instruction to do so. Therefore the $\%H$ in (85) won't get identified with any other piece of the f–structure.

References

Andrews, A. D. 2004. Glue logic vs. spreading architecture in LFG. In C. Mostovsky (Ed.), *Proceedings of the 2003 Conference of the Australian Linguistics Society*. URL: http://www.als.asn.au/.

Andrews, A. D. 2007. Generating the input in OT-LFG. In J. Grimshaw, J. Maling, C. Manning, and A. Zaenen (Eds.), *Architectures, Rules, and Preferences: A Festschrift for Joan Bresnan*, 319–340. Stanford CA: CSLI Publications. URL: http://arts.anu.edu.au/linguistics/People/AveryAndrews/Papers.

Andrews, A. D., and C. D. Manning. 1999. *Complex Predicates and Information Spreading in LFG*. Stanford, CA: CSLI Publications.

Asudeh, A. 2002. A resource-sensitive semantics for Equi and Raising. In D. Beaver, S. Kaufmann, B. Clark, and L. Casillas (Eds.), *The Construction of Meaning*. Stanford, CA: CSLI Publications.

Asudeh, A. 2004. *Resumption as Resource Management*. PhD thesis, Stanford University, Stanford CA. http://http-server.carleton.ca/~asudeh/ (viewed Oct 2006).

Asudeh, A. 2005. Relational nouns, pronouns and resumption. *Linguistics and Philosophy* 28:375–446.

Asudeh, A. 2006. Direct compositionality and the architecture of lfg. In M. Butt, M. Dalrymple, and T. H. King (Eds.), *Intelligent Linguistic Architectures: Variations on Themes by Ronald M. Kaplan*, 363–387. Stanford, CA.

Asudeh, A., and R. Crouch. 2002. Coordination and parallelism in glue semantics: Integrating discourse cohesion and the element constraint. In *Proceedings of the LFG02 Conference*, 19–39, Stanford, CA. CSLI Publications. URL: http://csli-publications.stanford.edu.

Barwise, J., and R. Cooper. 1981. Generalized quantifiers and natural language. *Linguistics and Philosophy* 4:159–219.

Benton, N., G. M. Bierman, J. M. E. Hyland, and V. de Paiva. 1992. Term assignment for intuitionistic linear logic. Technical report. URL: citeseer.ist.psu.edu/article/benton92term.html.

Butt, M., T. H. King, M.-E. Niño, and F. Segond. 1999. *A Grammar-Writer's Cookbook*. Stanford CA: CSLI Publications.

Butt, M., M. E. Niño, and F. Segond. 1996. Multilingual processing of auxiliaries within LFG. In D. Gibbon (Ed.), *Natural Language Processing and Speech Technology*. Berlin: Mouton de Gruyter.

Casadio, C. 1988. Semantic categories and the development of categorial grammar. In E. B. R.T. Oehrle and D. Wheeler (Eds.), *Categorial Grammar and Natural Language Semantics*, 95–123. Reidel.

Crouch, R., and J. van Genabith. 1999. Context change, underspecification, and the structure of glue language derivations. In Mary Dalrymple (Ed.), 117–189.

Crouch, R., and J. van Genabith. 2000. Linear logic for linguists.
URL: `http://www2.parc.com/istl/members/crouch/`.

Dalrymple, M. (Ed.). 1999. *Syntax and Semantics in Lexical Functional Grammar: The Resource-Logic Approach*. MIT Press.

Dalrymple, M., V. Gupta, J. Lamping, and V. Saraswat. 1999. Relating resource-based semantics to categorial semantics. In Mary Dalrymple (Ed.), 261–280. Earlier version published in *Proceedings of the Fifth Meeting on the Mathematics of Language*, Saarbucken (1995).

Dalrymple, M., R. M. Kaplan, J. T. Maxwell, and A. Zaenen (Eds.). 1995. *Formal Issues in Lexical-Functional Grammar*. Stanford, CA: CSLI Publications.

Dalrymple, M., J. Lamping, F. Pereira, and V. Saraswat. 1997. Quantification, anaphora and intensionality. *Logic, Language and Information* 6:219–273. appears with some modifications in Dalrymple (1999), pp. 39-90.

Dalrymple, M. E. 1993. *The Syntax of Anaphoric Binding*. Stanford CA: The Center for the Study of Language and Information.

de Groote, P. 1999. An algebraic correctness criterion for intuitionistic multiplicative proof-nets. *TCS* 115–134. URL: `http://www.loria.fr/~degroote/bibliography.html`.

Deal, A. R. 2007. Property-type objects and modal embedding. URL: `http://people.umass.edu/amyrose/`.

Girard, J.-Y., Y. Lafont, and P. Taylor. 1989. *Proofs and Types*. Cambridge University Press. URL: `http://www.cs.man.ac.uk/~pt/stable/Proofs+Types.html`.

Gupta, V., and J. Lamping. 1998. Efficient linear logic meaning assembly. In *COLING/ACL 98*. Montréal. URL: `http://acl.ldc.upenn.edu/P/P98/P98-1077.pdf`.

Hindley, J. R., and J. P. Seldin. 1986. *Introduction to Combinators and λ-Calculus*. Cambridge University Press.

Jacobson, P. 1999. Towards a variable-free semantics. *Linguistics and Philosophy* 22:117–184.

Jäger, G. 2005. *Anaphora and Type Logical Grammar*. Springer.

Jaśkowski, S. 1963. Über Tautologien, in welchen keine Variable mehr als zweimal vorkommt. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 9:219–228.

Kaplan, R. M. 1987. Three seductions of computational psycholinguistics. In P. Whitelock, M.M.Wood, H. Somers, R. Johnson, and P. Bennet (Eds.), *Linguistics and Computer Applications*, 149–188. Academic Press. reprinted in Dalrymple et al. (1995), pp. 337-367.

Kaplan, R. M. 1995. The formal architecture of LFG. In M. Dalrymple, R. M. Kaplan, J. T. Maxwell, and A. Zaenen (Eds.), *Formal Issues in Lexical-Functional Grammar*, 7–27. CSLI Publications.

Kokkonidis, M. 2008. First order glue. *Journal of Logic, Language and Information* 17:43–56. URL: `citeseer.ist.psu.edu/kokkonidis06firstorder.html`.

Kuhn, J. 2003. *Optimality-Theoretic Syntax–A Declarative Approach*. Stanford CA: CSLI Publications.

Lamarche, F. 1994. Proof nets for intuitionistic linear logic 1: Essential nets. Technical Report, Imperial College.

Lambek, J. 1965. The mathematics of sentence structure. *American Mathematical Monthly* 58:154–170.

Lev, I. 2006. Anaphora in glue semantics.

Lev, I. 2007. *Packed Computation of Exact Meaning Representations using Glue Semantics (with automatic handling of structural ambiguities and advanced natural language constructions)*. PhD thesis, Stanford University. URL: `http://www.stanford.edu/~iddolev/pulc/current_work.html`.

Mackie, I., L. Román, and S. Abramsky. 1993. An internal language for autonomous categories. *Applied Categorical Structures* 1:311–343.

Marantz, A. 1984. *On the Nature of Grammatical Relations*. Cambridge MA: MIT Press.

Melliès, P.-A. 2008. Categorical semantics of linear logic: a survey. unpublished draft, URL: `http://www.pps.jussieu.fr/~mellies/`.

Mints, G. 1981. Closed categories and the theory of proofs. *Journal of Mathematical Sciences* 15:45–62. Russian original published 1977.

Moot, R. 2002. *Proof-Nets for Linguistic Analysis*. PhD thesis, University of Utecht. URL: `http://www.labri.fr/perso/moot/`, also `http://igitur-archive.library.uu.nl/dissertations/1980438/full.pdf`.

Partee, B. H. 2006. Do we need two basic types. In H.-M. Gaertner, R. Eckardt, R. Musan, and B. Stiebels (Eds.), *Puzzles for Manfred Krifka*. Berlin. URL: `www.zas.gwz-berlin.de/40-60-puzzles-for-krifka/`.

Perrier, G. 1999. Labelled proof-nets for the syntax and semantics of natural languages. *L.G. of the IGPL* 7:629–655. URL: `http://www.loria.fr/~perrier/papers.html`.

Pollard, C. to appear. Hyperintensions. To appear in *Journal of Logic and Computation*. URL: `http://www.ling.ohio-state.edu/~hana/hog/pollard2006-hyper.pdf`.

Restall, G. in preparation. Proof theory and philosophy. chapters available at `http://http://consequently.org/papers/ptp.pdf`. An earlier version of this was 'Proof and Counterexample'.

Troelstra, A. S. 1992. *Lectures on Linear Logic*. Stanford CA: CSLI Publications.