Yet Another Attempt to Explain Glue

Avery D Andrews
ANU, Mar 13, 2010

'Glue semantics' is certainly the nearest thing there is to an 'official' theory of compositional semantics for LFG, but suffers from the problem that a great many practitioners of LFG syntax profess not to understand it. This is unfortunate for a variety of reasons, not the least of which is that with the addition of glue, LFG finally attains Richard Montague's standard of adequacy for grammatical theories, that the overt forms of sentences be explicitly connected to structures that can support a mathematical definition of entailment.[1] Conceptually, this puts LFG on a par with current fully formalized theories such as HPSG and Type-Logical Grammar, but the effect is certainly diluted if most LFG practioners don't understand how it works.

Glue furthermore provides an additional level of representation, which can be strictly binary branching if this is desired, which can be used to account for various phenomena that don't fit easily into the f-structure format. Scope of quantifiers, negation, various adverbs, etc. are some classic examples; more recent ones are Andrews' 2007b proposals about complex predicates, where glue is used to implement Alsina's (1997) concept of 'predicate composition', and Sadler and Nordlinger's (2008) work on the structure of NPs in Australian languages, where glue is used to differentiate interpretational differences that don't appear to be supported in the f-structures. This might provide a bridge between LFG and other frameworks such as the Minimalist Program, where binary-branching trees that reflect semantic composition play a central role.

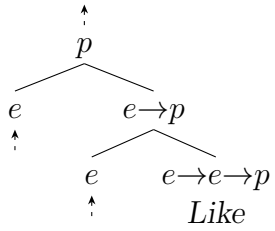
This presentation of glue will at first be streamlined and modified in certain respects from the standard one, as presented for example in Dalrymple (2001) and Asudeh (2004), but the differences will be explained later. Sections (1.1.-2.4.) are hopefully reasonably self-contained, and even attempt to introduce some basic ideas of formal semantics to people who aren't familiar with them, but 3.2. and later are at a significantly higher level (and have been worked over less in the interests of accessibility), so that after looking at 3.1. it is probably time to take on some other glue literature before proceeding. Asudeh (2005a, 2008) would be a good choices, alongside the longer Dalrymple and Asudeh works cited above. Several aspects of the presentation used here, 'prefab glue', are discussed more formally in Andrews (2008a).

But before launching into the details, I'll say a bit about the fundamental motivations. A basic idea of compositional semantics is that the meanings of sentences and other possibly complex utterances are produced by combining the meanings of the individual words and morphemes, and perhaps certain kinds of configurations (such as double object constructions, which often seem to be associated with the idea of the first object

[1] Which is, it seems to me, the most empirically straightforward aspect of the program of 'truth conditional semantics', whatever one thinks about the philosophical issues. There were proposals for semantic interpretation of LFG prior to glue, but they did not deal with the full range of phenomena involving quantifier scope, as discussed briefly in Andrews and Manning (1993).

getting or losing the second). It is an attractive idea to conceptualize this by representing the individual meaning contributions as pieces, which are then literally assembled. So for example, given a sentence such as *Bert likes Ernie*, we might regard the verb and two nouns as introducing meaning pieces that we could represent like this (or in all sorts of other ways):

(1)



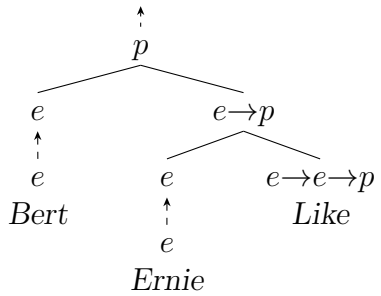
b.

c.

The little inbound arrows represent 'argument' positions into which meanings are to be plugged; the outbound one 'output' positions which can either be plugged into an input, or deliver the meaning of the entire assembly, and the labels represent semantic types, e for 'entity', p for 'proposition', and $a \rightarrow b$ for something that takes an a for an input and produces b as output, rightmost parentheses omitted.

So if we suppose that the inner/lower argument of *Like* represents the 'Likee', the outer/higher the 'Liker' (following a general convention motivated by the idea that more active arguments are more external to the meaning of verbs than less active ones), then the assembly that represents the meaning of the sentence can be produced by plugging outputs into argument positions in an appropriate way, getting:

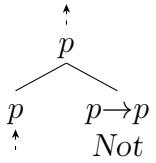
(2)



If this was all there was to semantic assembly, compositional semantics would probably not exist as an independent discipline, but language exhibits a wide variety of phenomena that require a more sophisticated treatment.

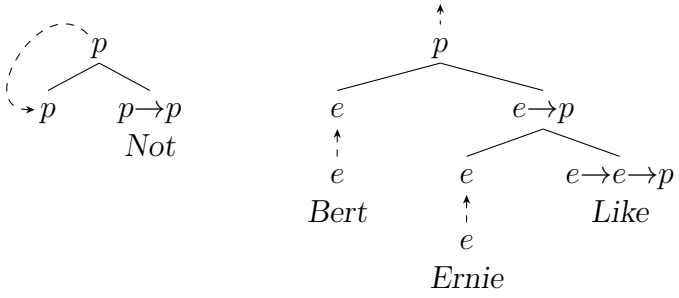
Negation, for example, can be understood as an operation that applies to a proposition to produce another proposition, yielding a structure piece like this:

(3)



The sensible way to combine (2) with (3) is to plug the output of (2) into the input of (3), but if we just obey rules of plugging 'likes into likes' (p into p, e (for 'entity') into e), there is another possibility, which does not appear to represent any coherent meaning:

(4)



The necessity of ruling out perversities like (4) means that we can't just naively make up kinds of structural diagrams and hope that everything will work out, but need to rely on mathematical work, which, fortunately, turns out to have already been done.

So we are going to work out ways of combining pieces into bigger structures, but what exactly is the point of this activity? There are various things that are often said about the goals of 'formal semantics', but I think that the goal that is the most straightforward and easy to describe is that of producing a mathematically solid theory of 'entailment':[2] the relation that exists between a collection of assertions C and some further assertion A when any sensible person who accepts C is *compelled* to accept A (if they don't, they're bonkers, and not worth trying to talk to about anything). So if somebody accepts these two sentences:

(5) a. All men are mortal

b. Socrates is a man

They are generally thought to be hopeless if they reject this one:

(6) Socrates is mortal

[2]Not to deny that there's more; entailment is simply the clearest and arguably most important.

Since Richard Montague showed in the 1970s that setting up theories of entailment for natural language was a viable enterprise, it has been generally accepted that the right way to do this was to associate sentences with mathematically defineable structures, using some kind of formal grammar, and then define entailment over the structures, using techniques from formal logic. The structures are essentially expressions in a formal language, with a simpler and more uniform syntax than is apparently the case for natural language.

'Glue semantics' is then one of many techniques for connecting sentences in natural languages to such structures. In spite of the rather straightforward nature of the simplest applications, it has some fairly extensive mathematical underpinnings and involvements; one can think of LFG glue as a little raft floating on a fairly deep lake: it is a matter of choice how far down one attempts to peer (and there is not necessarily anything very rewarding for linguistics down in the darker depths). In this exposition, we start with the simplest ideas, those relevant for 'predicate argument structure', and then move on to the somewhat more sophisticated world of quantifiers, and, finally, look at the much more subtle issues that arise in dealing with anaphora.

1. Basic Glue: Argument Structure

1.1. Semantic (Combinatorial) Types

We start by considering the nature of the 'semantic types' used in formal semantics, which are fundamental to glue, and have already been introduced as the p's and e's, representing propositions and entities, respectively. These can be thought of as a system of ontological categories that limit the ways in which concepts can coherently be combined.[3] Since the main function of these types appears to be to control combinations, rather than actually say anything about what the meanings are, it would also be sensible to call them 'combinatory types'.[4]

Theories of semantic/combinatory types start with the idea of an inventory of 'basic types', together with some operations for combining these to produce complex/derived types. In linguistics, it is common to recognize at least two basic types, entities, which we have already symbolized as e, and 'propositions', which we symbolize as p rather than the more common choise t, following Pollard (2007). There might be many more types, for reasons discussed for example in many works by Jackendoff and others, e.g. Jackendoff (2002), such as *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, not covered here,[5] LFG glue is generally thought to need at least the distinction between e and p, so we'll follow the tradition and assume these two.

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

[4]Not least because they have quite a lot to do with a subject called 'combinatory logic'.

[5]But discussed in Andrews (2008a).

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 $a \rightarrow b$.[6] This is the type of something 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*, or *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 $p \rightarrow p$. The implicational types can in fact be regarded as mathematical functions, which operate on a meaning of one type, to produce a different meaning of some other type. For this reason, they can also be called 'functional types'.

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 $e \rightarrow p$. So what about *likes* itself? Well, this is something to which one supplies an e, and gets an $e \rightarrow p$, so it can be put into the type $e \rightarrow (e \rightarrow p)$.

This is really a rather clever idea. The Ancient Greeks, such as Aristotle, had a pretty good grasp of the workings of the types e and $e \rightarrow p$ (and some others), but never came up with $e \rightarrow (e \rightarrow p)$, so their logic stalled (for more than 2000 years) on the question of what to do about transitive verbs. So they had a good story about deductions from intransitive clauses (the sentence below the line is supposed to follow from the ones above it):

(7) All men are mortal
Socrates is a man

Socrates is mortal

But their system couldn't account for a deduction involving both the subject and object of a transitive verb, such as:

(8) Every American likes every Muppet
George is an American
Ernie is a Muppet

George likes Ernie

This problem wasn't overcome until the late 19th century (by C.S. Peirce and Gottlob Frege, working independently), using a significantly different treatment of transitive

[6] In systems where there is only one type-constructor, this is often omitted, and $<a, b>$ is written rather than $a \rightarrow b$.

verbs, which people who haven't done 'Predicate Calculus' in a basic course in logic shouldn't worry about now.[7]

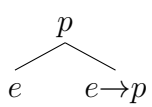
We can use this idea to get various more types; for example, a ditransitive verb such as *give* would be $e\rightarrow(e\rightarrow(e\rightarrow p))$, while a 'propositionally ditransitive' verb such as *tell* in *John told Mary that she was sick* would be of type $p\rightarrow(e\rightarrow(e\rightarrow p))$.

The parenthesis-nests associated with multi-argument verbs can get confusing, so people often follow a convention of omitting rightmost pairs of parentheses, so that the three types above become $e\rightarrow e\rightarrow p$, $e\rightarrow e\rightarrow e\rightarrow p$, and $p\rightarrow e\rightarrow e\rightarrow p$. For computer programmers, these conventions have the pleasant property of being identical to the Haskell notation for declaring the types of functions.

A significant 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 (Jackendoff (1973) is an especially important early example) provides evidence for a non-arbitrary arrangement of the arguments, so that the Theme (thing conveyed) would be innermost, then Recipient, and finally Agent. Marantz (1984) argues that this hierarchy should be thought as a matter of 'order of application', with the less active arguments applied first. I suggest that it would perhaps be better to see it as a hierarchy of tightness of structural bonding, 'earlier application' being tighter bonding. Regardless of how it's construed, it provides a non-arbitrary answer to how the arguments should be ordered, in most (but not all) cases. ('John predeceased Mary' versus'Mary survived John' seem to constitute an example where no substantive principle seems to be available to determine the order of arguments).

1.2. Expressions

The types are supposed to represent the possibilities for expressions to combine, so the next step is to provide some formulation of how to combine them. Implicational types are thought of as things with a 'hole' in them, of some type, into which something else of the same type is to fit, so what we want is an expression combining things of these two types into one, producing an expression of the appropriate resulting type. For example, combining something of type e with something of type $e\rightarrow p$ to produce something of type p. In logic, people are concerned with writing expressions as linear strings ('formulas'), but for linguistics, we really only want the parse-trees for the formulas. The obvious tree for combining an e with an $e\rightarrow p$ is:

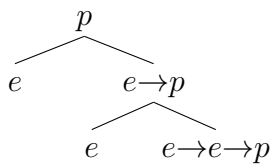
(9)


[7]For those who have, however, the representations used here, called 'Curried', are equivalent to the 'n-place predicates' in first order logic, though it might take a beginner a certain amount of pondering to see this.

Well, obvious except for our decision to put the functional type second rather than first in the tree; as things work out, this seems to be on the whole best for layout, although the decision is arbitrary (and the trees could in fact be regarded as unordered).

And for a transitive verb, we'd have something like this:

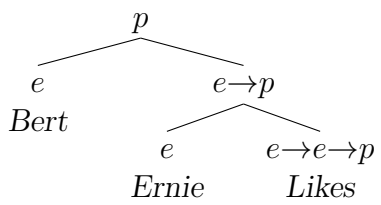
(10)



Following Marantz's idea, the inner e position would be associated with the less active semantic role (for example, Liked), the outer one, the more active (Liker). This is in conformity with standard ideas about the organization of 'argument-structure', for example Alsina (1996), as discussed in Andrews (2007b).

To produce a standard 'logical form', we could write in the expressions under the leaf-nodes indicating their type, e.g.:

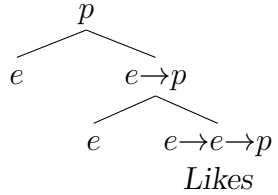
(11)



But our goal is not merely to represent combinations of meanings,[8] but to also say something about how the one that is meant by the speaker can be figured out by the hearer, on the basis of the grammatical structure of the sentence.

So suppose we have a collection of words/meanings of various types. From the types alone, we can produce pieces of the structures they will fit into:

(12) a. e *Ernie* b. e *Bert* c.



Rules for producing the tree-pieces from the types can be described as follows:

[8]Note the phrasing: we are not claiming to represent meanings, but only how they are to be combined, since the meanings themselves are not given, only their English glosses in a special font. This is standardly called 'compositional semantics', sometimes 'combinatorial semantics', as opposed to 'lexical semantics', or, one could say 'full semantics', in which both the lexical and syntactic contributions to meaning are treated.

(13) a. Starter: the starter node has the type as its 'semantic type', and the meaning as its 'meaning-label'.

b. Atom: if the current (initially, starter) node has atomic semantic type, the construction is finished.

c. Implication: if the current node is of semantic type $a \rightarrow b$, make it the right-daughter of new node of semantic type b, with new left-daughter of semantic type a. Repeat (b-c) with the new mother as current (neither new node has a meaning label, although there is a way to assign meaning-labels to all nodes of fully assembled structures).

There is an important situation which these rules don't cover, which we will get to later.

What we'll do now is show how to assemble these 'prefab' pieces, so-called because they're put into the structure rather than the type format before rather than after assembly, into full structures.

Happily, we don't have to make it up by ourselves. There is a branch of logic concerned with 'proof-nets for linear logic', which addresses this issue, exactly. Even better, you don't have to know anything about linear logic and proof-nets to learn the rules and use them (though you will, thereby, wind up having learned something about linear logic and proof-nets).

One basic ingredient in the recipe is a concept of 'polarity', which is used to control how the trees are assembled, and which is responsible for the little inbound and outbound arrows from the introduction. The polarities are 'negative' (outbound) and 'positive' (inbound), and assigned to the tree-nodes by the following rules:

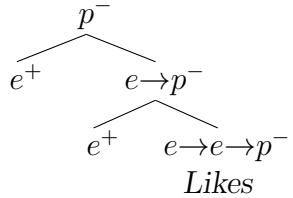
(14) a. Rightmost (meaning-bearing) leaf-nodes have negative polarity.

b. If an implication has negative polarity, so does its mother, but its (left) sister has positive polarity.

These polarity assignments may seem a bit odd for the purposes of linguistics, and have occasionally been reversed in the LFG glue literature, but they are standard in logic, and it's probably best to follow that tradition here.

(15) a. e^-
Ernie

b. e^-
Bert

c.


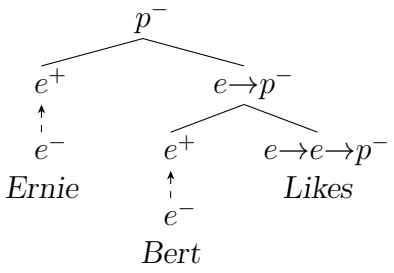
Note that (a) and (b) get negative polarity because they are rightmost (that they are also leftmost doesn't matter). It would be a good exercise to write out the tree for *tell* without the polarities, and then use the rule (14) to fill them in (answer in (130)). It should be apparent that both the polarities and the non-basic node-labels are predictable from the structure of the tree together with the basic/atomic labels. So our representation is rather redundant, but I think the redundancies are helpful for use, at least in the beginning. The next task is to describe a technique for assembling these and similar pieces.

What we're about to look at might seem unnecessarily complex, but its properties are motivated by the fact that it will be able to deal with somewhat more difficult situations we will encounter later, where naive, intuitive schemes are likely to be open to various kinds of abuse.

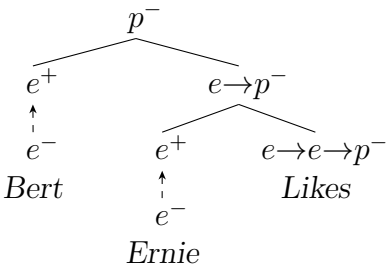
The basic idea is that we will assemble the pieces by running things called 'axiom links' from nodes with negative polarity to those with positive polarity that are labelled with a basic type. Nodes with basic type labels are called 'literals', and constitute most of the 'vertical frontier' of the tree, that is the nodes that either have no mother or no daughter (so far, the negative literal is at the top, with no mother, the positive literals are the various non-rightmost leaves, with no daughters). The only vertical frontier node that doesn't have to be a literal is the one that carries the meaning, the rightmost daughter. The axiom links themselves will be depicted as dashed arrows.

What we intend is that there be two possible assemblies for the pieces in (15):

(16) a.



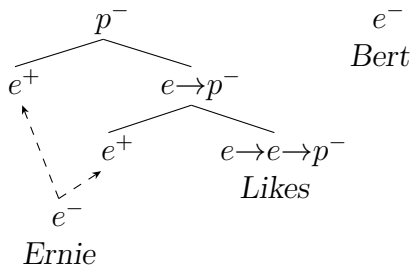
b.



These can be converted into the earlier expression format of (11) by erasing the polarities, and 'contracting' the axiom links, that is, fusing the two nodes connected by an axiom-link into one.

Typical configurations that we would want to exclude are non-connected positive literals (as for example in (15), where no axiom-links have (yet) been added, and 'multiple' connections, such as:

(17)



Here we want to reject the structure whether or not *Bert* is included or omitted. The left-hand tree might seem to be a good idea for the analysis of reflexive pronouns, but in fact, it doesn't seem to work out to try to do this in the first stages of meaning-assembly.

The basic principles that rule out these and various other kinds of bad results are:

(18) a. Matching: axiom-links go from negative to positive literals of the same type

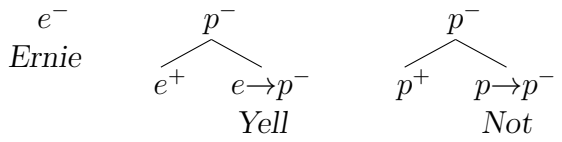
b. Monogamy: forming non-overlapping pairs

c. One Left Over: with one negative polarity literal left over, and all other literals in axiom-linked pairs

These rules forbid all possibilities but those in (16) for assembling the pieces in (15). Observe that the idea of contracting along the axiom links wouldn't produce very sensible results if we didn't obey (a-b). However, deviously minded individuals can come up with some additional possibilities for certain other collections of pieces.

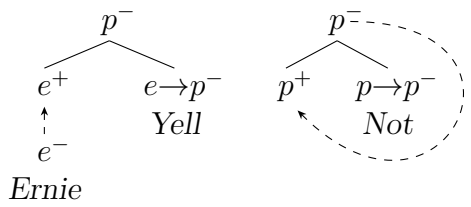
Consider for example some (hypothetical, oversimplified) component meanings for *Ernie doesn't yell*:

(19)



Here, in addition to the sensible assembly which you can hopefully come up with yourself (answer in (131)), there is also a perverse one that satsifies the conditions of (18):

(20)



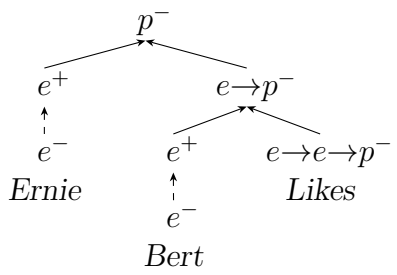
Here we've plugged the negative 'result' p of the *Not*-piece into its positive 'argument' p. To rule this out definitively, we need an explicit principle.

The one that we will use is called the 'Correctness Criterion for proof-nets', which exists in a very large number of different-looking but equivalent formulations.[9] The version we will use is based on de Groote (1999), and for the kinds of cases we've considered so far, can be impressionistically stated as: 'the assembled structure must be a tree'. But this is too vague for serious work, and also won't work in the most general case we would like to be able to do deal with, so we need to define a few things more explicitly.

Technically, the assemblies we're making are 'graphs',[10] which, for our purposes, can be thought as sets of 'nodes' (which can have various kinds of labels and other attributes) connected by 'directed arcs', which go from one node, the source of the arc, to another, its target. For our structures so far, the arcs are directed from the daughters of a node to the node itself (the mother), that is, from the nodes representing the antecedent of an implication and the implication itself to the one representing the consequent of that implication.

With the direction ('orientation') of the arcs indicated, assembly (16a) becomes:

(21)



The tree-nodes, with arcs directed as indicated, constitute a structure called the 'dynamic graph' in de Groote (1999). To formulate the Correctness Criterion, we need two more definitions:

(22) a. The 'Principal Inputs' are the rightmost leaves (the nodes with meaning-names such as *Ernie* and *Like*; it doesn't matter if they're also leftmost).

b. The 'Final Output' is the unlinked negative literal (there must be one and only one, according to the One Left Over principle of (18)).

Now we can state the criterion as follows:

(23) Correctness Criterion (clause 1): From every Principal Input, the dynamic graph must provide a path to the Final Output.

[9] To which the most accessible general introduction is probably Moot (2002:ch.4), although this would still be pretty hard for people who don't know a reasonable amount about logic.

[10] In one of several meanings of the term that are used in mathematics. This one is unfortunately quite different from what you probably studied in secondary school algebra, but is one of the standard ones studied in 'Graph Theory'.

This is 'clause 1' because we'll be needing a second provision before too long.

You should try to work through that the sensible assemblies all satisfy (23), while (20) does not. More discussion of (21) and (20) appears in (132). An essential characteristic of this system is that it requires that each meaning introduced by the grammatical and lexical structure of a sentence be used once and once only in its interpretation, unless special provisions are made to avoid this (which can happen, under certain circumstances). For example, if a sentence contains a negation, you can't understand it as positive, either by ignoring the negative, or by interpreting it twice to get a double-negation[11] (but overt multiple negations can be packed into one by LFG feature-unification, and thereby interpreted as a single positive).

We now have half of what will be a system for assembling 'typed meanings', that is, a combination of a meaning and its semantic type, into sensible combinations. But before producing the other half, we will look into how the grammatical structure of a sentence is to constrain its assembly.

1.3. Connecting to Grammar

Typed meanings already 'know' a fair amount about how they are to be put together. For example, there is only one way to assemble the pieces of (19). But for (15), there are two, so we must appeal to the syntax to account for why *Bert likes Ernie* and *Ernie likes Bert* aren't both ambiguous. In LFG, these two sentences would differ in their f-structure:

(24) a.
$$f:\begin{bmatrix}\text{SUBJ} & g:\begin{bmatrix}\text{PRED} & \text{'Bert'}\end{bmatrix}\\ \text{PRED} & \text{'Like(SUBJ, OBJ)'}\\ \text{OBJ} & h:\begin{bmatrix}\text{PRED} & \text{'Ernie'}\end{bmatrix}\end{bmatrix}$$

b.
$$f:\begin{bmatrix}\text{SUBJ} & g:\begin{bmatrix}\text{PRED} & \text{'Ernie'}\end{bmatrix}\\ \text{PRED} & \text{'Like(SUBJ, OBJ)'}\\ \text{OBJ} & h:\begin{bmatrix}\text{PRED} & \text{'Bert'}\end{bmatrix}\end{bmatrix}$$

LFG glue uses the f-structure to constrain assembly by building references to it into the typed meanings that are to be assembled.

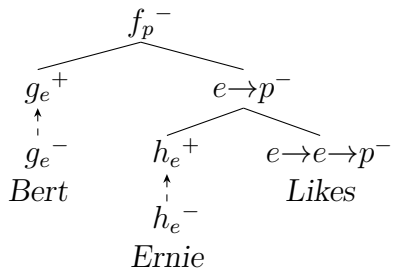
This is done by supplementing each literal with a reference to an 'f-structure correspondent'. We'll represent this by writing the type-literals as f-structure labels, subscripted with the semantic type. For (a) above, for example, the now-supplemented typed meanings might be:

(25) $\quad \textit{Bert} : g_e$
$\quad\quad \textit{Ernie} : h_e$
$\quad\quad \textit{Like} : h_e \rightarrow g_e \rightarrow f_p$

[11] This a linguistic version of the 'no copying, no deletion' principle of quantum logic, enlighteningly explored in Baez and Stay (2008). Luckly for linguists, glue seems to need only a miniscule fraction of the sophistication needed for physics.

If Matching is taken to require identity of f-structure labels as well as types, then there is only one way to put together the above typed meanings:[12]

(26)



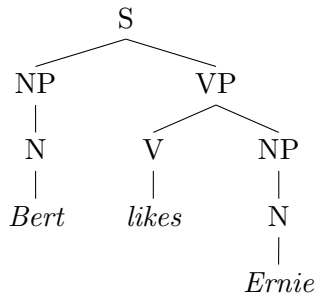
So, if we can get the syntax to control this supplementation of the literals, then the syntax will constrain the combination of the meanings. Fortunately, we can do this with the usual LFG technique of instantiation.

Suppose we include in lexical entries, alongside of the usual material, additional components called 'meaning-constructors', which will simply be typed meanings, with f-structure designators formed in the usual way with the ↑-arrow and GF names (and, sometimes, ↓ as well). Entries for the words in our current examples would be:

(27) *Bert* : $\uparrow_e$
Ernie : $\uparrow_e$
Like : $(\uparrow \text{OBJ})_e \rightarrow (\uparrow \text{SUBJ})_e \rightarrow \uparrow_p$

To produce the f-structure (24a), we'd use these lexical entries of the words in this tree:

(28)



And then, as a consequence, the functional designators in the meaning-constructors will evaluate to yield the f-structure labels used in (25).

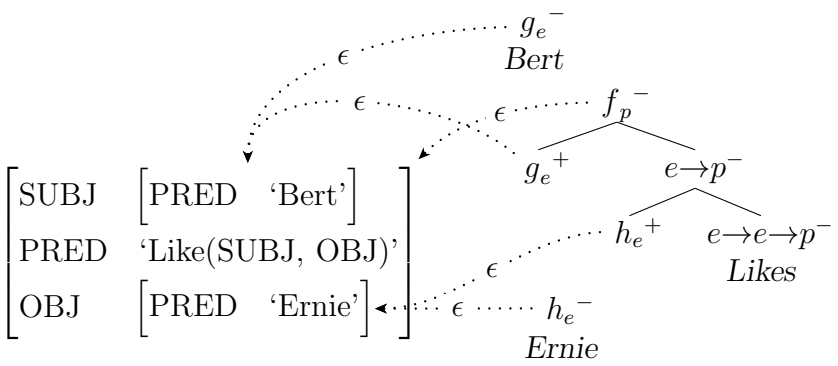
A slightly different way to look at what is going on here, which leads to a useful pictorial representation, is this. We can think of the functional designators and f-structure labels as defining a correspondence relation between the semantic combination tree on the

[12]We are only writing in the f-structure component for the literals, to reduce redundancy. Recall that a minimally redundant system would omit all non-literal type labels.

one hand, and the f-structure on the other. It has been customary since Kaplan (1987) to refer to correspondence between f-structure and semantics as 'σ', but since this one goes in the opposite direction from the f-structure-to-meaning direction that's usually assumed, we shall here called it ϵ, for 'expresses' (I've used σ in some previous work, ϵ in Andrews (2008a)). The use of an explicit name for a correspondence going in the glue-to-f-structure direction is a presentational innovation of this approach, but I believe that it is implicit in the argument of Asudeh and Crouch (2002b) that glue-assemblies should be regarded as a level of linguistic structure. We also innovate in leaving out the 'semantic projection' that is usually included in presentations of glue, for reasons discussed in Andrews (2008a).

The regular LFG grammar will now generate an f-structure together with a collection of meaning-constructors, whose literals are connected to various parts of the f-structure by the ϵ relation:

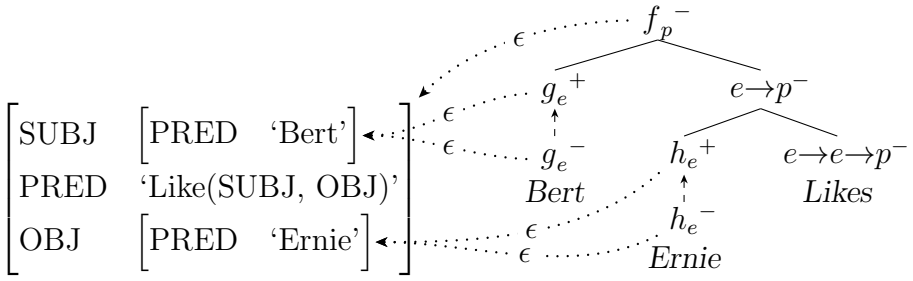
(29)



The constructors are at this point just floating around, attached to the f-structure by the ϵ-links, but not yet assembled into a coherent meaning.

The final step is to assemble them, so that all of the rules are satisfied, including the new one that axiom-linked nodes must be linked to the same piece of f-structure. For (29), there is only one way to do this:

(30)



Matching of the ϵ-correspondent means that if we contract the axiom-links, ϵ has a unique value for each merged literal, and so is a function, just like the ϕ-correspondence from phrase-structure to f-structure, but running in the opposite direction, from meaning to f-structure rather than overt form to f-structure.

It is hopefully evident that if the sentence had been *Ernie likes Bert*, then only the other sensible assembly would work, so we have achieved our goal of getting the syntax to constrain the meaning. But, so far, we haven't used the glue system to represent anything that isn't fairly clearly represented in the f-structure. Ultimately, we will do this by considering quantifiers and scope, but, first, we need to make a detour into a somewhat logical topic, the 'lambda-caluclus'. But before doing this, I will close this section by noting that Andrews (2007a, 2008b) describe an alternative technique for producing the meaning-constructors from the syntactic structure that is less reliant on the c-structure than the (standard) one used here, but instead derives them from the f-structure using a 'Semantic Lexicon'.

2. Intermediate Glue: Quantifiers

2.1. Substitution and 'Lambda-calculus'

So far we've merely specified how meanings can be put together, based on the idea that some meanings are functions which apply to other meanings, without saying anything substantive about them. This is very likely a wise policy, given the high degree of controversy that exists concerning what meanings really are. Nevertheless, it is useful to have some way of saying more about the meanings that are being assembled than just giving them names such as *Like*, and a very popular tool for doing this is something called the 'lambda-calculus', which, aside from its popularity, has the further virtue of being mathematically quite closely related to glue, in ways we will look at later in this introduction.

One way to motivate our approach is to consider the problem of explaining the relationships between different uses of lexical items. It is clear that the intransitive and transitive use of *boil* in these examples have different but related meanings:

(31) a. The water boiled

b. John boiled the chicken

If we know what *boil* in (a) means, we can do a reasonable (alhough probably not complete) job of describing the meaning of (b) as follows:

(32) [John put the chicken$_i$ in boiling water for some time.]$_j$
Because of this$_j$, after this$_j$, the chicken was not like it was before.

This is in a modified version of Wierzbicka's Natural Semantic Metalanguage (NSM),[13] where we've used subscripts to indicate coreference, and assumed the availability of *put*,

[13]Presented in for example Goddard (1998, 2008), Wierzbicka and Goddard (2002), and the NSM homepage at `http://www.une.edu.au/bcss/linguistics/nsm/semantics-in-brief.php`)

boiling and *water* as either semantic primitives (rather unlikely) or previously defined concepts ('molecules'). To get a general sense of transitive *boil* that that can be applied to any appropriate kind of thing, such as perhaps leather, eggs, etc, it is standard NSM practice to formulate an 'explication' like this:

(33) X boiled Y
X put Y in boiling water for some time
Because of this, after this, Y was not like Y was before.

Here I've left out the brackets and subscripts, because the 'variables' X and Y seem to suffice for the disambiguations.

So then the idea of substitution is that to get a meaning for a particular sentence, you need to plug in 'real meanings' for the X and Y positions. Some general discussion of what is involved in doing this for NSM can be found in Andrews (2006).

The standard tool used in formal semantics for describing substitutions of this sort is the 'lambda-calculus', which can be regarded as a technique for supplementing a formalized language with two additional constructions, 'abstraction, and 'application'. An application consists of a 'function' and an 'argument', usually written in that order, with the argument in parentheses:

(34) $f(a)$

function argument

The negative polarity implications in our glue-trees are in fact just syntax trees for applications, with the function written on the right rather than the left, for convenience of the layout for line-drawing.

An abstraction on the other hand consists a 'head' and a 'body', where the head consists of the symbol λ and a variable, and the body is a formula in which the variable may appear any number of times, including none:

(35) $\lambda x.b$

head body

We don't yet have abstraction in our glue-structures, but soon, we will.

The basic idea of substitution is that it's stupid (under some circumstances) to write something like this:

(36) $(\lambda x.b)(a)$

where we've created an abstraction with body b and variable x, and applied it to argument a. Rather, this combination of the two constructions should be replaced by

a simpler format in which neither appears, but, instead, a replaces all 'free' occurrences of x in b, 'free' being a technical concept that we'll have a bit to say about later. A very simple example is that $(\lambda x.Yell(x))(Bert)$ should be replaced by $Yell(Bert)$ (people often put parentheses around the lambda-abstraction, to help clarify what is supposed to apply to what, even when it's not really necessary). For a more complicated example, work out what $(\lambda x.Not(Yell(x)))(Bert)$ should be replaced by (answer in (133)).

To describe this more precisely, we formulate a rule called 'β-reduction', which, in words, says that the result of applying a lambda-abstraction to an argument in the application construction is the same thing as the body of the abstraction, with the argument substituted for all 'free' occurrences of the variable that appears in the body of the abstraction. In symbols, one writes:

(37) $(\lambda x.b)(a) \quad \Rightarrow_\beta \quad [a/x]b$

The idea of the $[a/x]b$ notation is that one 'divides out' x and 'multiplies in' a, in b. Textbooks such as Hindley and Seldin (1986) provide a rigorous definition of this notation, but I don't think we need that here.

So what makes an occurrence of a variable x 'free'? Nontechnically, it must sit neither in a head, nor in a body whose head is 'λx'. So, for example, in an expression such as $(\lambda x.Dog(x))(x)$, only the third and last occurrence of x is free. A good technique for avoiding problems with free versus non-free is to use a different variable for each λ in all of the abstractions that are going to be used in a discussion. Then, an occurrence of the variable that occurs in the head of given lambda-sub-expression will be free if it occurs outside that expression, not free (bound) if it occurs within it.

Finally, we need to connect lambda-calculus to types. There are two general kinds of lambda-calculus, 'typed', and 'untyped'. The latter doesn't have types, and is therefore a bit simpler to formulate, but its meaning and behavior can be rather subtle. In the typed lambda calculus, every expression has a type, and the formulations are a bit more complicated, but the actual behavior easier to understand. First, there is an implicational type constructor, which is connected to the application and abstraction constructions as follows:

(38) a. If f is of type $A \rightarrow B$, and a is of type A, then $f(a)$ is of type B.

b. If x is of type A, and b is of type B, then $\lambda x.b$ is of type $A \rightarrow B$.

The typing rule for (a) is clearly the same as the one we've used for our negative-polarity implications, which are, so far, the only ones we have. Indeed we can say that these are the applicational construction in a system of formulas, with only superficial differences from the standard application notation in the lambda-calculus. We don't have abstraction in these structures yet, but this will change.

But we can still use abstraction to provide some integration of our compositional semantics with meaning-definitions for lexcial items. One way of presenting this idea is

to convert the structures produced by glue into application structures in the lambda-calculus notation (this step is not in fact necessary, but does provide a convenient way to demonstrate the equivalence of the two kinds of expressions), and then use β-reduction to implement the substitutions.

We can do this is with a 'semantic labelling' procedure whereby lambda-terms in the meaning language are assigned to glue-tree nodes in a bottom-up manner, following the dynamic graph (that is, the tree-structure). So far, the leaves of the tree/dynamic graph will be only be rightmost daughters with semantic labels, so the following rule will suffice:

(39) a. The semantic label of a negative implication is that of its right daughter applied to that of its left daughter.

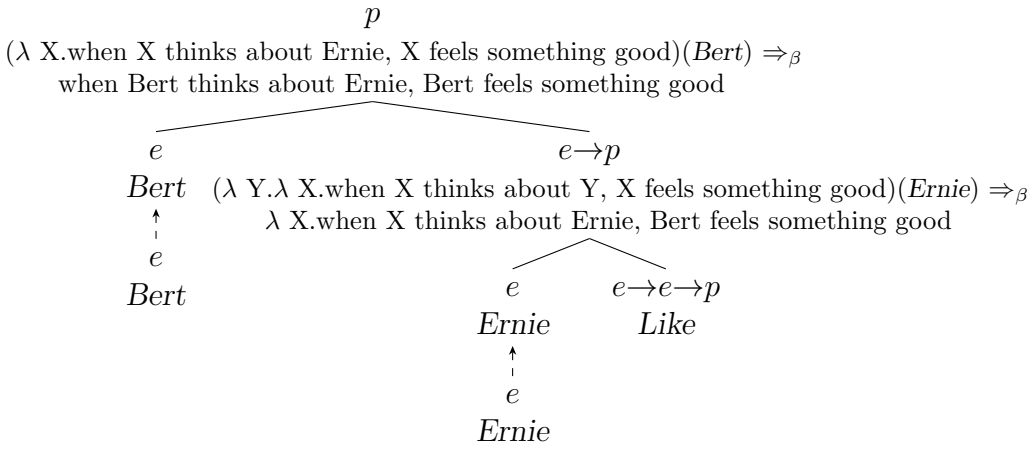
b. The semantic label of the target of an axiom-link is that of its source.

To illustrate the procedure, suppose we've decided that the meaning of *Like* is:[14]

(40) when X thinks about Y, X feels something good

Then we could produce an explication for *Bert likes Ernie* as follows, taking some notational shortcuts to fit things onto the page:

(41)



Actually doing substitutions into a natural language NSM involves a variety of problems of agreement, case-marking, etc, briefly discussed in Andrews (2006), but hopefully the basic idea is viable.

Before moving on from this illustration of β-reduction, I'll say something about what the types do. Any given expression in the lambda-calculus might provide any number of opportunities to apply β-reduction. It turns out that if we keep applying it, we

[14]There are problems with this, for example what if somebody you like has just been diagnosed with terminal cancer? But it will do for illustration.

eventually use them all up and can do no more, and furthermore get the same final result no matter what order we chose to do them in (you can check this for (41)). These two facts are very pleasant, and the situation is highly analogous to what happens with fractions, which can always be reduced to lowest terms: the presence of an opportunity to perform β-reduction is analogous to the presence of some common factor in the numerator and denominator of a fraction, and the application of β-reduction is analogous to reducing the fraction by removing this factor from both places. It turns out that without the types, this no longer holds, and it is possible to produce expressions where the application of β-reduction just produces another such opportunity, and the attempted reduction never ends. The fact that β-reductions in a typed system eventually have to stop is sometimes described by saying that they are 'terminating', and that the final result doesn't depend on exactly what order they are applied in by saying that they are 'confluent'. Termination and confluence are very nice properties for any system of intuitive simplifications over representations.[15]

It should also be clear that we only have to do the semantic labelling and substitutions if we want to make use of the information provided by the definitions of the lexical items: if all we're interested in is how the lexical meanings are to be combined, the glue tree itself provides all the relevant information. We have now introduced the idea of lambda-abstraction for describing lexical meanings; now we consider how it arises in purely compositional semantics.

2.2. Quantifiers and Scope

Many ambiguities are reflected in f-structure in a fairly obvious way, but the 'scope ambiguities' in sentences such as these are not:

(42) a. Everybody doesn't yell

b. Everybody loves somebody

In LFG, the f-structures for these sentences would be, respectively:

(43) a.
$$\begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{'Everybody'} \end{bmatrix} \\ \text{POLARITY} & \text{NEG} \\ \text{PRED} & \text{'Yell(SUBJ)'} \end{bmatrix}$$

b.
$$\begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{'Everybody'} \end{bmatrix} \\ \text{PRED} & \text{'Love(SUBJ, SUBJ)'} \\ \text{OBJ} & \begin{bmatrix} \text{PRED} & \text{'Somebody'} \end{bmatrix} \end{bmatrix}$$

[15] When he was in elementary school, the author would have been horrified at the idea that he would ever describe any property of fractions as 'pleasant' or 'nice', but, nevertheless, this is the case.

Each structure represents both meanings, and a satisfactory way to represent scope phenomena in f-structure itself has not yet been devised.[16]

Although it would not be crazy to suggest that scope is handled in a rather different fashion from the aspects of linguistic structure that are encoded in f-structure, there are nonetheless various phenomena that depend on scope. For example, the indefinite pronouns *somebody* and *anybody* differ that *somebody* resists being within the scope of negative, while *anybody* demands to be in the scope of a negative or certain other kinds of operators. Therefore, in these examples, (a) prefers wide scope for its quantifier object NP, while (b) requires narrow:

(44) a. John doesn't like somebody

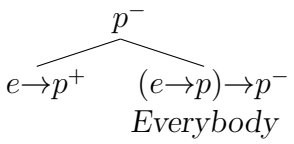
b. John doesn't like anybody

Glue normally uses the 'generalized quantifier' analysis of Barwise and Cooper (1981), which can in fact be seen as a somewhat upgraded analysis of the treatment of quantifiers in Classical Antiguity. The basic idea is that a quantificational NP such as *everybody*, *somebody* or *nobody* is a 'predicate of properties', that is, something that is true or false of things of type $e \rightarrow p$, that is, something that is of type $(e \rightarrow p) \rightarrow p$. *Nobody*, for example, is true of a property *Yell* just in case the intersection of the collection of yellers with the collection of people is empty, while *Everybody* is true of *Yell* just in case the people are a sub-collection of the yellers (you might have to think about this for a bit to see it).

A typed meaning for *everybody* will therefore be:

(45) *Everybody* : $(e \rightarrow p) \rightarrow p$

Converting this into our tree-format with the rules of (13), and assigning polarities, we get the structure:

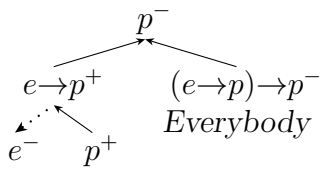
(46)


But now we're stuck, because we don't know what to do with the implicational left-daughter (so far, all left daughters have been literals). What we'll do is simply give the answer, then discuss it.

[16] Andrews and Manning (1993) might be the most conscientious attempt, but nobody has attempted to base any further work on it. Other proposals tend to fail to deal with (b), since there is only one clausal level in the f-structure, while we really need two to represent the ambiguity. For (a), things aren't so clear, since the presence of the auxiliary gives a biclausal analysis some degree of plausibility.

In the first place, we'll trivially reformulate our rules so that they build the tree and assign polarities at the same time. So, in the process of tree-building, the implicational left-daughter of (46) will get positive polarity. Then we decree that a positive implication will expand (downward) to a negative 'left pseudo daughter', labelled with the antecedent of the implication, and a positive 'right daughter', labelled with the consequent. The left daughter is described as 'pseudo' because the link that connects it to its mother is *not* in the dynamic graph, while the link that connects the right daughter to the mother is in the dynamic graph, oriented upwards. We represent this difference by connecting the left pseudo-daughter to its mother with a dotted line (oriented here from mother to daughter). So then (46) becomes (47), where we've also indicated the dynamic graph directionality of the solid links:

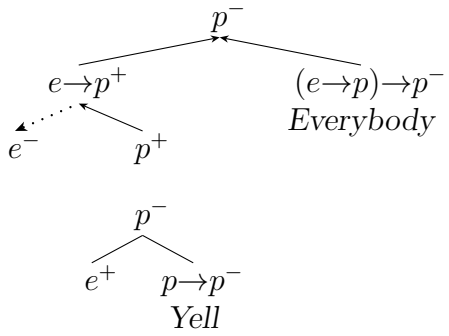
(47)



So how do we combine these new structures with other things?

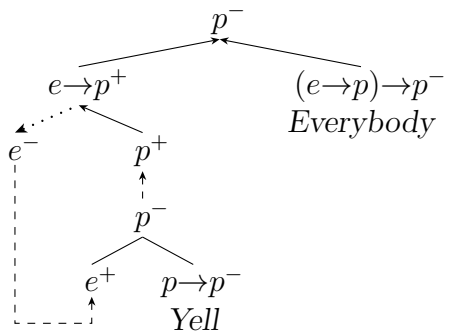
We'll first look at the simplest case, for which the machinery is a bit cumbersome, but in ways that will be essential a bit later. Since *Everybody* is of type $(e{\to}p){\to}p$, we should expect it to apply to something of type $e{\to}p$, such as, for example, *Yell*. So we have these two trees, arranged arbitrarily but conveniently for what comes next:

(48)



Now, the assembly rules say that we're supposed to run axiom links from positives to negatives of matching type, subject to the Correctness Criterion, and the only way to do this for (48) is like this:

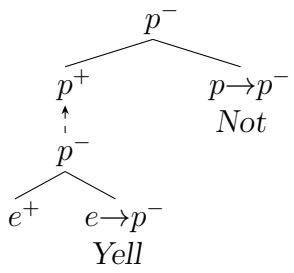
(49)



This could seem a bit excessive, because we're putting in two axiom-links to express one predicate-argument relation, namely, that *Yell* is the argument of *Everybody*. More will be said about this later.

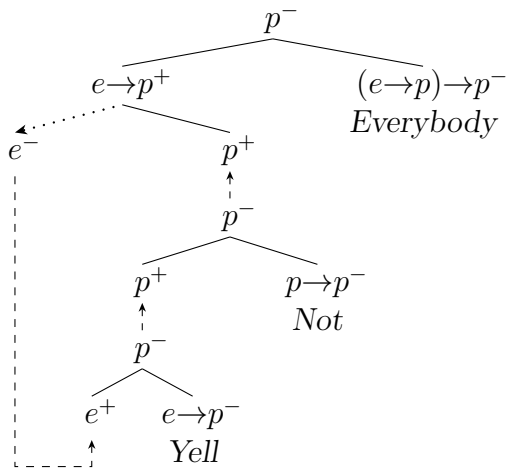
But for a more complicated example where this technique pays off, suppose we first put together typed meanings for *not* and *yell*, like this:

(50)



Then we can combine this with the *Everybody* constructor like this:

(51)



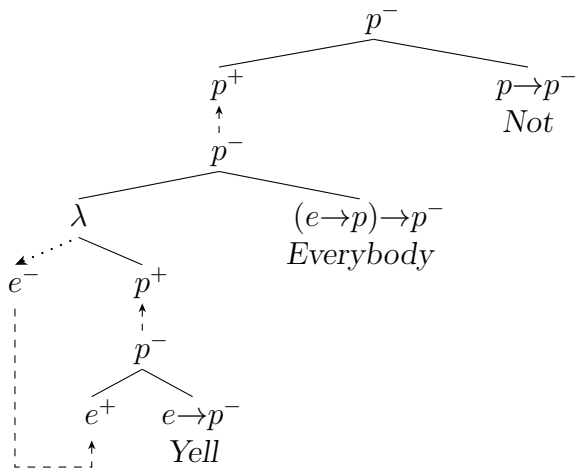
The intended sense of this is that everybody has the property of not yelling (wide-scope reading of the quantifier).

To motivate this intuitively, think of the negative e^- of the quantificational pronoun as shooting out 'test entities', which enter the sub-assembly (50), which returns 'true'

or 'false' (perhaps relativized to a 'possible world' or some other kind of context) to its top, un-axiom-linked negative p. If, whenever the test entity is a person, the returned value is 'true', then, the quantificational assembly (51) represents a true proposition, otherwise, a false one.

But there is another way of putting the pieces together, which delivers the narrow scope reading. Fir this, first combine the quantifier directly with the verb, as we did in the simple example, and then apply the negative to the result of doing that. But since the positive implications have the same type-assignment rule as lambda-abstractions, and their function is essientially the same, it is convenient to label them simply with λ, giving this for the narrow-scope reading:[17]

(52)



This corresponds to shooting the test entities into the *Yell* predicate, and then applying negation to the result of the quantification.

So we seem to have an intuitively workable representation, but we have also created some problems. One of which is that our structures really are not trees anymore, due not only to the pseudo-daughter links, but also because of some things that happen in more complex configurations, discussed in Andrews (2008a). For this reason, we will stop calling them 'trees', and instead, consistently call them 'glue-structures', a term we have already used occasionally.

A more substantial problem is that once we have positive polarity implications, it turns out that we need an additional clause to the Correctness Criterion:

(53) Correctness Criterion (clause 2): From every left-pseudo-daughter of an implication, the dynamic graph must provide a path to the right-daughter (and, therefore, equivalently, to the implication itself).

[17] People familiar with the lambda-calculus might well note that this logical form is not 'η-reduced', an issue we will consider later.

As a relatively difficult exercise, you might try to devise a structure that satisfies the principles so far except for (53), but doesn't make sense (one is provided in Andrews (2008a)).

Another perspective emerges from the 'semantic readings' of the previous section. Our representations of the quantifier meanings are rather 'austere' compared to what one would find in any logic or formal semantics course. In introductory logic, for example, a quantified NP such as *everybody* would probably be read as a combination of the 'universal quantifier' $\forall$, and the 'material implication' $\supset$, which obeys principles making it roughly equivalent to the '*if...then*' construction. In formal semantics, on the other hand, it is common to find quantifiers in a 'tripartite' construction where they relate a variable and two propositions. These possibilities are illustrated here, where Φ is the formula which is supposed to be true of everybody:

(54) a. $(\forall x)(Person(x) \supset \Phi)$

b. $Every(x, Person(x), \Phi)$

We can use either of these decompositions in a semantic reading method by explaining what to do with the positive implications.

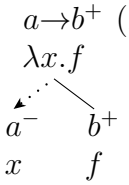
What we do is add the following two principles:

(55) a. the semantic reading of a left pseudo-daughter is a 'new' variable (not yet used in the construction of a semantic label).

b. the semantic reading of a positive implication is the lambda-term with the variable assigned to its left pseudo-daughter in its head, and the reading of its right daughter as its body.

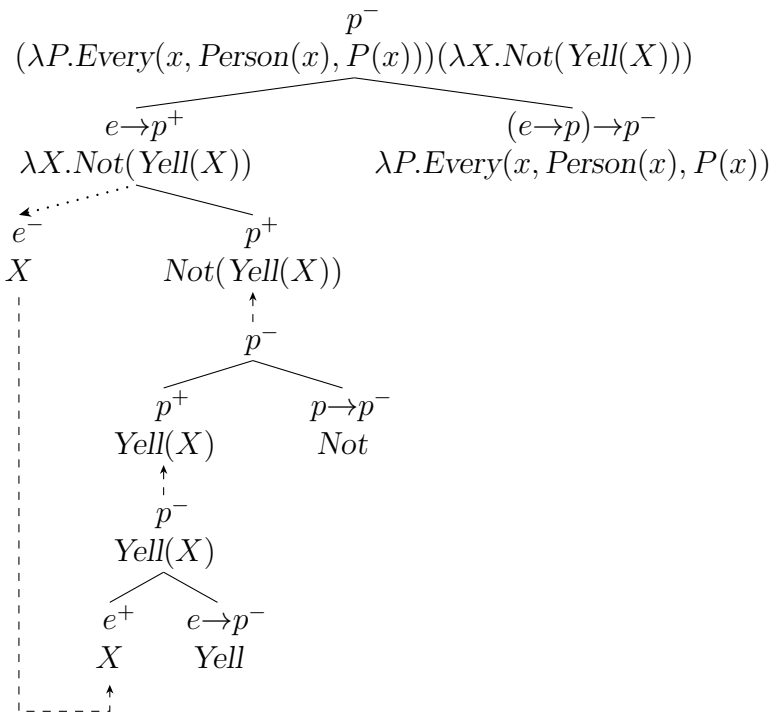
We can represent the intended effect of these rules schematically like this:

(56)



Given these rules, the semantic reading for (51) can be calculated as follows, assuming the tripartite structure representation for the quantifier:

(57)



Working out the final reading of the top p^- is a bit of a chore, because it requires two β-reductions:

(58) $(\lambda P.Every(x, Person(x), P(x)))(\lambda X.Not(Yell(X))) \Rightarrow_\beta$
$Every(x, Person(x), (\lambda X.Not(Yell(X)))(x)) \Rightarrow_\beta$
$Every(x, Person(x), Not(Yell(x)))$

As with the semantic readings for negative implications, those for positive ones aren't needed until we're doing something with the internal structure of the meanings of the constructors, since if we're not, the glue trees themselves are a perfectly adequate representation of the semantic composition. And, at present, we have no clear reason for assuming that the result at the end of (58) is truly a better representation than $Every(\lambda x.Person(x))(\lambda y.Not(Yell(y))))$, in spite of its greater popularity and perhaps intuitive intelligibility (I'm not sure how much the latter depends on what you're used to).[18]

So next, we move on to the problem of connecting quantifiers to the grammatical structure. Here the issue is that the propositional literals connect to some higher structure. Often this is the clause immediately containing the quantifier NP, but not always:

(59) John thinks that Mary likes everybody

[18]But the latter representation involves an issue of 'η-reduction', which we'll consider later.

This sentence has a (less natural) reading where, for every person x, John thinks that Mary likes x.

The treatment of this that I will use here is based on the 'propositional glue' system of Andrews (2008a), which is mathematically simpler than the ones used in the previous glue literature.[19] The basic idea is to use local names and 'inside-out functional uncertainty' (iofu) paths to connect the type p literals to the f-structure. An iofu path is a description of a path through the f-structure that climbs 'backwards' up a chain of grammatical functions (from value of, say, SUBJ to whatever it is that takes that value as its SUBJ-value), while a local name is a label, usually an uppercase letter preceeded by %, applied to the destination of some path in a rule, for re-use within the rule, but not more widely.[20] So here we see a meaning-constructor together with an equation relating a local name to the position of the quantifier, using an iofu path:

$$\begin{array}{ll} (60) & \mathit{Everybody} \ : \ (\uparrow_e \multimap \ \%H_p) \multimap \ \%H_p \\ & \qquad (\mathrm{GF}^*\uparrow) \ = \ \%H \end{array}$$

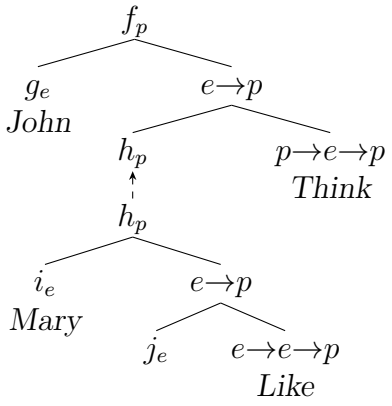
The local name $\%H$ can represent any piece of the f-structure which, as stated by the iofu equation underneath, can be gotten to by following an upward path of grammatical functions from $\uparrow$.

To see what (60) does, first consider what (59) gets as functional and glue structure without the quantificational pronoun (omitting the polarities, since supplying them should be routine by now, and representing ϵ with f-structure labels in the glue-structure):

$$(61) \quad \left[\begin{array}{ll} \mathrm{SUBJ} & g{:}\left[\mathrm{PRED} \quad \text{'John'}\right] \\ \mathrm{PRED} & \text{'Think(SUBJ, OBJ)'} \\ f{:} & \left[\begin{array}{ll} \mathrm{SUBJ} & i{:}\left[\mathrm{PRED} \quad \text{'Mary'}\right] \\ \mathrm{COMP}\, h{:} & \left[\begin{array}{ll} \mathrm{PRED} & \text{'Like(SUBJ, OBJ)'} \\ \mathrm{OBJ} & j{:}\left[\mathrm{PRED} \quad \text{'Everybody'}\right] \end{array}\right] \end{array}\right] \end{array}\right]$$

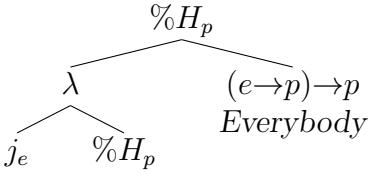
[19] Girard's 'System F' in most recent glue literature, First Order Linear Logic in Kokkonidis (2008), which also provides considerable background information and useful discussion.

[20] For a careful presentation of these, see Dalrymple (2001:145-148).



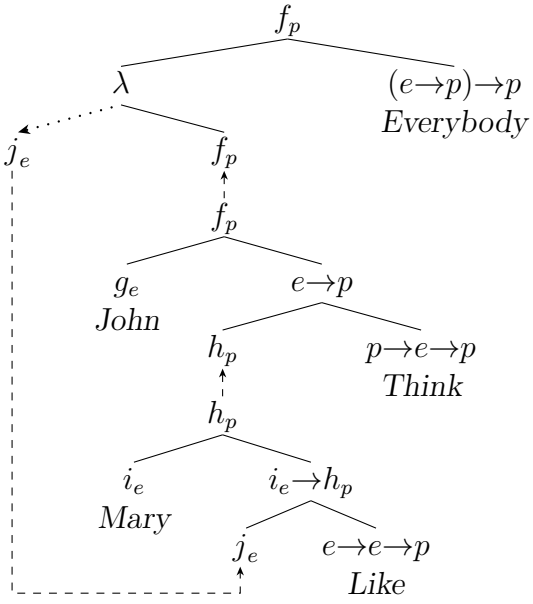
By what we've said so far, the tree-form of the quantifier constructor will be:

(62)



j_e here will have to be axiom-linked to j_e in (61), but for the $\%H$-literals, things are more complicated, since $\%H$ can be instantiated to either f or h. Either of these choices provides a sensible reading, the former wide-scope, the latter narrow. Here is the former:

(63)



As an exercise, write out the structure for the narrow-scope reading.

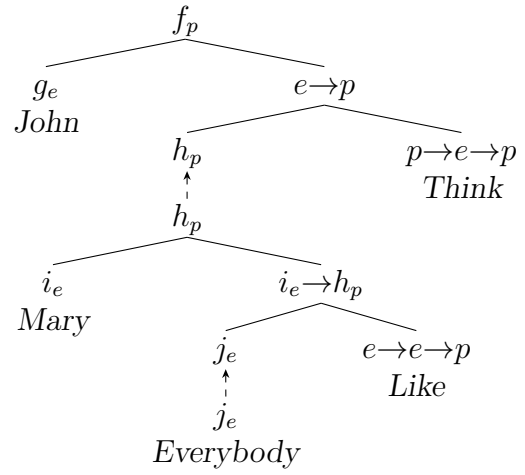
In this example, the scope ambiguity is produced by two different instantiations of the propositional literal, but in the ambiguity of *everybody loves somebody*, there is only

one possible instantiation for this literal. But there are two possible arrangements of the quantifiers, leading to two readings.

A final point about the quantifiers that is worth considering is this. The ambiguities they induce can be rather subtle and hard to see, so it would not be crazy to suggest that they arise at a relatively late level of linguistic processing that is not in fact always performed. The present treatment is actually consistent with this, thanks to some observations by Gupta and Lamping (1998).

Suppose that, in a preliminary assembly, we simply ignore the propositional literals, and pretend that they're not there. The quantifier will then act, for purposes of assembly, as if it was merely an expression of type e, and will fit into the tree of (61) like this:

(64)



This provides an 'underspecified' representation for both readings, and one of Gupta and Lamping's points is that if we need a more specific representation, we can derive it from (64) or equivalent by choosing some f-structure correspondent for the propositional literals, and then 'splicing in' the *Everybody* constructor to the appropriate position. There are also various clever tricks one can use to record the viable possibilities, but going into that is more a topic in Computational Linguistics than basic glue.[21]

2.3. Common Nouns and the Meaning-side

The somewhat arduous work we've done with β-reduction on the meaning-side begins to pay off when we look at how common nouns fit in with quantifiers. We need to account for the relationships between quantificational pronouns such as *everybody*, and quantified full NPs such as *every dog*, which we can now do, essentially by lambda-abstracting on two property-variables rather than just one.

We start by considering NPs such as *every dog*. It is a fairly standard analysis of common nouns that they are of type $e \rightarrow p$, on the basis that an entity might be, or not

[21] Some other approaches to underspecified representations with significant relationships to this one are Crouch and van Genabith (1999) and (Copestake et al. 2005) (Minimal Recursion Semantics).

be, a dog. So, in a sentence such as:

(65) Every dog barks

the effect of the quantificational determiner seems to be to compare the collection of entities that are dogs with the collection of entities that are barkers, and return 'true' if the former is included in the latter, 'false' otherwise. Other quantificational determiners, such as *some* and *no*, seem to effect similar comparisons of the 'extensions' of properties, that is, the collections of thing that those properties are true or false of. *some* for example, appears to require that the extensions overlap, *no* that they don't.

The semantic type of these quantificational determiners therefore seems to be:

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

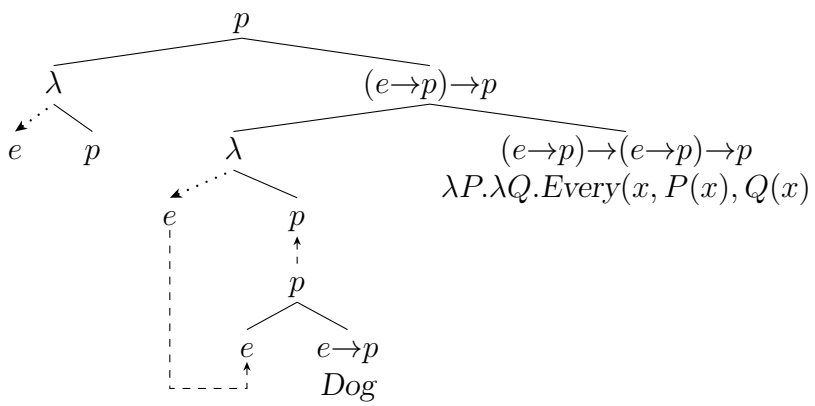
This would tend to be intimidating to people without much background in formal semantics, so we'll walk through it. According to what we've been saying, the meanings of quantificational determiners appear to be relations between properties (indeed, ones that depend only on their extensions, which is really quite interesting). So by the 'Currying' technique we've been using, of applying arguments one-at-a-time, their type would be something like **property**$\rightarrow$**property**$\rightarrow p$. But 'properties' are of type $e{\rightarrow}p$, so this is 'just' (66).

Next, it is plausible to associate the first of these properties with the nominal, the second with the sentence in which the common noun is used (on the general grounds of applying the 'closest' thing first). Given our previous work, and assumping the tripartite representation for the quantifier, this will be the appropriate internally-structured meaning-side:

(67) $\lambda P.\lambda Q.Every(x, P(x), Q(x))$

Now suppose this appears in a structure like this:

(68)



The semantic reading for the innermost λ-node is $\lambda X.Dog(X)$ (we consider equivalent all variants of this which differ only in their choice of novel variable for X). This ought to mean just the same thing as *Dog*, but β-reduction won't effect this identification. There is another principle, called η-reduction, which we'll see later, that will, but we don't really need it, because when we apply this as argument to the *every* meaning, we get:

(69) $(\lambda P.\lambda Q.Every(x, P(x), Q(x)))(\lambda X.Dog(X)) \Rightarrow_\beta$
$\lambda Q.Every(x, (\lambda X.Dog(X)(x), Q(x)) \Rightarrow_\beta$
$\lambda Q.Every(x, Dog(x), Q(x))$

Now, *every dog* has assembled to produce something with the same potential for further assembly as *everybody*, so that the treatment of sentences such as *every dog doesn't bark* will follow what we've already done.

This would be a good time to discuss some discrepancies between the glue-structures and the lambda-calculus. One is that in the glue-structure notation, each negative antecedent (of a positive implication, essentially a λ, is associated with *exactly one* position in the consequent of that implication. In the lambda-expressions produced by our labelling procedure, this cashes out to the requirement that the head of each lambda-expression bind exactly one occurrence of a variable in its body, a restriction which is called 'linearity'. This creates problems in the treatment of anaphora, which we will discuss later, but makes the computations simpler. Another, much less significant difference, which we've already commented on, is that in the lambda-calculus, there are going to be many effectively equivalent versions of any abstraction, different simply in which variable is used to represent the binder and bindees.

So for example we don't want to posit any significant difference between $\lambda x.Like(x)(x)$ and $\lambda y.Like(y)(y)$. This is standardly achieved by defining a concept called 'α-equivalence', whereby the above two formulas are α-equivalent, but, for example, neither is α-quivalent to $\lambda x.Like(x)(y)$ (where the pattern of free vs bound variables and what binds them changes). Since glue-structures are a form of graph, the issue of α-equivalence just doesn't arise. One might think that it would be a straightforward matter to generalize the format of glue-structures to ones not obeying the linearity restrictions, and perhaps it actually is, but I'm not aware of any presentation where this is actually done in a careful and easy-to-understand way.[22] So lambda-expressions now emerge is a representation that is more powerful than glue-structures, but also having the annoying issue of α-equivalence.

[22]Two corners of the literature for people interested in this to poke around in are 'proof-nets for exponential linear logic' and 'interaction nets', but neither of these areas are for beginners or the faint-hearted.

2.4. Intersective Adjectives

We are now set up to treat 'intersective' adjectives such as *crazy*, as might be found in an example such as *every crazy dog barks*. The idea of intersective adjectives is that they apply, as it were, jointly, to the referent of the NP along with the head nominal. So something is a 'crazy dog' if and only if it is a dog, and also crazy. Some non-intersective adjectives are *putative* and *self-styled*:

(70) a. A putative goose is in the backyard.

b. A self-styled physicist announced a water-burning engine.

There's no such thing as just being 'putative', and a putative goose might not actually be a goose (but, perhaps, merely a duck). Similarly for a self-styled physicist.

So how to accomplish this intersective interpretation. For reasons presented in Dalrymple (2001:ch.10), it seems to be necessary to use two meaning-constructors. The first is associated with the lexical item and simply expresses a property:

(71) $\mathit{Crazy} \;:\; \uparrow_e \rightarrow \uparrow_p$

Some motivation for this constructor is that it is very close to what we want for the predicative use of these adjectives. If for example we follow the usual LFG analysis of treating predicate adjectives as APs bearing the XCOMP grammatical function, all we need is the following variation of (71):

(72) $\mathit{Crazy} \;:\; (\uparrow \mathrm{SUBJ})_e \rightarrow \uparrow_p$

Adjectives such as *asleep* that can occur as predicate adjectives but not attributives will have entries with the glue-side of (72) but not (71) (see Coppock (2008) for recent discussion of these).

But now, if we just introduce an intersective attributive adjective under an AP introduced with the ADJUNCTS GF, as is generally assumed, then $\uparrow$ in (71) will instantiate to the f-structure correspondent of this AP, producing the following f-structure and instantiated constructors:

(73)
$$g\!:\begin{bmatrix} \mathrm{PRED} & \text{'Dog'} \\ \mathrm{ADJUNCTS} & \left\{ h\!:\begin{bmatrix} \mathrm{PRED} & \text{'Crazy'} \end{bmatrix} \right\} \end{bmatrix}$$

$$\begin{aligned} \mathit{Dog} &\;:\; f_e \rightarrow f_p \\ \mathit{Crazy} &\;:\; g_e \rightarrow g_p \end{aligned}$$

In order for this to assemble into anything, we need a further constructor, which can be plausibly associated with the PS rules (although Dalrymple includes it in the lexical entry of the adjective, as an additional constructor).

The idea is that the head N is already providing one property, the AP another, and what this additional constructor is doing is combining them with logical 'and', often symbolized as $\wedge$ (among various other possibilities). If we suppose that these adjectives are introduced as AP under $\overline{\text{N}}$ recursively, this rule will do the job:

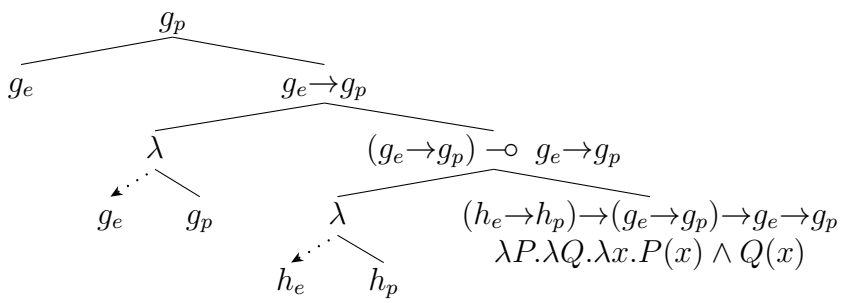
(74) $\overline{\text{N}} \quad \rightarrow \quad \begin{array}{c} \text{AP} \\ \downarrow \in (\uparrow \text{ADJUNCTS}) \\ \textbf{intersective} \end{array} \quad \begin{array}{c} \overline{\text{N}} \\ \uparrow = \downarrow \end{array}$

intersective:
$\lambda P.\lambda Q.\lambda x.P(x) \wedge Q(x) \ : \ (\downarrow_e \rightarrow \downarrow_p) \rightarrow (\uparrow_e \rightarrow \uparrow_p) \rightarrow \uparrow_e \rightarrow \uparrow_p$

A walkthrough of how this rather intimidating-looking thing works might be useful.

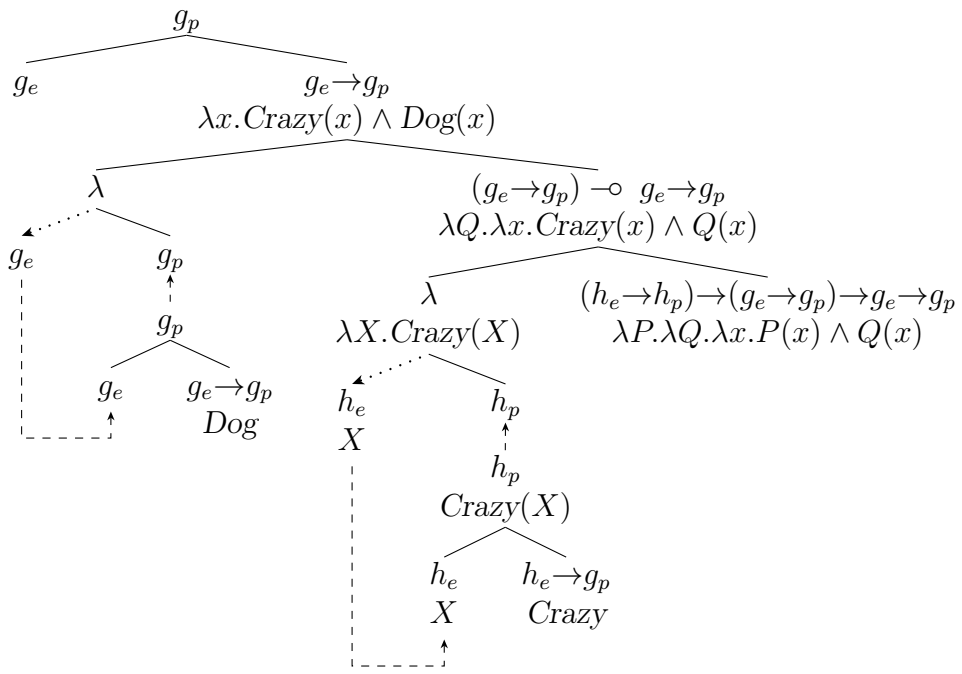
Expanding **intersective**, with some typical instantiations, to a tree, we get (whether the adjective or the noun should be innermost is an arbitrary decision):

(75)



Here, we've put the f-structure information into the complex node labels, as well, to make the construction easier to follow. Combining this with the *crazy* and *dog* constructors, we get:

(76)



β-reduced semantic readings are provided for some but not all of the nodes. You can supply those for the upper λ subassembly as an exercise.

Writing out these tree structures can be a bit of a chore, but when you understand what is going on, you can abbreviate things considerably by first writing out the instantiated constructors in the standard format:

(77)
$$\begin{aligned} Dog &: f_e \multimap f_t \\ Crazy &: g_e \multimap g_t \\ \lambda P.\lambda Q.\lambda x.P(x) \wedge Q(x) &: (g_e \multimap g_t) \multimap (f_e \multimap f_t) \multimap f_e \multimap f_t \end{aligned}$$

and then drawing in axiom-links:

(78)
$$\begin{aligned} Dog &: f_e \multimap f_t \\ Crazy &: g_e \multimap g_t \\ \lambda P.\lambda Q.\lambda x.P(x) \wedge Q(x) &: (g_e \multimap g_t) \multimap (f_e \multimap f_t) \multimap f_e \multimap f_t \end{aligned}$$

Two problems with this method are (a) there isn't any convenient place to write in the semantic readings (b) one can make mistakes in checking the Correctness Criterion. If this is a problem (as it might be in a sufficiently complex example), one can just draw the trees (or write out the proofs in the conventional format, as we will get to soon).

The attributive constructor is complex, but similar in form to the one used for apposition in Australian language NPs by Sadler and Nordlinger (2008), so working through

this should set you up to manage that. A more extensive treatment of analysis of adjectives, including modal adjective such as *former* and *self-proclaimed*, as well as adverbial modification of adjectives (*an obviously daft proposal*) is provided in Andrews (2009).

3. Relating to the Literature

3.1. Notational Interlude

Conceptually, we've gone through enough material to actually read a glue paper, but there are still some notational issues to deal with.

The first is that most, but not all, glue presentations use the symbol $\multimap$, colloquially read 'lollipop', instead of $\rightarrow$, for implication. When the lollipop appears, one can assume that the logic being used is 'linear', instead of 'classical', as will be explained in the next section. But this really has nothing to do with the nature of implication *per se*, but rather with the general environment in which it is working, so that the use of the lollipop could be regarded as unnecessarily intimidating (and $\rightarrow$ is used in some literature, such as Troelstra (1992) and Lev (2007)).

The second issue is the 'semantic projection'. We've been supposing that the ϵ-correspondents of glue literals are f-structures, but it is normally assumed that they are residents of a special 'semantic projection' σ, that comes off f-structure, and has a rather small number of attributes used for semantic bookkeeping. Indeed, it seems to me to hardly do anything at all, so I propose dispensing with this level in Andrews (2008a), and various previous discussions.[23]

The semantic projection is constructed as usual by the meaning-constructors, so that for example (74) becomes (also using t for p):

(79) $\overline{\mathrm{N}} \quad \rightarrow \quad \underset{\downarrow \in (\uparrow \mathrm{ADJUNCTS})}{\mathrm{AP}} \quad \underset{\uparrow = \downarrow}{\overline{\mathrm{N}}}$

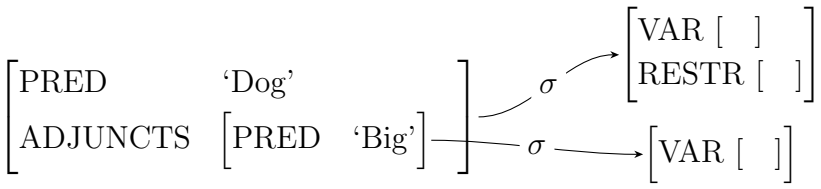
intersective

intersective:

$$\lambda P.\lambda Q.\lambda x.P(x) \wedge Q(x) \ : \ ((\downarrow_\sigma \mathrm{VAR})_e \multimap \downarrow_{\sigma t}) \multimap ((\uparrow_\sigma \mathrm{VAR})_e \multimap (\uparrow \sigma \mathrm{RESTR})_t) \multimap (\uparrow_\sigma \mathrm{VAR})_e \multimap (\uparrow_\sigma \mathrm{RESTR})_t$$

Given a $\overline{\mathrm{N}}$ such as *big dog*, this PS rule will produce the following combination of f-structure and semantic projection:

[23] See also Kokkonidis (2008).

(80) 

The ϵ-correspondents of the glue literals will then be occupants of this new projection, rather than f-structures.

Notice that the semantic projection is quite disconnected, and could be viewed as being merely a place to park certain attributes such as VAR and RESTR, which aren't traditionally found in f-structure, and don't have the same kind of motivation as the things that are. Indeed, there is a bit of a question as to what the motivation for these attributes is in the first place.

One possible motivation is that it might be bad to have glue literals of different types, such as e and p, connected to the same syntactic object, such as the f-structure of the subject. So literals of type e are connected to the value of a VAR attribute on the semantic projection, while those of type t (for which we've been using p), are connected to either a semantic projection itself or to a RESTR (for 'restriction') attribute on this projection.

However, so far at least, there is hardly any solid argument for this (an argument presented by Asudeh (2005b) is countered in Andrews (2008a)), and it seems to be mostly esthetics and some traditions. So here, the semantic projection will be dispensed with. But, given a good argument for it, there would be no problem in re-introducing it into the present approach, although the structure diagrams would become more complex, with the semantic projection objects sitting between the f-structures and the glue trees.

Also recall that in the standard presentation, the ϵ projection is entirely implicit in the co-labelling between glue-proof literals and semantic projection structures, although its existence is sometimes exploited, for example in Crouch and van Genabith (1999) and Asudeh and Crouch (2002b).

We can now read and use constructors, except for analyses that use 'tensors' (mainly, anaphora), but the way in which we are using them is still different from the standard presentation, since most of the glue literature uses deductions rather than proof-nets, and our format for proof-nets is also non-standard. So deductions come next on the list of topics.

3.2. Glue by Natural Deduction

There have unfortunately been two schemes of deduction used in glue. The older glue literature used a conceptually immensely important[24] but very unintuitive system called 'Gentzen Sequents', which we'll ignore here (see Crouch and van Genabith (2000) when you want to know more about them), but then mostly shifted over to the 'tree style Natural Deduction' format, used for example by Dalrymple and Asudeh. To which we now turn.

Superficially, the Natural Deduction formulation is quite different from our present glue-structure format, but they are really the same thing, except for the not-so-minor point that the glue-structure format represents only proofs in what is called 'η-long normal form'. To explain this, we have to discuss certain aspects of how deductive proofs can be presented.

Natural deduction is a family of similar techniques of formal logical deduction developed between the 1930's and the 1960's to provide a more intuitive approach than the 'Hilbert systems' that were first developed to formalize modern logic (see Pelletier (1999) for a historical discussion). The basic idea was to formalize the kinds of thought-processes actually used in normal mathematical arguments, which are quite different from the structure of proofs in a Hilbert system. Here we will be concerned with only one form of Natural Deduction, Gerhard Gentzen's 'tree style', and only one connective, implication, which we'll write as $\rightarrow$.

Natural Deduction works by providing, for most connectives, a pair of rules, one to eliminate that connective, another to introduce it. Implication elimination is the classic rule of 'Modus Ponens', whereby, it is held that if one assents to 'if A then B', and also to 'A', one ought also to assent to 'B' (if you don't, you might be deemed not worth trying to have a sensible discussion with). We can see why it's called elimination of implication, since one of the two premises, the so-called 'major premise' has an implication connective at its top level, which is gone in the conclusion (along with the minor premise and the antedecent). In tree-style, the rule would typically be written like this:

(81) $$\frac{A{\rightarrow}B \qquad A}{B}\,\mathcal{E}{\rightarrow}$$

The premises appear above the line, the conclusion, which is supposed to follow from them, below it, and the name of the rule justifying this reasoning step appears to the side, being $\mathcal{E}$ for eliminations, $\mathcal{I}$ for introductions, followed by the connective being managed (there are many other notations for the justifications). There seems to be a tradition of putting the major premise on the left, though this doesn't matter.

ND deductions are produced by combining such rule applications, using the conclusion

[24] See for example Melliès (2008), the earlier sections of which are quite accessible to people with a smattering of logic, although the mathematical level rises inexorably.

of one application as one of the premises of another. Below, for example, we derive C from A, B, and $A{\rightarrow}B{\rightarrow}C$:

(82)
$$\dfrac{\dfrac{A{\rightarrow}B{\rightarrow}C \qquad A}{B{\rightarrow}C}\,\mathcal{E}{\rightarrow} \qquad B}{C}\,\mathcal{E}{\rightarrow}$$

At this point you might have noticed that these deductions look very much like glue-assemblies without lambdas, with the axiom-links contracted, but put rightside-up, on their roots, rather than hung upside-down, as linguists tend to like their trees. Especially, the relationships between the types are the same.

This highly non-accidental resemblance becomes stronger when we introduce the introduction rule. The idea behind this rule is that if we can derive B from the assumption A and perhaps some others, then we can derive $A{\rightarrow}B$ from those other assumptions alone, without A. So, for example, if we can prove 'the world is doomed' from the premises 'Bush is a moron' and 'Ahmedinejad is a maniac', then we can we can prove 'if Bush is a moron, then the world is doomed' from the premise 'Ahmedinejad is a maniac'. The rule is formulated like this:

(83)
$$\dfrac{\begin{array}{c}[A]^i\\ \vdots\\ B\end{array}}{A{\rightarrow}B}\,\mathcal{I}{\rightarrow}^i$$

It is implicit in this format that there may be other assumptions involved in the justification for B, and the superscripted brackets indicate that the assumption so-bracketed is 'discharged', that is, no longer needed to derive the conclusion. The superscript on a bracketted assumption is shared with the introduction step which discharges that assumption.

This formulation is temporally telescoped, since it should be regarded as the output resulting from a prior stage of the derivation that looks like this:

(84)
$$\begin{array}{c}A\\ \vdots\\ B\end{array}$$

But no information is lost by the notation, due to the superscripting of the introduction-step with the premise that it discharges. For the portion of the proof-tree that is above the discharge step will depend on the premise (i.e. the brackets and superscript are to be ignored), while the portion below does not (the brackets are noted as indicating that the premise is no longer needed to derive the current conclusion).

With both the introduction and elimination rule, we can begin to prove a wider variety of results. A very simple proof is:

(85) $$\frac{[A]^1}{A\to A}\,\mathcal{I}\to^1$$

What we've done here is taken A of (85) to represent both A and B above the line in (84), with the vertical dots representing a derivation with 0 steps, and then discharged A, producing a 'theorem' which often goes by the name 'I' (for 'Identity'). By 'theorem' we mean a formula derivable in the system that depends on no premises, since we have discharged the only premise used.

This might seem a bit puzzling, since it is perhaps not entirely clear that the rule (84) is meant to be used this way: the graphic representation of Natural Deduction falls somewhat short of full explicitness. Nevertheless, it's so convenient that people use it anyway, tending to use other formulations only when a higher degree of formal rigor is required.[25]

A more complicated example is this derivation of $B\to A\to C$ from the premise $A\to B\to C$:

(86) $$\dfrac{\dfrac{\dfrac{A\to B\to C \qquad [A]^1}{B\to C}\,\mathcal{E}\to \qquad [B]^2}{C}\,\mathcal{E}\to}{\dfrac{A\to C}{B\to A\to C}\,\mathcal{I}\to^2}\,\mathcal{I}\to^1$$

If we discharge the remaining assumption, we get a theorem called 'C', easy to remember due to its connection with Commutativity. It is written out in (135).

However one area in which the interpretation of our graphically formulated rules is not entirely clear in what is often called the 'discharge policy'. The basic question is *how many* instances of an assumption can be discharged by a single step. Consider this putative proof:

(87) $$\dfrac{\dfrac{\dfrac{A\to A\to B \qquad [A]^1}{A\to B}\,\mathcal{E}\to \qquad [A]^1}{B}\,\mathcal{E}\to}{A\to B}\,\mathcal{I}\to^1$$

Here we've discharged two instances of the assumption A in one step. In Classical Logic, this is allowed, but in the Linear Logic it isn't, and each $\mathcal{I}\to$ step can only discharge one instance of a given hypothesis. A consequence is that (88) is a theorem of Classical Logic but not Linear:

[25] But a fully rigorous presentation of tree-style Natural Deduction is provided in Jäger (2005).

(88) $(A{\rightarrow}A{\rightarrow}B){\rightarrow}(A{\rightarrow}B)$

Systems that have this as a theorem (called 'W') are described as 'having Contraction', on the basis that if two instances of an assumption can be used to support a conclusion, they can be 'contracted' into one, and still support it (this is a bit confusing when we get to annotating proofs with proof-terms, since Contraction is the rule that lets one variable be 'copied' as two).

Another difference between Classical and Linear Logic is that for Classical, we need to be able to discharge hypotheses that aren't there at all:

(89) $$\frac{A}{B{\rightarrow}A}\,\mathcal{I}{\rightarrow}^{1}$$

Logics where you can do this 'have Weakening' (or 'Thinning'), on the basis that a claim 'A' can be 'weakened' to the claim 'if B, then A'. To keep the symbolism and terminology confusing, the Weakening theorem ($A{\rightarrow}B{\rightarrow}A$) is called 'K'.

Weakening causes considerable philosophical disquiet, since it is involved in various intuitively suspect inferences. For example, if we know that Mary is very busy, it seems wrong or at least misleading to say 'If the Red Sox are winning, then Mary is very busy', due to the likely lack of any connection between the Sox' game and Mary's level of busyness. There has therefore been a tradition of interest in 'Relevant Logics', where Weakening is unavailable. See Restall (2006) for further discussion of discharge policies.

Disallowing Weakening and Contraction means that in a proof, each premise can and must be used once, and only once. So that, for example, if your proof that the world is doomed makes two uses of the premise 'Bush is a moron', and does not involve Ahmedinejad at all, then you can't conclude 'the world is doomed' from 'Bush is a moron' and 'Ahmedinejad is a maniac'. Rather, your premises must be 'Bush is a moron' and 'Bush is a moron' (two instances of one premise). This strikes many people as a rather wierd requirement to impose on proofs, but it can be made more natural by thinking of a very fussy person who (a) doesn't want irrelevant premises in their arguments (so that they will apply as widely as possible), and (b), is interested in *how many times* each premise is used. Counting the number of times a premise is used is equivalent to using a different copy of the premise on each occasion of use, so once you're doing that, you're really doing linear logic.

Another popular way of thinking about linear logic is in terms of 'resources', for constructing things. One might regard a super-size cheeseburger, for example, as something that converts a 200-pound American into a 202-pound American, that is, something of type **big**→**bigger**. But if we apply this to something of type **big**, then both disappear, and we're left with just something of type **bigger**. So linear logic also also referred to as (a kind of) 'resource-sensitive' logic.

Philosophy aside, we certainly get a cleaner format for tree-style deductions if we disallow both Weakening and Contraction, so that only hypotheses that appear in the tree can be discharged, and that only once. And, as a consequence, the trees become quite close in form to our glue-structures with the axiom-links contracted. One consequence of the resemblance is that the semantic reading rules we proposed for the trees apply as well to the proofs, where they are standardly called 'labels', and proofs that are 'decorated' with lambda-terms ('proof-terms') are called 'labelled deductions', and viewed as constructed by these two 'labelled deduction rules':

(90)
$$\frac{f : A{\rightarrow}B \qquad a : A}{f(a)}\,\mathcal{E}{\rightarrow} \qquad \frac{\begin{array}{c}[x : A]^i \\ \vdots \\ b : B\end{array}}{\lambda x.b : A{\rightarrow}B}\,\mathcal{I}{\rightarrow}^i$$

The connection between the rules for labelling deductions and those for labelling glue-structures shuld be intuitively evident; we will look at it a bit more closely later.

For now, we will make the point that on the proofs, they can be regarded as a convenient representation for the structure of the proof above the node that they label. But, very importantly, this was not their historical origin: the lambda-calculus format of the labels and the proof structures were historically developed indepedently of each other, and it was a major discovery, the so-called 'Curry-Howard Isomorphism' (CHI), that they are structurally the same.

3.3. Normal Forms

ND Proofs and lambda-terms are fully equivalent to each other, but they are both more permissive than glue-structures in two important ways. The easier but less important of these can be seen by considering the application of the **intersective** constructor to an adjective meaning. Consider the following instantiated versions of these (kinds of) constructors:

(91)
$$\begin{array}{rcl}\lambda P.\lambda Q.\lambda x.P(x) \wedge Q(x) & : & (h_e{\rightarrow}h_p){\rightarrow}(g_e{\rightarrow}g_p){\rightarrow}g_e{\rightarrow}g_p \\ \mathit{Crazy} & : & h_e{\rightarrow}h_p\end{array}$$

We can construct an $\mathcal{E}{\rightarrow}$ proof-step as follows:

(92)
$$\frac{\lambda P.\lambda Q.\lambda x.P(x) \wedge Q(x) : (h_e{\rightarrow}h_p){\rightarrow}(g_e{\rightarrow}g_p){\rightarrow}g_e{\rightarrow}g_p \qquad \mathit{Crazy} : h_e{\rightarrow}h_p}{\lambda Q.\lambda x.\mathit{Crazy}(x) \wedge Q(x) : (g_e{\rightarrow}g_p){\rightarrow}g_e{\rightarrow}g_p}\,\mathcal{E}{\rightarrow}$$

The formulas are longish, but what's going on is pretty simple: a complex antecedent is applied to a complex premise.

But we can't do this with glue-structures, because complex arguments have to be structured as positive implications a.k.a. λ-terms, which are represented as $\rightarrow$-introductions. If you look back at the glue- structure version of (76), you'll see that the application of **intersective** to the adjective would be represented like this as an ND proof (well, actually, you'll probably have to do some figuring to really see it; keep in mind that application nodes in the structure correspond to $\mathcal{E}\rightarrow$ steps in the proof, abstraction nodes to $\mathcal{I}\rightarrow$ steps, and axiom-links are contracted):

(93)
$$\dfrac{\lambda P.\lambda Q.\lambda x.P(x) \wedge Q(x) : (h_e \rightarrow h_p) \rightarrow (g_e \rightarrow g_p) \rightarrow g_e \rightarrow g_p \qquad \dfrac{\dfrac{Crazy : h_e \rightarrow h_p \qquad [X : h_e]^1}{Crazy(X) : h_p}}{\lambda X.Crazy(X) : h_e \rightarrow h_p}\,\mathcal{I}\rightarrow^1}{\lambda Q.\lambda x.Crazy(x) \wedge Q(x) : (g_e \rightarrow g_p) \rightarrow g_e \rightarrow g_p}$$

As a proof, the upper-right hand part of this seems kind of dumb. We've got *Crazy* of type $h_e \rightarrow h_p$, we apply it to a hypothesis of type h_e, then discharge that hypothesis to get something of type $h_e \rightarrow h_p$ again, which, furthermore, under our understanding of the labels, does exactly the same thing as the original *Crazy*. A rather pointless 'detour', it would appear.

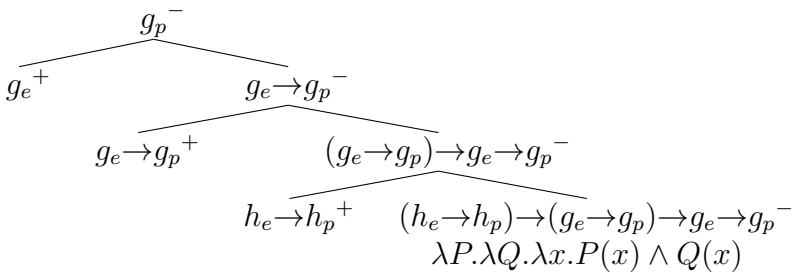
Whenever we find ourselves doing something like this, it would seem more sensible to eliminate the detour, and just use the proof-piece we had to begin with (here, $Crazy : h_e \rightarrow h_p$). This idea, formalized, becomes the 'proof normalization' rule of η-reduction, which says that a proof-piece that fits the format on the left should be simplified as indicated on the right:

(94)
$$\dfrac{\dfrac{f : A \rightarrow B \qquad [x : A]^i}{f(x) : B}\,\mathcal{E}\rightarrow}{\lambda x.f(x) : A \rightarrow B}\,\mathcal{I}\rightarrow^i \quad \Rightarrow_\eta \quad f : A \rightarrow B$$

With ND, we can use the η-reduced version of the proof, while with the glue-tree format, we can't.

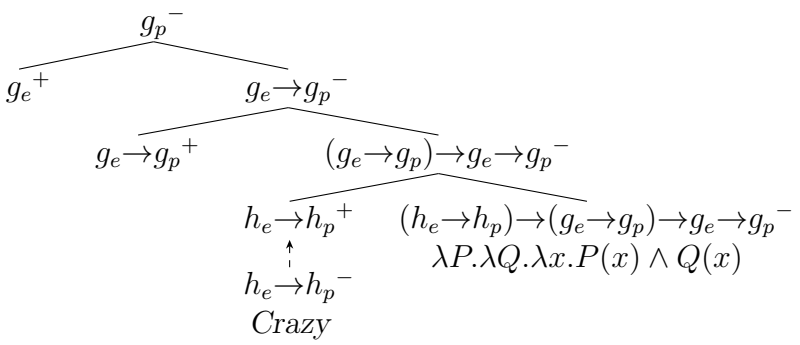
If you stare at (76) for long enough, you might be able to see that the problem is that we are only allowing ourselves to run axiom-links between atomic formulas. Suppose we abandon this restriction, and also stop insisting on expanding the meaning-constructors to glue-trees to the fullest extent possible. Then, for **intersective**, we might rest content with this partial expansion:

(95)



Now, if we could run axiom links from a negative implication to a positive one, we could effect a one-step assembly like this:

(96)



Well, why not? One part of the answer is that we'd have to do some formal work to make sure that we're not messing up the notion of the ϵ-correspondence that we're using, which is important for the linguistics. Another is that there's a really important sense in which it doesn't make any difference (so why spend a lot of effort trying to fix it).

It is one of the basic ideas of proof theory that there are many superficially different-looking proofs that really ought to be regarded as being the same, and the ones connected by η-reduction are good examples of this. Indeed, there are some contexts where the 'η-expanded' versions are regarded as being better, even though they are longer. In general, η-reduction is not seen as much of an issue, and it is a matter of convenience for the immediate purposes whether you use η-contracted forms, or 'η-expanded' ones.

But there's another thing we can do with ND proofs but not our glue-trees which is considerably more significant, which is most easily approached by looking at the proof-terms. Something that you can do with lambda-calculus is first define a function with abstraction, and then apply it to something, with application. In terms of undecorated proofs, this corresponds to the following form of structure:

(97)

$$\dfrac{\dfrac{\begin{array}{c}[A]^i\\ \vdots\\ B\end{array}}{A{\to}B}\,\mathcal{I}{\to}^i \qquad A}{B}\,\mathcal{E}{\to}$$

Here, the upper left segment is the definition, the final step, the application.

In terms of proofs alone, it looks dumb to do this, because we wind up with a proof of B from A (and maybe some additional assumptions), just as we had as represented by the A and B in the upper left, connected by a vertical string of dots. That is, we derived B from A and perhaps some other assumptions, then discharged A to get $A{\to}B$, then applied this to A again to get B again, from A and those other assumptions.

So we should have used the 'β-reduced' version of (97), which is just:

(98)
$$\begin{array}{c} A \\ \vdots \\ B \end{array}$$

But if we consider the proof-terms that would appear in the labelled version of β-reduction, it doesn't look so dumb anymore.

The labelled version of (97) will be:

(99)
$$\dfrac{\dfrac{\begin{array}{c}[x:A]^i \\ \vdots \\ f:B\end{array}}{\lambda x.f : A{\to}B}\,\mathcal{I}{\to}^i \qquad y:A}{(\lambda x.f)(y):B}\,\mathcal{E}{\to}$$

where f will in general have internal structure. So what about its β-reduced version?

Since we're discharging the assumption A labelled with x, we want x to disappear, and the 'obvious' thing to do is to redo the derivation of B from A using y instead of x, so that what we wind up with for the label of B is f, with y substituted for x. Restating the β-reduction rule with the labels in all their glory, we get:

(100)
$$\dfrac{\dfrac{\begin{array}{c}[x:A]^i \\ \vdots \\ f:B\end{array}}{\lambda x.f : A{\to}B}\,\mathcal{I}{\to}^i \qquad y:A}{(\lambda x.f)(y):B}\,\mathcal{E}{\to} \quad \Rightarrow_\beta \quad \begin{array}{c} y:A \\ \vdots \\ [y/x]f : B \end{array}$$

One important point here is that if we just grind through the steps designated by the vertical dots, using $y:A$ instead of $x:A$, $[y/x]:f$ is just what we get at the end: this fact does not depend on how we define the $[y/x]f$ notation for substitution; rather, the notation is a convenient technique for describing what we get by following the procedure.

Another is that although the non-β-reduced (left-hand) version is a dumb thing to do in isolation, it can be quite clever if we have various independent uses for the $\lambda x.f$ result. That is, if there are various different things we can apply it to. For we can 'store' the possibly rather complex construction of $\lambda x.f$, and then use a single application to apply it to various different things, getting various results by following the relatively simple rules for substitution, rather than the possibly difficult creative process of coming up with f and $\lambda x.f$ from scratch each time. This corresponds very closely to the techniques used to define functions in computer-programming languages.

Now we get to the main point of all this, which is that in ND format, we can produce non-β-reduced proofs, but in the glue-structure format, we can't. The insight into why is a bit harder than in the case of η-reduction. It is essentially that we don't have any way to produce a structure where a lambda-abstraction is made to apply to something. Lambda-abstractions are represented by positive implications, which appear only in 'argument position', as left-daughters to negative implications. There, if an implication appears as right-daughter of a negative implication, it is negative itself, and so is either the result of an application, or has a semantic label.

This deficiency, if it be reckoned a deficiency, can be fixed by adding another kind of link, called 'cut link', that runs from a positive implication to a negative one. But for our purposes, it's not a deficiency at all, but a virtue, because it means that the glue-structure system can only represent proofs that are in β-normal form. To see why this is good, consider how much easier elementary school arithmetic would have been if there were some convenient way to write down fractions that could only be used to produce fractions that were already in lowest terms, the way the math teacher wanted them.

So, in spite of coming much later in the average student's education, β-reduction of proofs and lambda-terms is actually a lot easier than reducing fractions, and we even have a format where non-reduced ones can't be written down at all!

The conceptual background to this is that Proof Theorists want to tame the bewildering variety of different ways in which something might be proved by grouping 'essentially equivalent' proofs into classes, and regarding these classes as being the real proofs. We would then like to pick representatives from these classes, so that, for example, that if there are three classes of 'essentially equivalent' proofs of some proposition, we can represent the resulting three 'essentially different' proofs by writing down representatives, ideally chosen in some non-arbitrary fashion.

β-reduction produces a very useful classification of proofs, whereby any proofs that can be interconverted by a chain of β-reductions and reversed β-reductions are held to belong to the same class. And the β-reduced proofs are the obvious representatives (just like the lowest term form of a fraction is the obviously best representative of the rational number designated by the fraction). Now for our glue-structure scheme we also have η-reduction to think about, but here we're not using the η-reduced forms. But this is harmless, due to the following fact. If we start with a basic predicate, say

Crazy, and η-expand once, we get $\lambda x.Crazy(x)$. β-reduction can do nothing with this. But if we η-expand again, we get $\lambda y.(\lambda x.Crazy(x))(y)$, which is in fact subject to β-reduction, which will knock it back to $\lambda y.Crazy(y)$. But that's just what we got after a single η-expansion (we ignore different choices for bound variables; this is technically called 'α-equivalence'). This means that there is a coherent notion of 'β-reduced, but η-expanded as much as possible', which is sometimes designated as '$\beta\eta^{-}$-reduced', on the basis that something is η^{-}-reduced if it is η-expanded as much as possible without thereby becoming subject to β-reduction (sort of like claiming as many tax deductions as possible without incurring a high probability of getting audited). This is often called η-long normal form.

So, the final conclusion is that the glue-tree format gives us a single representative, in $\beta\eta^{-}$ normal form, of each possible ND proof. And these proofs, however represented, provide instructions on how to compose the meanings of the words in the utterance, to the extent that such composition can be understood as the application of functions to arguments. Which is what Montague Grammar was clearly the first kind of system to achieve for reasonable fragments of English and comparable languages, a result which glue semantics therefore has let us transfer to languages that can be given a syntactic analysis with LFG.

We have also managed to work through the most fundamental part of the famous 'Curry Howard Isomorphism', which is the discovery that the system of implicational proofs, constructed with $\mathcal{E}\rightarrow$ and $\mathcal{I}\rightarrow$, has exactly the same mathematical structure as the independently invented system of typed lambda-calculus, with application and abstraction constructions, with the β and η reductions corresponding to the proof-simplification schemes which are given the same name, thanks to this discovery. These things were invented independently, in the first part of the 20th century, so that it was a real discovery that they were basically the same thing. And the CHI holds up under various kinds of further development. For example 'discharge policy' for proofs is the same thing as 'binding policy' for lambdas: a 'linear lambda' can and must bind one and only one instance of its variable, while a 'relevant lambda' must bind at least one, but can bind as many as desired.

We haven't quite filled in all the details of the relationships between implicational proofs and implicational proof-nets, but the fact that they both produce proof-terms that encode the structure of the proof or proof-net that produced them indicates that they are essentially the same thing. This is discussed more carefully in Perrier (1999).

4. Anaphora: A Deeper Dive

4.1. Tensors and Anaphora

We've now dealt with the 'implicational' part of glue, but there is a further development of what is called 'tensors', used in many analyses of anaphora (and with various other potential applications discussed in Asudeh (2004)). Tensors, in the context of glue,

are essentially a way of packaging two resources as one, and we'll begin by considering why anaphora makes them potentially useful. Consider the interpretation of a sentence such as:

(101) Every man$_i$ loves his$_{i,j}$ mother

This is generally taken to be ambiguous, having one reading where the subject and possessor of the object are taken to be 'coreferential' (scare quotes because neither is in fact referential at all), awkwardly paraphrasable as 'For every man x, x loves x's mother', and another reading where there is some male person previously introduced in the conversation, such that every man loves that person's mother.

The former reading is an instance of 'bound anaphora', and it is generally accepted that to account for it, we need some way of in effect getting the same variable into the subject and possessor-of-object positions, so that *every man* applies to the property '$\lambda x.x$ loves x's mother'. Tensors are not the only way to accomplish this; a purely implicational alternative is discussed in Lev (2007), and there are also proposals to use glue in a version of Discourse Representation Theory (Lev 2007, Kokkonidis 2005). But the tensor-based analysis is used in quite lot of important work by Asudeh, and so is worth understanding in spite of the existence of alternatives. It is also the only treatment of bound anaphora that (a) avoids a problem of spurious ambiguity discussed by Lev that arises with the tensor-free alternative (b) uses pure glue+lambda-calculus mechanisms to account for bound anaphora.

We begin our consideration of (101) with the meaning-constructors that we already understand:

(102)
$$\begin{aligned} \textit{Every} &: (g_e \rightarrow g_p) \rightarrow (g_e \rightarrow f_p) \rightarrow f_p \\ \textit{Man} &: g_e \rightarrow g_p \\ \textit{Love} &: h_e \rightarrow g_e \rightarrow f_p \\ \textit{Mother} &: j_e \rightarrow h_e \\ \textit{His} &: ? \end{aligned}$$

assuming this f-structure, where the ANT(ECEDENT) GF connects the pronoun to its antecedent:

(103)
$$f{:}\begin{bmatrix} \text{SUBJ} \;\; g{:}\begin{bmatrix} \text{SPEC} & \text{'Every'} \\ \text{PRED} & \text{'Man'} \end{bmatrix} \\ \text{PRED} \quad \text{'Love'} \\ \text{OBJ} \quad h{:}\begin{bmatrix} \text{PRED} & \text{'Mother'} \\ \text{POSS} \;\; j{:}\begin{bmatrix} \text{PRED} & \text{'Pro'} \\ \text{ANT} \;\; g{:}[\;\;] \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

A way to get the object's possessor j to have access to the variable assigned to the subject g would be to give the subject a constructor such as:

(104) $\lambda x.x \ : \ g_e \rightarrow j_e$

But this would leave no way for the subject argument of the verb to be satisfied. Tensors solve this problem by letting us take the resource from subject position, and put one resultant in the object possessor's position, and another back in the subject position, so that both of the relevant arguments get satisfied.

The form of the (instantiated) constructor is:

(105) $\lambda x.[x, x] \ : \ g_e \rightarrow g_e \otimes j_e$

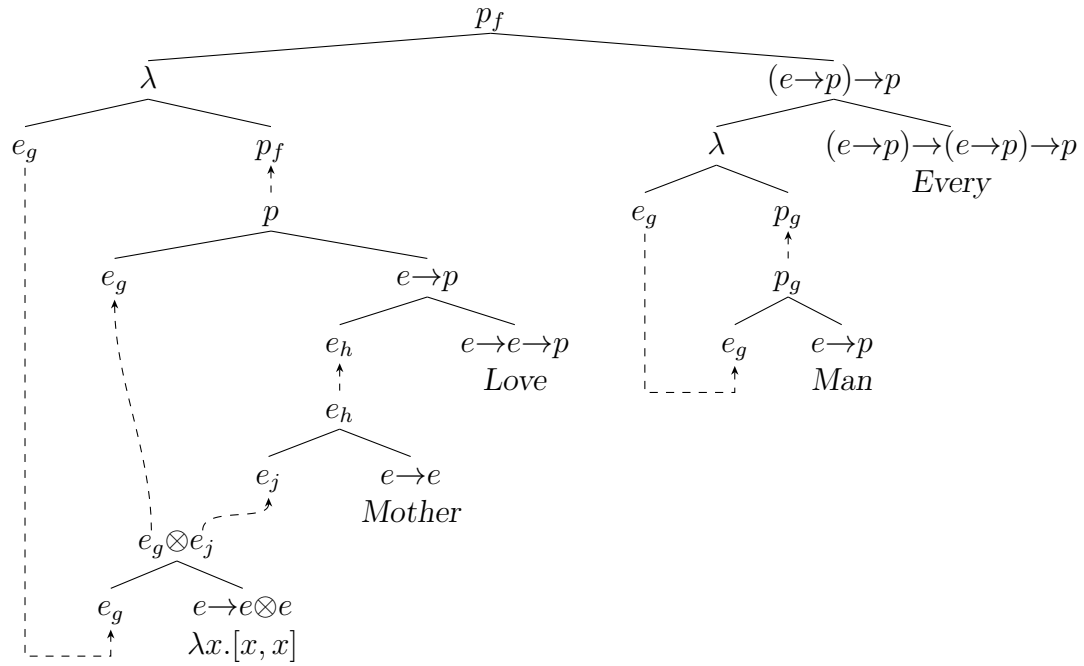
$\otimes$ is the somewhat intimidating symbol used for the tensor[26] and what it means is that by consuming the resource at g, we get a pair of resources, one at g, the other at j. The meaning-side says that what you get by consuming the resource at g is a pair of resources, each being a copy of the original (this concept of copy is not part of glue itself, but of the semantics to which glue is providing the input). One copy then goes to each location.

The 'resource' interpretation of linear logic is quite useful for tensors. On this interpretation, tensoring two resources amounts to simply combining them into a single resource that includes both. So if we have **big** (the 200-pound American) and **big**$\rightarrow$**bigger** (the 2-pound cheeseburger), both of these items together can be described as **big**$\otimes$(**big**$\rightarrow$**bigger**). In 'Relevant Logic', the tensoring operation is called 'fusion' and is symbolized as '$*$'; this choice of symbol, used in Troelstra (1992), would perhaps have made for a more beginner-friendly presentation of glue. The essential characteristic of tensors/fusion as opposed to standard conjunction is that once you've got one of these combined resources, you have to use both components (in Relevant Logic, you can copy, but not delete; in Linear, you can do neither).

In terms of our 'tree'-notation (scare quotes because once we do this, they aren't in general trees anymore), we interpret the tensor as something that produces two outputs, each of which can be sent to a different location (but not completely independently, since there are some constraints that have to be satisfied). Here is an example:

[26] There is actually a connection to the linear algebra tensor product concept, although it is somewhat abstract, and not at all required to understand glue tensors. See Baez and Stay (2008) for some general discussion.

(106)

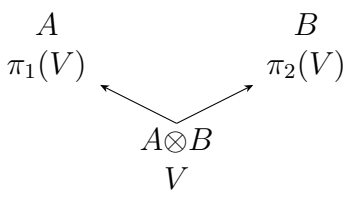


This diagram shows how the inputs and outputs are hooked up, but how do we get the semantic readings?

4.2. Proof Terms for Tensors

This can be done with (a) a propagation rule (b) a β-reduction. The propation rule needs to assign values to the two components of a tensor, on the basis of the value of the tensor. To illustrate this, it is useful to display the tensor a bit more explicitly, with the two components at the ends of upward pointing branches. Below, the values are written underneath, so that if the value of the whole tensor is V, then the left branch is $\pi_1(V)$, the right $\pi_2(V)$:

(107)



So if the value of the tensor in (106) is $[x, x]$, then the values of the two components are $\pi_1([x, x])$ and $\pi_2([x, x])$. So to get the effect of copying that we want, we need these β-reductions:

(108) a. $\pi_1([U, V]) \Rightarrow_\beta U$

b. $\pi_2([U, V]) \Rightarrow_\beta V$

You can now work out that, given this propagation scheme, the semantic value of the top node of (106) will now be $\mathit{Every}(\mathit{Man})(\lambda x.\mathit{Love}(\mathit{Mother}(x))(x))$ (after β- and η-reductions).

This gives a bit of an idea of what they do, but to use them legitimately, some constraints must be obeyed. One can see the need for this by noting that sentence (109) below ambiguously allows wide or narrow scope for the quantifier (the latter most plausible in a highly dynastic pseudo-democracy):

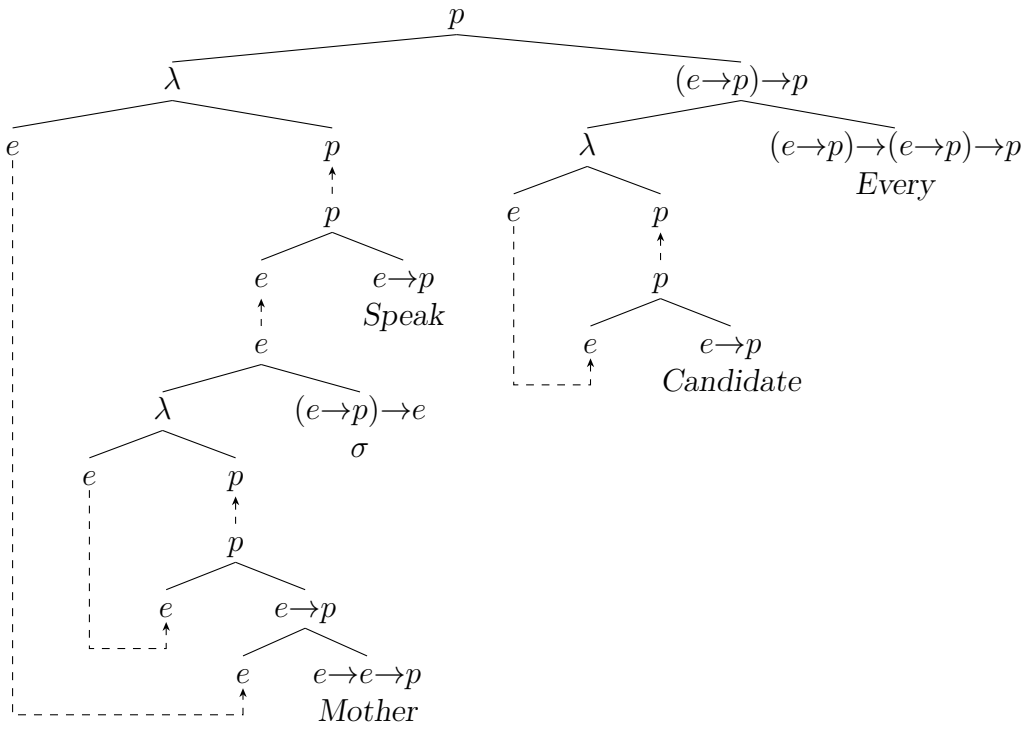
(109) The mother of every candidate spoke.

But if we include a bound pronoun in a non-subject argument, only a wide-scope reading for the quantifier becomes possible:

(110) The mother of every candidate$_i$ says that he$_i$ is patriotic

To represent the ambiguity of (109), we need a representation of the definite article, for which we will use the symbol σ of type $(e{\to}p){\to}e$, interpreted as converting a property to the unique individual that satisfies that property if there is one, and returning undefined/error if there isn't.[27] Now the wide and narrow scope readings of (109) can be represented by (a) and (b) respectively, with tense ignored, and a more conventional logical notation appearing underneath each structure:

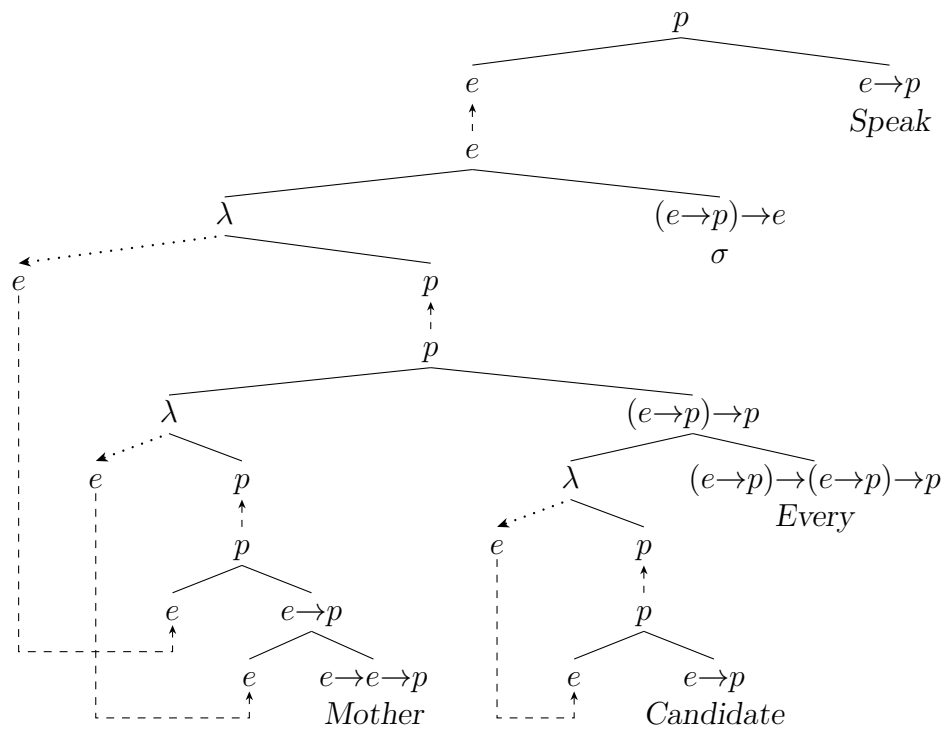
(111) a.



[27]This is the definite description operator conventionally symbolized as ι, but we use σ to cover the extension of this to deal with plurals in the algebraic approach to plurality developed in Link (1998).

$Every(x, Candidate(x), Speak(\sigma(y, Mother(y, x))))$

b.



$Speak(\sigma(y, Every(x, Candidate(x), Mother(y, x))))$

Now consider a variant of (b) where the main verb is a two-place predicate, containing a pronoun bound by the quantifier internal to the subject.

The pronoun's constructor would have to take content from the quantifier, and return one copy to a position within the scope of the quantifier and another to a position not within the scope of the quantifier, a situation we clearly want to rule out. But to understand the principles needed to exclude this, it is helpful to shift to a deductive view of tensors.

First, there will be a rule to introduce them. Given their interpretation as a composite of two resources, the following rule to introduce them is not too surprising:

(112) $$\frac{A \qquad B}{A \otimes B}\,\mathcal{I}\otimes$$

However, somewhat annoyingly, this very easy rule isn't used in current glue, because there aren't any current analyses in which tensors are used as arguments (certain phenomena, such as coordinate structures, provide plausible applications, but analysis so far, such as Asudeh and Crouch (2002a), haven't used them). We will however use it in some theoretical discussions below.

What is used instead is the elimination rule, which is discouragingly complex on first encounter. To understand it, return to the idea of the tensor $A \otimes B$ is something that provides both A and B (together) as resources. This can be taken as meaning that if there is some C that can be produced given that we have an A and a B, then we can also produce C given an $A \otimes B$. This suggests the following rather complex rule, in which two premises get discharged at once:

(113)
$$\dfrac{A \otimes B \qquad \begin{array}{c}[A]^i [B]^j \\ \vdots \\ C\end{array}}{C}\, \mathcal{E}\otimes^{i,j}$$

The right-hand branch represents the possibility of deriving C from A and B as assumptions, while the whole thing says that if we really have $A \otimes B$, we can then use this to derive C without having to assume A and B.

What proof-terms will we use for this rule? There are actually two choices. A minority choice, found for example in some of the type-logical grammar literature such as Moot (2002) and Jäger (2005), is essentially the same as we used for semantic labelling of our glue-structures:

(114)
$$\dfrac{w : A \otimes B \qquad \begin{array}{c}[x : A]^i [y : B]^j \\ \vdots \\ z : C\end{array}}{[\pi_1(w)/x, \pi_2(w)/y]z : C}\, \mathcal{E}\otimes^{i,j}$$

This proof-term for the final conclusion says: replace x with the left projection of the tensor's proof term, y with the right projection of the tensor's proof term, in the term we z that we first get for C.

For a simple example, consider the proof for *John likes himself*:

(115)
$$\dfrac{\dfrac{\lambda x.[x,x] : e_g \rightarrow e_g \otimes e_h \quad \mathit{John} : e_g}{(\lambda x.[x,x])(\mathit{John}) : e_g \otimes e_h} \qquad \dfrac{\dfrac{\mathit{Like} : h_e \rightarrow g_e \rightarrow f_p \quad [y : h_e]^1}{\mathit{Like}(y) : g_e \rightarrow f_p} \qquad [x : g_e]^2}{\mathit{Like}(y)(x) : f_p}}{\mathit{Like}(\pi^2(\lambda x.[x,x])(\mathit{John}))(\pi^1(\lambda x.[x,x])(\mathit{John})) : f_p}$$

In this deduction, we have only taken steps that are doable in the linear logic portion of glue (meaning-assembly); when we take the innards of the meaning-constructors into account and move to full lambda-calculus, then both of the projection terms reduce to *John* (using the β-reductions of (108)), yielding the appropriate final result.

In the glue (and almost all of the linear logic) literature,[28] on the other hand, the perhaps more initially confusing (116) is found:

(116)
$$\frac{w : A \otimes B \qquad \begin{array}{c}[x : A]^i [y : B]^j \\ \vdots \\ z : C\end{array}}{\textbf{let } w \textbf{ be } x \times y \textbf{ in } z : C}\, \mathcal{E}\otimes^{i,j}$$

Using this term-assignment, the proof-term for (115) becomes:

(117) $\textbf{let } (\lambda x.[x, x])(\mathit{John}) \textbf{ be } x \times y \textbf{ in } \mathit{Like}(y)(x) : f_p$

To get the desired result we need the following β-reduction:

(118) $\textbf{let } [u, v] \textbf{ be } x \times y \textbf{ in } z \Rightarrow_\beta [u/x, v/y]z$

These terms produce the same β-reduced results in the end, via different-looking intermediate stages.

There is unfortunately no discussion that I have been able to find in the literature of the issues behind the choice between these two forms of proof-term, but there are some observations to be made. One is that the latter form preserves the straightforward notion of 'linear' lambda-term, whereby a lambda-term is linear if each lambda-bound variable appears once and only once in the body of its lambda-term. So for example if we discharge the assumption $w : A \times B$ after deriving C in (116), the resulting labelled conclusion will be:

(119) $\lambda w.\textbf{let } w \textbf{ be } x \times y \textbf{ in } z : C$

where w appears only once in the body of the abstraction.

On the other hand, using (114), we get:

(120) $\lambda w.[\pi_1(w)/x, \pi_2(w)/y]z$

where w literally appears twice, once in each projection-term. It is clearly more work to keep track of whether a proof-term in this style is actually linearly well-formed than is the case with the other one (especially if you've taken projections of things that contain projections ...).

This reflects that fact that the proof-terms of (116) accurately preserve the form of the actual natural deduction, while those of (114) don't, but rather give the same

[28] I think I once saw projection terms used in an early paper by Philip Wadler, but have not managed to relocate it.

representation to certain differerent- looking deductions (examples below—you should try to devise some now). This might seem to be a fatal flaw, but actually it isn't, for the reason that proof-theorists want to treat these so-identified deductions as actually being the same, by means of 'commuting conversions' that declare various deductions to be equivalent. To understand the reason for this, we need to consider the notion of 'normal proof' for ND deductions involving tensors.

The following two kinds of pointless detours, with their reductions, can be identified:[29]

(121)

a.
$$\dfrac{\dfrac{\overset{\vdots}{\alpha}}{u:A} \quad \dfrac{\overset{\vdots}{\beta}}{v:B \;\; v:B}}{\dfrac{[u,v]:A{\otimes}B \qquad \begin{matrix}[x:A]^i[y:B]^j\\ \vdots\\ \gamma\\ \vdots\\ z:C\end{matrix}}{\textbf{let } [u,v] \textbf{ be } x \times y \textbf{ in } z}\,\mathcal{E}{\otimes}^{i,j}} \quad \Rightarrow_\beta \quad \dfrac{\begin{matrix}\overset{\vdots}{\alpha}\\ u:A\end{matrix} \quad \begin{matrix}\overset{\vdots}{\beta}\\ v:B\end{matrix}}{\begin{matrix}\vdots\\ [u/x,v/u]\gamma\\ \vdots\\ [u/x,v/y]z:C\end{matrix}}$$

b.
$$\dfrac{\begin{matrix}\overset{\vdots}{\alpha}\\ z:A{\otimes}B\end{matrix} \qquad \dfrac{[x:A]^i \qquad [y:B]^j}{[x,y]:A{\otimes}B}\,\mathcal{I}{\otimes}}{\textbf{let } z \textbf{ be } x \times y \textbf{ in } [x,y]:A{\otimes}B}\,\mathcal{E}{\otimes}^{i,j} \quad \Rightarrow_\eta \quad \begin{matrix}\overset{\vdots}{\alpha}\\ z:A{\otimes}B\end{matrix}$$

(a) in particular may seem dauntingly complex, but note that it isn't an extraneous rule that has been added to the system, but rather an exhibition of a relationship that already exists within it: if we have a proof of the form specified on the left, we also have an intuitively simpler one that does the same job on the right. But, interpreted as rules applying from left to right, these reductions share the property of the implicational β and η reductions of being terminating and confluent, so they produce decent normal forms for proofs. The η-rule furthermore has the property that if you apply it in the opposite direction twice (as an 'expansion', the second application applying to the $z : A{\otimes}B$ term in the result of the first), you produce an opportunity to apply the β-rule, so that there is a notion of η-long normal form, just as with the implicational expansions, which turns out to be closely related to the structure of proof-nets.

So far, so good. But the annoying fact is that there are proofs that look like they ought to be reducible, but aren't according to the system so far, because 'something is in the way'. For example (proof terms omitted; supplying them would be a good exercise):

(122)
$$\dfrac{A{\otimes}B \qquad \dfrac{A{\otimes}B{\rightarrow}C \qquad \dfrac{[A]^1 \qquad [B]^2}{A{\otimes}B}}{C}}{C}\,\mathcal{E}{\otimes}^{1,2}$$

[29] The β-rule can be found in Benton et al. (1992) and Mackie et al. (1993), the η-rule in the latter.

This doesn't reduce 'as is', but will if we can 'permute' the $\mathcal{E}\otimes$ step upward around the function-application, so that it applies within the minor premise of the latter:

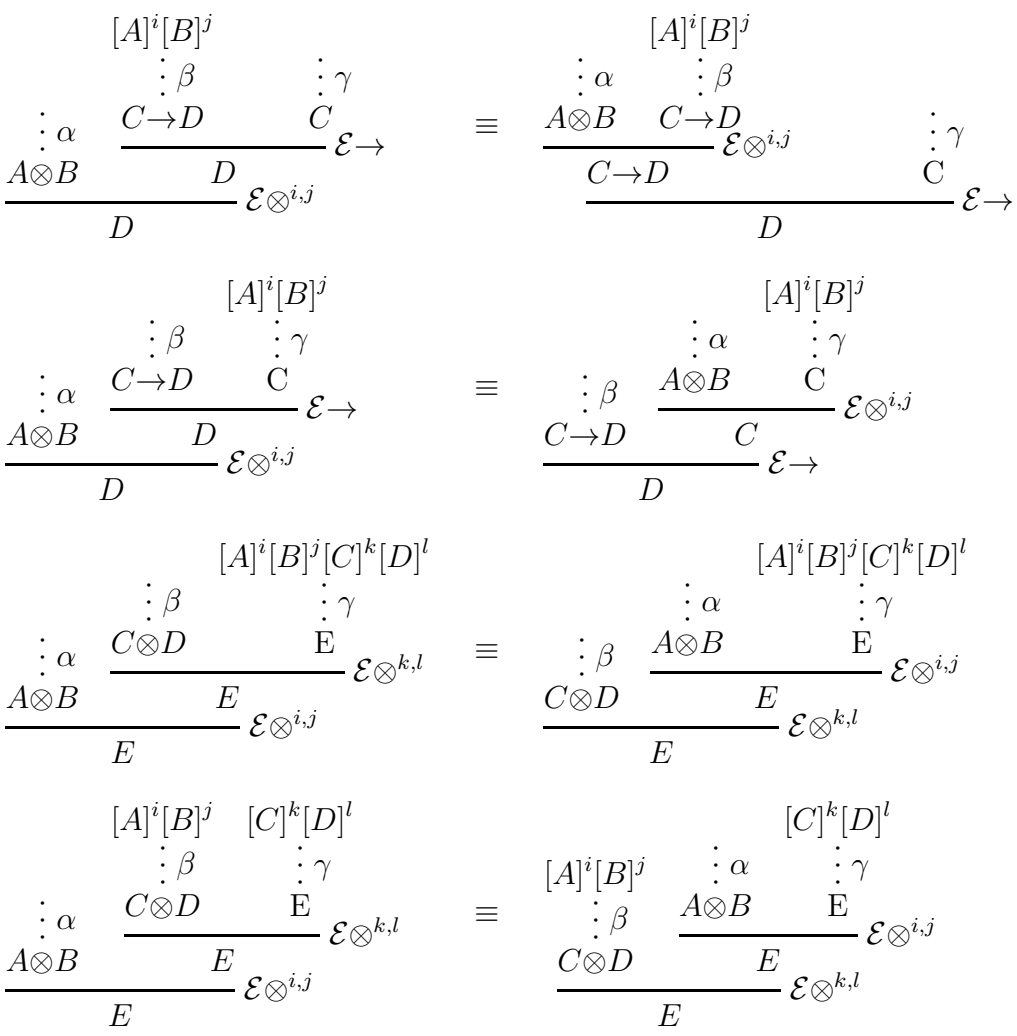
(123)
$$\dfrac{A\otimes B\to C \qquad \dfrac{A\otimes B \qquad \overset{[A]^1 \quad [B]^2}{\overline{A\otimes B}}\,\mathcal{E}\otimes^{1,2}}{C}}{C}$$

There's certainly nothing terrible about doing this, since (123) is a valid proof, and now we can apply η-reduction to get:

(124)
$$\dfrac{A\otimes B\to C \quad A\otimes B}{C}$$

Unlike the reductions, the commuting conversions don't have any inherent directionality, and they do not simplify, but merely set things up for simplifications. We'll formulate them as rules that move a $\mathcal{E}\otimes$ step upward, into one or the other premise of other rules (but if one of the terms being discharged appears in each premise of a rule, we obviously can't commute. Some of the conversions can be found in Benton et al. (1992), all of them in Mackie et al. (1993), formulated as operations on proof-terms:

(125)



We don't give a rule for $\mathcal{I}\otimes$, since we're not using it in glue (it should be straightforward to produce, on the model of the others).

Note that in each case, from the existence of a proof of the form on the left, it follows that there exists the one on the right, deriving the same conclusion from the same premises. What might be regarded as 'artificial' is not the existence of the relationship, but the decision to regard proofs so-related as being the same. Exactly how far one should go in identifying proofs is an aspect of Proof Theory that people seem on the whole to be a bit unsure about (as long as all proofs of a given conclusion from the same assumptions aren't identified, which would collapse the 'identity of proof' relation into the entailment relation), but enough identification to support normalizations (or cut-elimination, in the case of sequent proofs) seems to be universally supported.

And, the real point of all this is that the apparent discrepancy between the structure of the proofs and that of the proof-nets is resolved by the conversions. For it is easy to work out that if we use projection-based terms, the proofs equated by the conversions of (125) will get the same proof-terms. It is furthermore the case, although not quite

so straightforward to show (see Appendix B for the proof), that two proofs that get the same projection-based proof-term will be equivalent according to the conversions of (125). So the conversions resolve the apparent discrepancy between the ND proofs on the one hand, and the structure of the proof-nets and the projection-based proof terms on the other.

There is a somewhat minor linguistic point that might be made here as well. If one works considers and example such as:

(126) John probably likes himself

it will be apparent that there are two ND proofs, depending on whether the anaphora is managed before or after the interpretation of the adverb. But this structural ambiguity does not correspond to any real one. A similar point about a non-tensor-based treatment of anaphora is made by Lev (2007:360), but for the tensor-based analysis, the commuting conversions/projection-terms resolve the problem, since the putatively different proofs are actually deemed to be the same.

The final observation I'll make about this is that if we do include $\mathcal{I}\otimes$ in glue (which amounts to allowing tensors to serve as arguments, instead of Currying everything), then the proof-nets are η-long with respect to the tensor-reduction rules, basically for the reason that we can only connect literals with axiom-links, not complex formulas. η-longness is sometimes characterized by linguistics referees as a problem with the proof-net notation, and perhaps they are right, but from a mathematical point of view there's no issue here.

4.3. Yet Another Idea

There is an alternative path to these terms which also might be seen as interesting, due to its dependence on a rather non-trivial theorem, and its relationship to the intuitionistic deductions that are equivalent to the not necessarily linear typed lambda-calculus proof terms that we want in the end for the semantics.

This starts with the logical notion of an 'intuitionistic sequent', which is simply a pair consisting of a sequence of formulas on the left, and a single formula on the right, which is 'valid' if there is a proof of the formula on the right from those on the left. The symbol '$\vdash$' is normally used to separate the two members of a sequent, so it follows from the CHI that a collection of words of semantic types $t_1, \ldots, t_n$ can assemble to produce a result of type p iff and only if $t_1, \ldots, t_n \vdash p$ is a valid sequent of linear $\rightarrow, \otimes$-logic.

The next step on the path is to notice that a proof-net is essentially the same thing as a valid sequent where each literal that appears at all appears only twice (validity will then require that one appearance be in a positive, the other in a negative, polarity position). This is because all we do when constructing the net is to connect pairs of type-identical literals (of opposite polarity); any technique for specifying a pairing will determine the

net. A sequent is said to be 'balanced', or to have the 'two-property', when every literal appears at most twice. So a valid sequent with the two-property represents a proof-net, and proves every sequent that can be derived from it by substitution of literals for literals (its 'literal substitution instances').

And so finally, we get to use the following theorem due to Babaev and Solov'ev (1982) (also proved in Troelstra and Schwichtenberg (2000)):

(127) A balanced intuitionistic $\rightarrow, \wedge$-sequent has at most one normal proof.

This theorem applies to full intuitionistic logic where we have weaking and contraction, which is equivalent to standard typed lambda-calculus with pairing. In such logics, the tensor connective $\otimes$ behaves equivalently to $\wedge$, which has the same introduction rule as $\otimes$, but the following elimination rules, standardly formulated with projections:

(128) $$\frac{z : A \wedge B}{\pi^1(z) : A}\,\mathcal{E}\wedge \qquad \frac{z : A \wedge B}{\pi^2(z) : B}\,\mathcal{E}\wedge$$

These are just the rules that we used to assign values to the components of a negative tensor.

The result is that a balanced $\rightarrow\otimes$ sequent of linear logic also has a unique normal intuitionistic $\rightarrow\wedge$ (nonlinear) proof, and it is quite straightforward to show that Perrier's scheme for assigning semantic values to a proof net, extended by the tensor rules, produces that proof. So the projection-based terms represent not only natural equivalence classes of ND proofs, but also the structure of the nonlinear intuitionistic counterpart of the linear proof. The one discrepancy between the structure of the nets and of the intuitionistic proofs is that the proofs contain multiple copies of the derivations feeding into $\otimes/\wedge$-elimination, while the nets have only one. There is a formalism called 'deduction graphs' (Loeb 2007) which looks like it might address this issue, but I haven't looked into it.

4.4. Further Prospects for Anaphora

It seems quite likely that glue semantics can implement versions of Jäger's (2005) 'static semantics' Type-Logical Grammar analyses for a variety of phenomena such as Verb Phrase Ellipsis, Antecedent-Contained Deletion, and Paycheck Sentences, by rendering a Jäeger type such as $A \mid B$ (something that requires an antecedent of type B to produce something of type A) as $B{\rightarrow}B{\otimes}A$, where Jäger's 'limited contraction' cashes out as the requirement that the meaning-side of these constructors be of the form $\lambda x.[x, F(x)]$, F being some function (usually, identity, spelled out by the constructor). For example, we can get the 'strict' and 'sloppy' readings for (129) without postulating ambiguity in the first clause:[30]

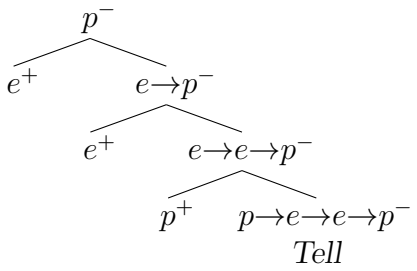
[30]I'm indebted to Ash Asudeh for pointing out to me that this can indeed be one; note that the glue processing of the second clause requires quite a bit of reworking of the glue analysis of the first, but

(129) John$_i$ loves his$_i$ mother, and so does Bill

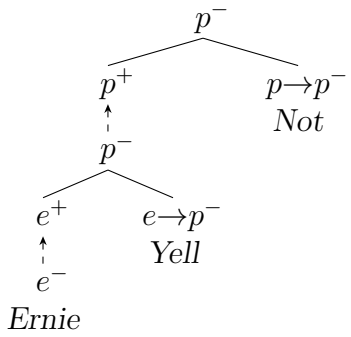
But it is much less clear what to do when we consider the standard evidence for 'dynamic' semantics, such a donkey and intersential anaphora; the major proposals to date seem to be Crouch and van Genabith (1999), Dalrymple (2001), Kokkonidis (2005) and Lev (2007). The main problem, it seems to me, is that static semantic interpretation can be construed quite cleanly as a category-theoretic functor from the glue-proofs into a 'real semantics' (construed as a Cartesian Closed Category), while it is quite unclear to me at least what the general nature of the various dynamic proposals are, so that the relationships between the notations they use and the concepts they are supposed to express is not very well understood.

Appendix A: Structures for Exercises

(130)



(131)



(132) a. (20) fails to satisfy (23) because the node with meaning-label *Not* is a Principle Input, but there is no dynamic path from it to the the Final Output.

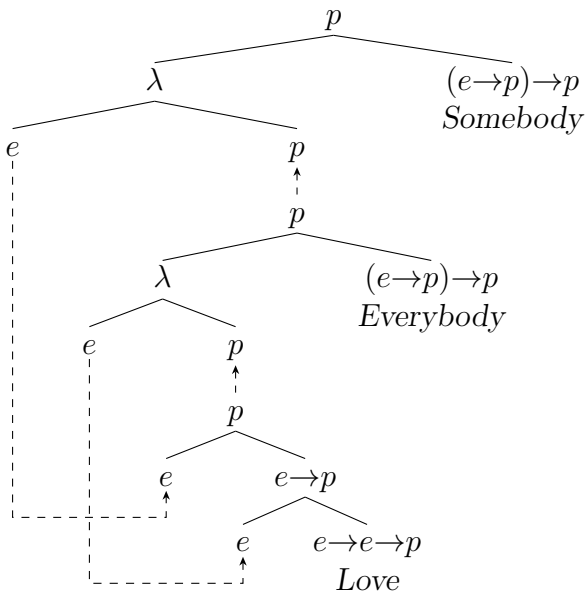
b. (21) satisifes (23), because the Principle Inputs are the nodes with the meaning-labels *Bert*, *Likes* and *Ernie*, and, from each of these, the dynamic graph provides a path to the Final Output.

(133) *Not*(*Yell*(*Bert*)).

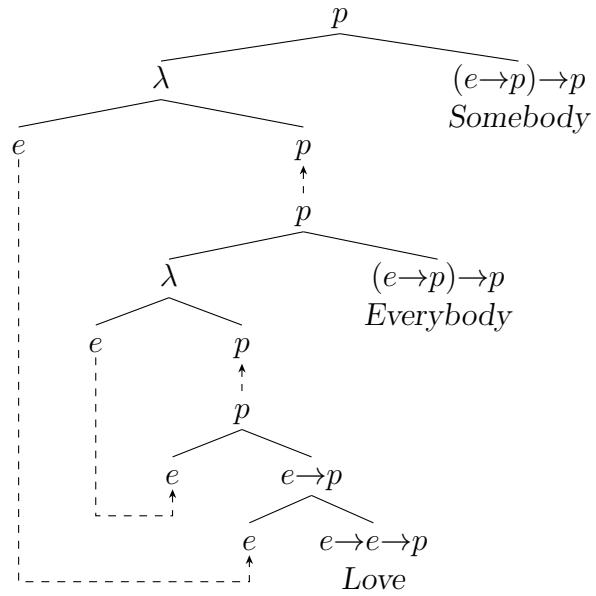
---

the meaning-term associated with the first clause remains the same, after the β-reductions associated with the tensors.

(134) a.



b.



(135) $(A{\to}B{\to}C){\to}(B{\to}A{\to}C)$

Appendix B: Term Equivalence Proof

(136) Theorem: Two ND proofs with the same projection-based proof-term are equivalent modulo the commuting conversions, and (reversible) substitutions of eliminated variables.

Proof: each $\otimes$-elimination step introduces a pair of 'projection-term-halfs', which can be uniquely co-indexed with the step that introduces them. E.g.

(137)
$$\frac{\begin{array}{cc} & [x:A]^i[y:B]^j \\ & \vdots \\ w:A\otimes B & z:C \end{array}}{[\pi_1(w)^k/x, \pi_2(w)^k/y]:C}\, k, \mathcal{E}\otimes^{i,j}$$

We'll call the two halfs a 'projection-term', so that projection-terms are discontinuous parts of the entire proof-term.

We procede by induction on the number of $\otimes$-elimination steps, equivalently, projection-terms. If there are none, the result is immediate.

Otherwise, there is a projection-term that doesn't contain any others. Then the $\otimes$-elimination step corresponding to this projection term can be migrated by the conversions to the end of each proof (without changing the proof-term, call this 'step 1'). So now, each of the two proofs is equivalent to one of the form (137), but with possibly different derivations for $w : A\otimes B$ and $z : C$. But these proofs contain fewer $\otimes$-eliminations, so are connected by chains of conversions. But these conversions are equally valid when viewed as applied internally to the two proofs we derived in step 1, so the original proofs are connected by a chain of commuting conversions (which are reversible). QED.

References

Alsina, A. 1996. *The Role of Argument Structure in Grammar.* Stanford, CA: CSLI Publications.

Alsina, A. 1997. A theory of complex predicates: Evidence from causatives in Bantu and Romance. In A. Alsina, J. Bresnan, and P. Sells (Eds.), *Complex Predicates*, 203–246. Stanford, CA: CSLI Publications.

Andrews, A. D. 2006. Semantic composition for NSM, using LFG + Glue. In K. Allan (Ed.), *Selected Papers from the 2005 Conference of the Australian Linguistic Society.* `http://www.als.asn.au/proceedings/als2005.html/` (accessed 19 Feb 2010).

Andrews, A. D. 2007a. 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. `http://arts.anu.edu.au/linguistics/People/AveryAndrews/Papers` (accessed Feb. 19 2010).

Andrews, A. D. 2007b. Glue semantics for clause-union complex predicates. In M. Butt and T. H. King (Eds.), *The Proceedings of the LFG '07 Conference*, 44–65. Stanford CA: CSLI Publications. `http://cslipublications.stanford.edu/LFG/12/lfg07.html` (accessed Feb 19, 2010).

Andrews, A. D. 2008a. Propositional glue and the correspondence architecture of LFG. to appear in Linguistics and Philosophy. Draft uploaded to semantics archive; also http://arts.anu.edu.au/linguistics/People/AveryAndrews/Papers.

Andrews, A. D. 2008b. The role of PRED in LFG+glue. In M. Butt and T. H. King (Eds.), *The Proceedings of the LFG '08 Conference*, 46–76. Stanford CA: CSLI Publications. http://cslipublications.stanford.edu/LFG/13/lfg08.html (accessed 19 Feb 2010).

Andrews, A. D. 2009. 'grammatical' vs. 'lexical' meaning constructors for glue semantics. to appear in the Proceedings of the 2009 ALS conference. http://arts.anu.edu.au/linguistics/People/AveryAndrews/Papers.

Andrews, A. D., and C. D. Manning. 1993. Information-spreading and levels of representation in LFG. Technical Report CSLI-93-176, Stanford University, Stanford CA. http://nlp.stanford.edu/~manning/papers/proj.ps (accessed 19 Feb 2010).

Asudeh, A. 2004. *Resumption as Resource Management*. PhD thesis, Stanford University, Stanford CA. http://http-server.carleton.ca/~asudeh/ (accessed 19 Feb 2010).

Asudeh, A. 2005a. Control and resource sensitivity. *Journal of Linguistics* 41:465–511.

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

Asudeh, A. 2008. Towards a unified theory of resumption. URL: http://http-server.carleton.ca/, http://semanticsarchive.net/Archive/jNjZWRkM/asudeh08-rp.pdf.

Asudeh, A., and R. Crouch. 2002a. 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.

Asudeh, A., and R. Crouch. 2002b. Derivational parallelism and ellipsis parallelism. In L. Mikkelsen and C. Potts (Eds.), *WCCFL 21 Proceedings*, 1–14, Somerville MA. Cascadilla Press. URL: http://http-server.carleton.ca/~asudeh/research/index.html.

Babaev, A., and S. Solov'ev. 1982. A coherence theorem for canonical morphisms in cartesian closed categories. *Zap. Nauchn. Sem* LOM1 88:3–39, 236. also J. Soviet Math. 20:2263-2279.

Baez, J., and M. Stay. 2008. Physics, topology, logic and computation: A rosetta stone. URL: http://math.ucr.edu/home/baez/rosetta.pdf.

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: `http://citeseer.ist.psu.edu/article/benton92term.html`.

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.

Copestake, A., D. Flickinger, C. Pollard, and I. A. Sag. 2005. Minimal recursion semantics: an introduction. *Research on Language and Computation* 3:281–332.

Coppock, E. 2008. *The Logical and Empirical Foundations of Baker's Paradox*. PhD thesis, Stanford University. `http://www.mendeley.com/profiles/elizabeth-coppock/` (accessed Feb 19 2010).

Crouch, R., and J. van Genabith. 1999. Context change, underspecification, and the structure of glue language derivations. In Mary Dalrymple (Ed.), *Syntax and Semantics in Lexical Functional Grammar: The Resource-Logic Approach*, 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. 2001. *Lexical Functional Grammar*. Academic Press.

Dalrymple, M., R. M. Kaplan, J. T. Maxwell, and A. Zaenen (Eds.). 1995. *Formal Issues in Lexical-Functional Grammar*. Stanford, CA: CSLI Publications. URL: `http://standish.stanford.edu/bin/detail?fileID=457314864`.

de Groote, P. 1999. An algebraic correctness criterion for intuitionistic multiplicative proof-nets. *Theoretical Computer Science* 224:115–134. `http://www.loria.fr/~degroote/bibliography.html` (accessed 19 Feb 2010).

Goddard, C. 1998. *Semantic Analysis: A Practical Introduction*. Oxford University Press.

Goddard, C. 2008. *Cross-Linguistic Semantics*. Amsterdam: John Benjamins.

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.

Jackendoff, R. S. 1973. The base rules for prepositional phrases. In S. R. Anderson and P. Kiparksy (Eds.), *A Festschrift for Morris Halle*, 345–357. Holt, Rinehard and Winston.

Jackendoff, R. S. 2002. *Foundations of Language*. Oxford: Oxford University Press.

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

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.

Kokkonidis, M. 2005. Why glue a donkey to an f-structure when you can constrain and bind it instead. In M. Butt and T. King (Eds.), *Proceedings of LFG 2005*. Stanford, CA: CSLI Publications.

Kokkonidis, M. 2008. First order glue. *Journal of Logic, Language and Information* 17:43–68. First distributed 2006; URL: `http://citeseer.ist.psu.edu/kokkonidis06firstorder.html`.

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

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.geocities.com/iddolev/pulc/current_work.html` (viewed June 2008).

Link, G. 1998. *Algebraic Semantics in Language and Philosophy*. Stanford CA: CSLI Publications.

Loeb, I. 2007. *Natural Deduction: Sharing by Presentation*. PhD thesis, Radboud Universiteit Nijmegen. URL: `http://dare.ubn.kun.nl/bitstream/2066/30014/1/30014.pdf` (seen Nov 2009).

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/`.

Moot, R. 2002. *Proof-Nets for Linguistic Analysis*. PhD thesis, University of Utecht. URL: `http://www.labri.fr/perso/moot/` (viewed June 2008), 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/`.

Pelletier, F. J. 1999. A brief history of natural deduction. *History and Philosophy of Logic* 1–31. slightly shortened version downloadable from: `//citeseer.ist.psu.edu/pelletier98brief.html`.

Perrier, G. 1999. Labelled proof-nets for the syntax and semantics of natural languages. *L.G. of the IGPL* 7:629–655. `http://www.loria.fr/~perrier/papers.html` (accessed 19 Feb 2010).

Pollard, C. 2007. Hyperintensions. To appear in *Journal of Logic and Computation*. URL: `http://www.ling.ohio-state.edu/~hana/hog/pollard2007-hyper.pdf` (viewed June 2008).

Restall, G. 2006. Proof theory and philosophy. chapters available at `http://http://consequently.org/papers/ptp.pdf`. An earlier version of this was 'Proof and Counterexample'.

Sadler, L., and R. Nordlinger. 2008. Apposition and coordination in Australian languages: an LFG analysis. ms under review. URL: `http://privatewww.essex.ac.uk/~louisa/aal/apposition6.pdf`.

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

Troelstra, A., and H. Schwichtenberg. 2000. *Basic Proof Theory*. Cambridge University Press. 2nd edition.

Wierzbicka, A., and C. Goddard. 2002. *Meaning and Universal Grammar - Theory and Empirical Findings, vols I and II*. Philadelphia, PA: John Benjamins.