# Proof-nets as Logical Forms for LFG

Avery D Andrews
ANU, Jan 2007
(substantial revision of Proof-nets as Linguistic Structures for LFG)
(typo and minor phrasing corrections Jan 18)

Although 'glue semantics' appears to be generally accepted as LFG's present account of semantic composition, its uptake amongst LFG practitioners doesn't actually seem to be that extensive (and a number of well-known syntacticians have claimed to me that they are pretty much baffled by it). A possible reason for this is that the standard presentation of glue is based on term-assigning deductions in linear logic, a kind of apparatus that isn't very congruent with normal LFG mechanisms.[1] Here I will show how proof-nets, an alternative format for deductions in certain forms of linear logic, can be construed as a linguistic level of 'logical forms', related to the other levels in LFG by correspondence relations in a manner reminiscent of the c-structure-f-structure correspondence $\phi$, as discussed for example in Kaplan (1995). Perhaps presenting glue in a manner more characteristic of linguistics in general and LFG in particular will make it easier to grasp, especially for LFG syntacticians.

Proof-nets are sometimes described as rather arcane and difficult, and indeed they can be, but the ones needed for LFG glue semantics are not; in fact these are essentially the same thing as a limited form of lambda-calculus term. The intended prerequisite for most of the paper is a grasp of LFG and basic formal semantics (especially the basics of the lambda-calculus, as may be found for example in Partee et al. 1993:358-343); towards the end, some portions might be more demanding.

For preliminary motivation, consider what might be involved in doing the semantic composition for the sentence *Bert yawned*, assuming a standard Frege-Montague system of semantic types where *Bert* is of type $e$ (for entity), *yawned* of type $e{\rightarrow}p$ ($p$ for proposition, following the usage of hyperintensional semantics (Pollard to appear), whatever we take propositions to actually be).[2] Representing types as trees with nodes labelled by subformulas in the obvious way, we would get:

(1) *Bert*     *Yawned*

$e$     $e{\rightarrow}p$ (daughters: $e$   $p$)

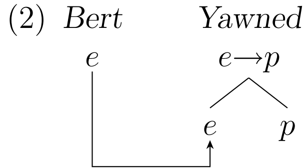
where $\rightarrow$ is the standard 'functional' type-constructor, whereby something of type $a{\rightarrow}b$ is something that accepts an argument of type $a$ to produce a result of type $b$ (in Montague grammar notation, this is often written as $<a, b>$).

If confronted with these items, and urgently requested to do something useful with

[1]And the fact that most early presentations were based on the Gentzen sequent calculus, a rather different form of logic than the natural deduction methods that most linguists get exposed to, probably didn't help.

[2]For some background on the origins of type theory, see Casadio (1988).

them, an obvious thing to try would be to connect the $e$ of *Bert* as 'output' to the $e$ of *Yawned* as 'input', like this:




The rationale for doing this would be the intuitive idea that a one-place predicate such as *Yawned* wants an entity as argument in order to produce a proposition, which *Bert* provides, via the link.

The proof-nets we will be using can be regarded as a disciplined version of this way of thinking. To then view them as constituting a part of a theory of the syntax-semantic interface, we need to do a few more things, such as show how they are connected to meanings as standardly conceived, and how they are constrained by the grammatical structure. There are also several layers of complexity in what is needed for these tasks, so we will procede in stages, looking first at application of predicates to arguments of simple types, then abstraction (functional types), and finally tensors (product types).

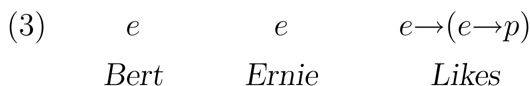
**1 Simple Arguments**

In this section we'll consider the treatment of cases where a predicate (in formal semantics, typically a function) applies to an argument of a simple type, such as for us, $e$ or $p$. This covers most cases of 'predicate argument structure' in LFG, except for functional control verbs where the controller is an argument of the matrix verb (Asudeh 2002, 2005), as well as many adjuncts (the ones that don't apply semantically to the subject as well as the proposition), and various other operators, such as epistemic modals and negation. We won't go through a huge range of examples, just a few representative ones.

## 1.1 Hookups

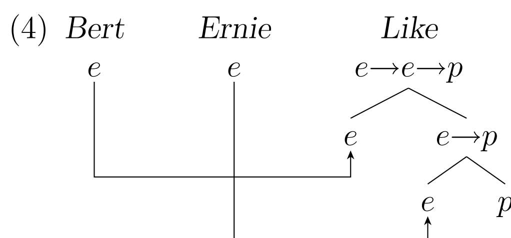
Our motivating example (1) isn't very informative, because it's too simple, and there's only one possible connection pattern that fits the semantic types. We'll start by looking at the slightly more complex case of transitive verbs.

Consider the semantic types for the words of the sentence *Bert likes Ernie*:



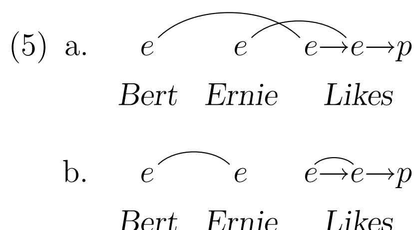
where the verb is presented in 'curried' form, as something that applies to one argument (typically the least active one) to produce something that applies to another to produce a proposition[3]

The obvious problem here is that there's more than one way to connect the different tokens of the same type $e$. The intended reading (given the convention above of applying the least active argument first) would be plausibly represented by:

(4)



where we have begun to follow the convention of omitting rightmost brackets with →.

Unfortunately, even if we ignore the issue of the directionality of the connections (which in fact doesn't need to be represented), there are two more ways of hooking up the $e$'s, one corresponding intuitively to a sensible (although wrong) reading of the sentence, the other not. Below we show these in an alternative, more compact, format, where the formulas aren't expanded as trees, and the connections are written as over-arcs:

(5)



We need some way for (5a) to be excluded as a reading for *Bert likes Ernie*, and presumably for (5b) to be excluded from consideration as a reading for anything. There is also the question of exactly how these things are related to logical forms presented in the more usual formats.

We will start with the task of excluding (b), then show how to construe proof-nets as more conventional-looking logical forms, and finally examine how the grammar imposes constraints. This amounts to first developing some internal properties of the proposed level, and then examining how it connects to the rest of the syntax.

A quite useful idea that can be used to rule out (b) is a concept of 'polarity', defined over the semantic type-occurrences and their subformula occurrences. The standard polarities were unfortunately reversed in Andrews (2004a) (for reasons that seemed good at the time), but here the standard ones will be used, which can be justified by thinking of the network as a system into which lexical meanings are dropped as 'inputs' (therefore negative), and a final result appears as 'output' (positive).[4]

[3] As argued for cogently, if not perhaps 100% convincingly, by Marantz (1984).

[4] The notion goes back quite a long way, at least as far as Kelly and Mac Lane (1971), where it is the 'variance' of the nodes in the graphs.

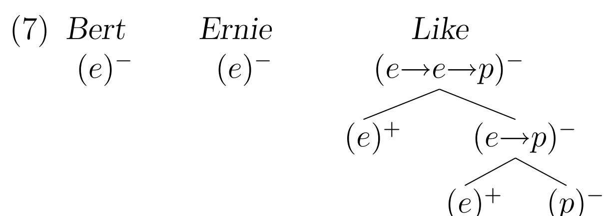
To formulate the rules, we need a bit more terminology, which is that a type-expression of the form $a \rightarrow b$ is called an 'implication' with antecedent $a$ and consequent $b$. The rules are then:

(6) a. An entire type-expression is negative

b. The consequent of an implication has the same polarity as the entire implication

c. The antecedent of an implication has the opposite polarity as the entire implication.

In expanded tree format, with polarities written in, the constructors of (3) will be:

(7) 


Following the usage of type-logical grammar, where proof-nets are widely employed, it is useful to call the type-expressions that are to be combined a 'frame'. For maximal elegance of the hookup rules, it is furthermore useful to introduce into the frame one additional type-expression, of type $p$, with positive polarity.

The basic requirements on linking are then:

(8) a. negatives connect one-to-one with positives

b. the types of connected nodes must match

If one thinks of content as 'flowing' through the network, it will then go from negative to positive along the links, which is a bit counterintuitive,[5] but follows from thinking of a proper name such as *Bert* as a both an input into the whole network (therefore standardly negative), and a provider of content to its predicate. The flow idea also motivates a negative-to-positive directionality. Intuition aside, the requirements of (8) suffice to rule out (5b), while allowing the two hookups of (3/7) that we have deemed to be sensible.

But for the one-to-one requirement to be satisfied, we need to add to the frame an extra basic type $p$, of positive polarity. This can be thought of as stating a requirement that the semantic contributions fit together to form a proposition, similar in both spirit and function to the specification of S as the 'initial category' of a phrase-structure grammar. The fact that there is only one additional positive also makes the logic of meaning-assembly 'intuitionistic', as discussed in more advanced treatments such as Crouch and van Genabith (2000).

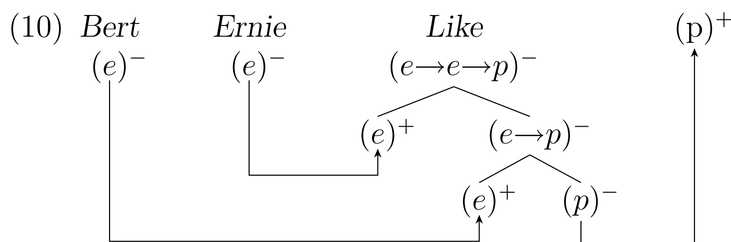
[5]Hence the polarity-reversal in Andrews (2004a).

To state linking more formally, we will want another bit of terminology, which is that the 'literals' in a type-formula are the subformulas (occurrences) that are basic types (here, $e$ and $p$), rather than formed by implication or any other type-constructor. We can now characterize linking as follows:

(9) A linking is a one-to-one correspondence from negative to positive literals that preserves semantic type.

'One-to-one correspondence' means that every negative must target a different positive, and every positive must be targetted by some negative (without the extra positive added to the frame, this last requirement wouldn't be satisfied, and we'd have to say 'every positive except for one must be targetted by a negative'). The links are often called 'axiom links', because they correspond to uses of the axiom $A \vdash A$ (given $A$, you can conclude $A$) in 'Gentzen sequent' proofs, which we won't be looking at, from which proof-nets were originally derived.

With the additional positive, and the axiom-links drawn in, the other linking for (9) becomes:

(10)



Technically, the rules for what we have described as 'hookups' are called 'proof-structures'. In order for them to be 'proof-nets', and therefore, adequate as logical forms, an additional criterion must be satisfied, the 'correctness criterion', which we will be developing below. But first, we'll show they can be read as conventional-looking logical forms.

## 1.2 Hookups as Formulas

The standard way to use type-expressions to help depict meanings for sentences is to use them as part of the syntax for 'logical forms', presented as formulas in some kind of artificial language. One could go on at some length about what a logical form is supposed to be; here I will take the position that it is a structure from which one can determine the entailments and other meaning-based properties of a sentence in a language-universal way, given information about the meanings of the basic elements. LFG f-structures could therefore be taken as logical forms as long as only predicate-argument structure conveyed by grammatical relations is considered (and coreference, given an ANTECEDENT grammatical function), but fail once scope phenomena associated with adjuncts, quantifiers, etc. are considered.

A syntax for logical forms usually (always?) contains something along the lines of an 'application' construction, for applying a function to an argument, often notated

along the lines of '$f(a)$' where $f$ is the function and $a$ the argument. The proof-net literature contains various schemes[6] for extracting expressions in such formal languages from proof-nets, but an important point is that these aren't actually necessary, since the proof-nets themselves can be regarded as being such expressions by a fairly simple change in perspective.

For our structures so far, this change can be implemented by reconstruing subtrees of the form (a) below as being of the form (b), where the directionality of the arrows is supposed to be from daughters to mothers:

(11) a. $(a{\to}b)^-$ with daughters $a^+$ and $b^-$   b. $(a{\to}b)^-$ with arrow to $b^-$; $a^+ \longrightarrow b^-$

We can read this as a syntax tree for a predicate-first logical form by viewing the downward pointing arrow in (b) as a left daughter-to-mother tree link, and the rightward pointing one as a right daughter-to-mother tree link (i.e. swivelling the diagram a bit more than 90° around the $b^-$ node), getting:

(12) $b$ with daughters $a{\to}b$ and $a$

These are minor, local, changes, to the extent that they are changes at all, so the effect is to reconstrue the type-tree as a structure-tree in the standard format. Since the directionality of the arrows in the tree-format of the proof-net is always upward, we'll leave the arrowheads out.

If we remodel (10) along these lines, and reposition the lexical type-trees in accordance with a conventional tree format, we get the following, where the axiom links have been rendered as dotted lines to visually distinguish them from the others:

(13)

```
                p
                ↑
                p
          ______|______
        e→p            e
     ____|____         ↑
  e→e→p      e         e
  Like       ↑        Bert
             e
           Ernie
```

If we now simply collapse the nodes connected by the axiom links (not a problem due to the constraint that these always have the same (basic) type-label), (13) becomes a syntax-tree for a conventional Montogovian logical form such as *Like*(*Ernie*)(*Bert*),

[6] Such as the 'maximal labelling' of Perrier (1999) and the 'semantic trips' of de Groote and Retoré (1996) and Morrill (2005).

with left-association of function application, as usual. The differences between (10) and (13), with or without collapsing of nodes connected by axiom-links, are trivial enough so that both can be considered as different views of the same objects. Note in particular:

(14) a. Given a collasped form, we can recover the uncollapsed one by expanding the basic types into pairs connected by an axiom link

b. Although polarity can't be assigned consistently to the nodes of a collapsed tree, we can uncollapse it and then assign polarity (with rules that are basically the same as with the regular proof-net format, although the details of formulation would be a bit different).

c. The relationship between the two formats would be a bit messier if we didn't have the additional positive in the frame.

So we can regard proof-nets as being equivalent to conventional logical forms, at least insofar as the application construction is concerned, and this equivalence will be extended further below.[7]

## 1.3 Constraints from Syntax

Our next problem is to account for the way in which the syntax constrains the linking, and thereby the interpretations. The basic idea is to use f-structure labels as an additional source of constraint on linkings, alongside of the semantic types. This affords an appropriate time to introduce the conventional format for 'meaning-constructors', the semantic contributions which are to be assembled in glue. In conventional notation, the constructors for *Bert likes Ernie* would be written as in (a) below, if the f-structure was (b):

(15) a.

$$\begin{aligned} \textit{Bert} &: g_e \\ \textit{Ernie} &: h_e \\ \textit{Like} &: h_e \multimap g_e \multimap f_p \end{aligned}$$

b.

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

The '$\multimap$ ' symbol here represents 'linear implication', about which, more below, but right now we can just regard it as an orthographic variant of the '$\rightarrow$' symbol we've

[7] The point that proof-nets were equivalent to logical forms was first made, to my knowledge, by de Groote and Retoré (1996), although not, it seems to me, in as direct and elementary way as it is being made here.

been using for the functional type-constructor.[8] So the meaning-constructors are just type-trees tagged with f-structure labels as well as semantic types.

Now we can regard the placement of axiom-links as constrained by the f-structure information as well as by the semantic type information, so that for (a) below, the only possible hookup is (b), which is the logical form for *Bert likes Ernie*, but not *Ernie likes Bert*.

A further condition is that the additional positive basic type needs to be tagged with the label of the entire f-structure, expressing the idea that we're looking for a meaning of propositional type for this f-structure. In our concise, non-tree-expanded notation, this additional positive will be separated by the others by the '⊢' (turnstyle) symbol, to indicate its special status. This is the notation for logical 'sequents', about which more below.[9]

(16) a. $g_e \quad h_e \quad h_e \multimap g_e \multimap f_p \quad \vdash \quad f_p$

*Bert* *Ernie* *Likes*

b. $g_e \quad h_e \quad h_e \multimap g_e \multimap f_p \quad \vdash \quad f_p$

*Bert* *Ernie* *Likes*

Now, clearly, if we swapped the f-structure labelling of the '*Bert*' and '*Ernie*' terms, then it would be the other of the two previously possible linkings that would be allowed by the f-structure information, so that we'd get the appropriate logical form for *Ernie likes Bert*.

So the final step is to explain where the f-structure tags come from. Here the answer is exactly the same is in standard presentations of glue: from the lexicon, via instantiation. So the words in the two sentences we have been considering would contain the meaning-constructors in uninstantiated form, with ↑-metavariables in the place of f-structure labels:

(17)
$$\begin{aligned} \textit{Bert} &: \uparrow_e \\ \textit{Ernie} &: \uparrow_e \\ \textit{Like} &: (\uparrow \mathrm{OBJ})_e \multimap (\uparrow \mathrm{SUBJ})_e \multimap f_p \end{aligned}$$

Lexical entries contain uninstantiated meaning-constructors along with f-structure equations, which are instantiated in the same way that the f-structure equations are, with ↑ referring to the f-structure correspondent of the preterminal node over the lexical item, and the resulting f-structure designators being resolved (constructively, it would appear, although I haven't seen this point discussed explicitly in the literature).

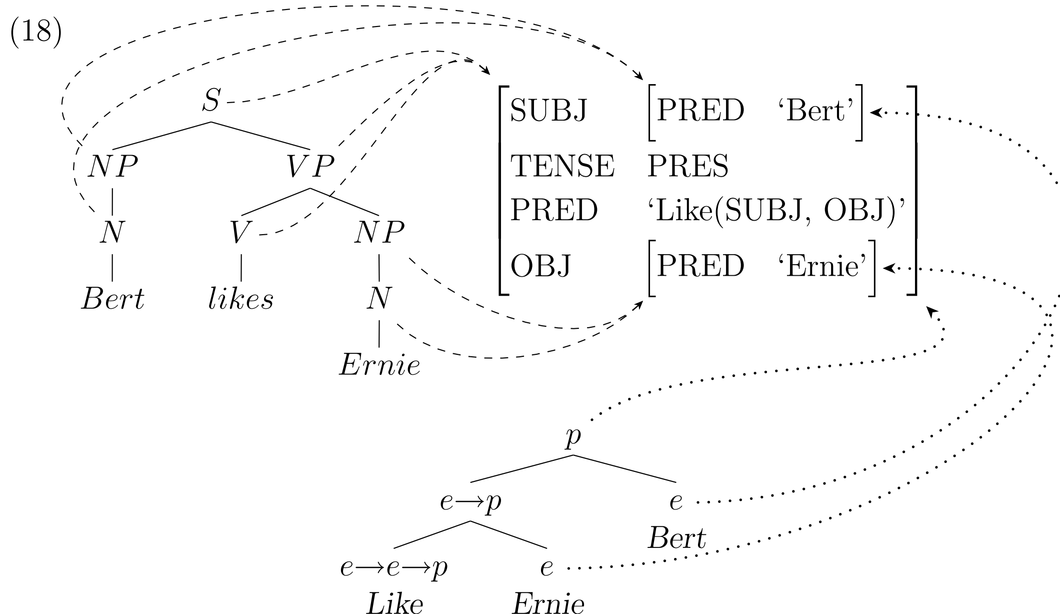
[8] Indeed, some presentations of glue, such as Lev (2006b), use → instad of ⊸ in meaning-constructors.

[9] Briefly, the formulas to the left of ⊢ are assumptions from which the formula to the right can be deduced. The restriction to a single formula to the right of the turnstyle imposes the 'intuitionistic' restrictions on the logic.

Since the nodes of the proof-net of (16b) are tagged with f-structure labels, we can think of it as a structure (a tree, in fact, as we have already discussed) some of whose nodes are related by a correspondence relation to an f-structure. This will be a correspondence between a kind of semantic representation (really, just a representation of how meanings are to be composed; we should perhaps call it a 'semantic compositional structure'), and the f-structure, so it would be reasonable to call it $\sigma$. The usual practice with correspondences between functional and semantic structure has been to run them from the former to the latter, but we will see soon that for $\sigma$ as constructed here, it is the reverse direction that is preferable, since more than one semantic node can correspond to the same functional substructure, but not vice-versa.

In this format, a full linguistic structure for *Bert likes Ernie* would look like this ($\phi$ dashed, $\sigma$ dotted):

(18)




In addition to opposite directionality, another major difference between the $\phi$ and $\sigma$ correspondences is that the latter is defined only for the literals (basic types), while the former is defined for all c-structure nodes (but not lexical items themselves).

The overall architecture then works as follows: the annotated tree produces both an f-description, which is solved to produce an f-structure, and a collection of semantic contributions (instantiated meaning-constructors), whose literals are linked to the f-structure by the $\sigma$ correspondence, which is somewhat like the c-structure-to-f-structure correspondence $\phi$, although different in detail and opposite in direction. These semantic contributions are then combined by in effect fusing the literals of the contributions (implemented by connecting them with axiom-links), subject to the requirement that their f-structure correspondents match, and to certain further constraints, as will be described shortly (the 'correctness criterion'). As a result, the integrated semantic contributions can be viewed as constituting a logical form for the sentence.

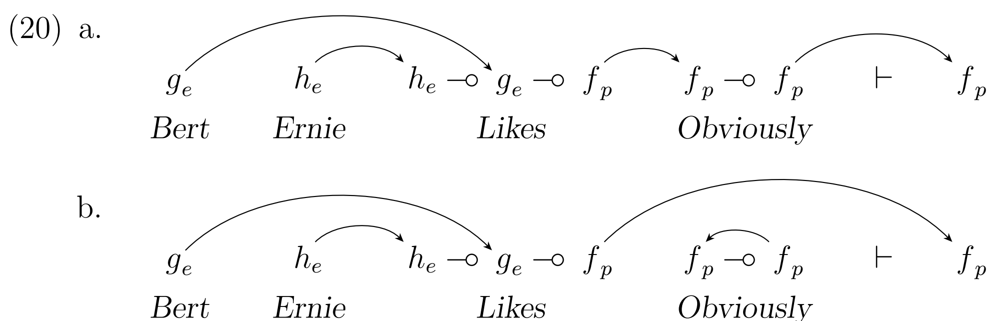
## 1.4 Correctness and Logic

So far is, however, actually, not so good, because when we extend our account to adverbs such as 'obviously', and similar operators, a problem emerges. Consider a sentence such as *Bert obviously likes Ernie.* Plausible instantiated meaning-constructors for this would be (significantly oversimplified; see Dalrymple (2001) for discussion):

(19) *Obviously* : $f_p \multimap f_p$
*Bert* : $g_e$
*Ernie* : $h_e$
*Like* : $h_e \multimap g_e \multimap f_p$

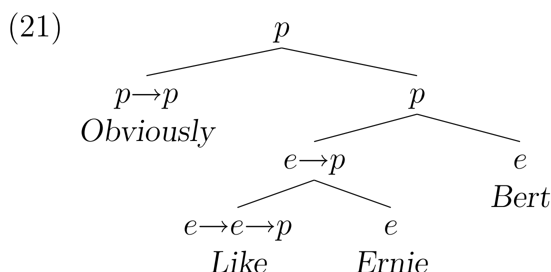
Unfortunately, there are two possible ways of hooking these constructors up, represented as follows in the 'linked sequent' format of (16b):

(20) a. b.




(a) corresponds to the plausible tree-structure logical form:

(21)




(b) on the other hand satisfies all of our principles so far, but doesn't represent a conceivable reading of the sentence.

To rule it out, we need an additional constraint, called the 'correctness criterion'. which can be approached from many different angles, and formulated in a wide variety of ways. The underlying idea involves logical proofs. We can regard the format of (a) as representing a proof that the type-formulas to the left of the $\vdash$ can be used to produce the one on the right, using the logical rule of 'modus ponens', which says that, given 'If $A$, then $B$', and '$A$', you can conclude '$B$'. Each axiom-link can be taken as representing an application of modus ponens (the names mismatch here, because the technical origin of the proof-nets was a different format of logic, the one-sided

Gentzen sequent calculus, and this description is oversimplified anyway), resulting in the ultimate deduction of the $f_p$ that appears to the right of the $\vdash$ (an intuitive sense of this is sufficient for here, we'll give a precise account later).

From this perspective, the problem with (b) would appear to have something to do with the fact that the *Obviously* premise isn't being used to produce the final conclusion, which certainly does follow from the material to the left, by the deduction indicated by the linkings, but only by virtue of ignoring the *Obviously* premise. In effect, there seems to be a requirement that all of the formulas on the left of the sequent be used in the production of the conclusion. This is a feature of what is called 'relevant logic', so called because it imposes a condition that all of the premises of a relevantly-valid argument actually be used in the production of the conclusion.

In terms of the proof-nets we have been considering so far, this condition can be imposed by requiring that the proof-net be connected, that is, not have any two sub-parts with no links between them (things will get a bit more complicated later, and note that 'no cycles' is another possibility, equivalent in these simple cases, but not in more complex ones that we'll see later). The actual logic we're using is not only relevant, but 'linear', meaning that in addition, each assumption can only be used once; this is enforced by other features of the proof-net scheme, such as the one-to-one linking (and, as already mentioned, it's intuitionistic, due to the restriction to a single conclusion).

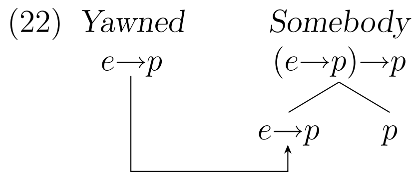
In addition to showing the need for a further condition on correctness, the (a) assembly also provides an example where several literals in the logical form correspond under $\sigma$ to a single f-structure. It would be a good exercise to write out the full f-structure and logical form, with $\sigma$-correspondence.

With this form of account of adverbials and similar elements in place, we can discuss a range of scoping phenomena that can't be handled explicitly in standard LFG, due to the 'flat' organization of the relevant meaningful elements in f-structure, along lines sketched in Andrews (to appear). The deficiency of f-structures as logical forms is therefore partially remedied.[10] There are however further scope phenomena, involving quantifiers, for which what we have done so far is insufficient.

## 2 Quantifiers and 'Higher Order' Arguments

Since the work of Barwise and Cooper (1981), it has generally been accepted that quantificational determiners such as *every* and *some* should be analysed as relations between sets, e.g. *every dog barked* is true iff the set of dogs is a subset of the set of barkers. So the type of quantified NP such as *every dog* or *somebody* (technically, these are the 'quantifiers' in the terminology of Barwise and Perry) will be $(e{\rightarrow}p){\rightarrow}p$. Then, given something of type $e{\rightarrow}p$, such as an intransitive verb, we could simply apply the quantifier to this thing to get a result of type $p$. In a proof-net, we could implement this by 'lopping' matching branches off the predicate and argument type trees, and connecting the resulting $e{\rightarrow}p$-type stubs, like this:

[10] Raising the question of whether f-structures should still represent predicate-argument relations and provide the structural basis for the Completeness and Coherence constraints (Kuhn 2001).




The lopping tactic will work for a significant variety of cases, such as *try*-type verbs as analysed by Asudeh (2002, 2005), or subject-oriented adverbs as analysed in Dalrymple (2001), but famously fails for quantifiers, since it can deal with neither quantification into object position, as in (a) below, nor wide-scope readings such as that of (b) below:

(23) a. Rover likes everybody

b. Somebody seems to have yawned

These are standardly analyzed by the use of either bound variables, or equivalent structures of combinatory logic using combinators. In the Barwise-Cooper treatment, logical forms for (23) might be:

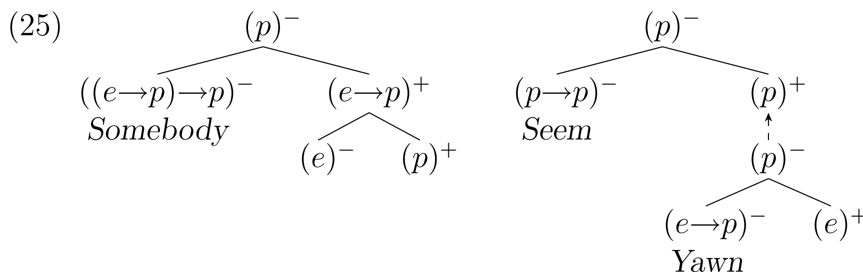
(24) a. $Everybody(\lambda x.Likes(x)(Rover))$

b. $Somebody(\lambda x.Seem(Yawn(x)))$

Lopping clearly won't help here: if we tried to deal with (a) by applying the subject first, then it would be quantification of subjects that was problematic, and whichever we did, *Everybody likes somebody* would be problematic. (b) is likewise intractable, so we would appear to need something equivalent to variable binding in our proof-nets. Such a thing turns out to already be there.

## 2.1 Positive Implications

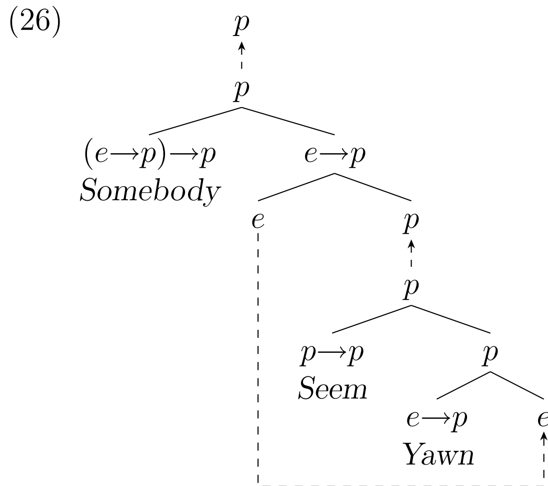
Using expanded tree-format, we could say that the problem is that we have the following two pieces to fit together:




And the rules we have so far already provide a solution.

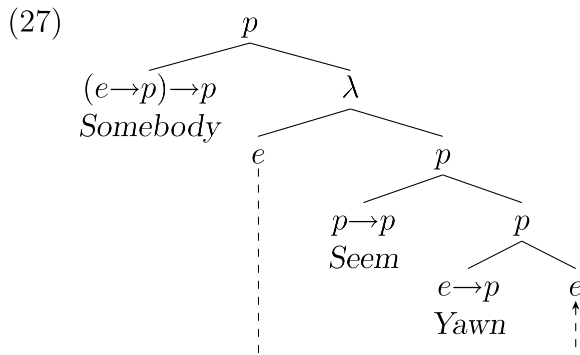
We have a negative $e$ in *Somebody* that satisfies the constraints for linking to the positive one in the *Seem*-*Yawn* subtree, and a negative $p$ in this latter that satisfies the constraints for linking to the positive $p$ in *Somebody*. Linking these up we get:

(26)




(No syntactic information yet). This looks a lot like a standard logical form using lambda-abstraction, with the linking of the two $e$-literals doing the work of variable-binding. We can increase the resemblance by writing the positive $e{\rightarrow}p$ as $\lambda$ (its type being so easily reconstructible from those of its daughters), and collapsing the axiom-linked $p$'s:[11]

(27)




Collapsing the two $e$'s would be conceptually better, but notationally awkward, so we won't do it (when the other literals are collapsed, one can think of the links connecting 'variables' to their binding sites as being a convenient notation for identity, like functional control links in f-structure notation).

We seem to have a plausible representation, but we're not done yet, because (a) we need an enhancement of the correctness criterion (b) we need to work out the relationship to the syntax (c) we need to deal with the fact that the same hookup pattern is also available for the simpler cases that we first managed with lopping.

Fixing the correctness criterion is best combined with explaining how the proof-nets are being reoriented in these examples. Specifically, for positive implications, they aren't: the positive $e{\rightarrow}p/\lambda$ subtree appears in the same orientation in (27) as it would in our regular proof-net (which is upside-down with respect to the standard presentation:

[11]For reduced redundancy, it would also be reasonable to relabel the negative complex type nodes with some symbol standardly denoting application, although for greater readability, we won't do this here.

logicians seem to like their trees sitting on their roots, while linguists like them hanging from them).

For this reason we might think to reconstrue the positive implication-links as two equivalent upward pointing arrows, but for our present version of the correctness criterion, it's better to treat them a bit differently.[12] So our reconstrual of a positive implication can be generally depicted like this:

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

$(e)^-$ $(p)^+$

The thick arrow will belong, together with the axiom-links and both thick arrows for the negative implications, to a structure introduced by de Groote (1999) called the 'dynamic graph',[13] while the thin one won't. If you examine (26) and contemplate what the dynamic graph would be, you'll see that it is a tree (the structure given, with the link from the positive implication to its $e$ antecedent omitted).

And it is a straightforward consequence of de Groote (1999) that the correctness criterion will be satisfied iff:

(29) a. The dynamic graph is either acyclic or connected

b. Every path in the dynamic graph starting at the antecedent of a positive implication passes through the consequent of that implication on its way to the root

The requirement (b) in particular looks just like the standard requirement on bound variables that they appear 'within the scope of' their binder. There is however one important difference between the lambda-binding implemented by these proof-nets and what is generally used in formal semantics, which is that the proof-net binding is only capable of binding a single position. This will afford difficulties if we want to deal with examples involving anaphora, such as *everybody washed himself.* This problem will be considered in section 4, where we look at tensors. Meanwhile, lambda-terms where each lambda binds one and only one occurrence of a variable are called 'linear'.

People who are so inclined can spend a great deal of time learning about the various formulations of the correctness criterion (Moot (2002), ch 4 is a good place to start), but the executive summary for linguists is that an implicational proof-structure satisfies the correctness criterion and is therefore a proof-net if and only if it looks like a sensible linear lambda term.

[12]However, the most often encoutered formulations of the correctness criterion treat them symmetrically; the asymmetric treatment used here appears to be a feature of formulations derived from de Groote's (1999) algebraic formulation of the criterion, and would seem to be best suited to intuitionistic implicational systems, due to the asymmetry of implication.

[13]Closely following a similar notion introduced by Lamarche (1994), although I must confess that I have not yet managed to understand much in this paper.

## 2.2 Relating to the F-structure

Now that we have an account of the logical forms of wide-scope structures, our next issue is how to get them and their narrow-scope variants. To sharpen the question, consider the two logical forms we want for *Somebody seems to yawn*, together with a reasonable f-structure for this sentence:

(30) a. $Somebody(\lambda x.Seem(Yawn(x)))$

b. $Seem(Somebody(\lambda x.Yawn(x)))$

(31)
$$f:\begin{bmatrix} \text{SUBJ} & g:\begin{bmatrix}\text{PRED} & \text{`Somebody'}\end{bmatrix} \\ \text{TENSE} & \text{PRES} \\ \text{PRED} & \text{`Seem(XCOMP)'} \\ \text{XCOMP} & h:\begin{bmatrix}\text{SUBJ} & g:[\ ] \\ \text{PRED} & \text{`Yawn(SUBJ)'}\end{bmatrix} \end{bmatrix}$$

Given the obvious instantiated constructors for *seem* and *yawn*:

(32)
$$\begin{array}{lcl} Seem & : & h_p \multimap f_p \\ Yawn & : & g_e \multimap h_p \end{array}$$

We can get the two readings by using one of the following two instantiated constructors for *somebody*:

(33)
$$\begin{array}{lcl} Somebody & : & (g_e \multimap f_p) \multimap f_p \\ Somebody & : & (g_e \multimap h_p) \multimap h_p \end{array}$$

(a) gives rise to the wide-scope reading, as can be seen by looking at the sequent-format version of the proof-net:

(34) $g_e \multimap h_p \quad h_p \multimap f_p \quad (g_e \multimap f_p) \multimap f_p \ \vdash \ f_p$

*Yawn* *Seem* *Somebody*

The syntactic constraints and correctness criterion jointly allow no other linking pattern.

For the narrow-scope reading, on the other hand, there are two possibilities, with and without lopping:

(35) a. $g_e \multimap h_p \quad (g_e \multimap h_p) \multimap h_p \quad h_p \multimap f_p \quad \vdash \quad f_p$

*Yawn* *Somebody* *Seem*

b. $g_e \multimap h_p \quad (g_e \multimap h_p) \multimap h_p \quad h_p \multimap f_p \quad \vdash \quad f_p$

*Yawn* *Somebody* *Seem*

What to do about this will concern us shortly; briefly, we view it as being a difference without a difference ($\eta$-equivalence, for people who already know a bit about the lambda-calculus).

Our present question is how to get the two constructors of (33) in the first place. The standard approach is to use quantification in the glue logic (which Kokkonidis (to appear) shows how to construe as first order quantification). Here we will instead use standard LFG ideas about correspondences between structural levels. The difference between the wide and narrow scope readings lies in which f-structure the $p$ type variables in the quantifiers meaning-constructor refer to. The invariant is that whatever they refer to, they both refer to the same thing.

This can be seen as a typical case of LFG architecture where there is a constraint that two things correspond to the same thing, but not a full specification of what that further thing must be. The most straightforward way to achieve this in LFG notation is to represent the f-structure correspondence information for the $p$-terms with a 'local variable' (Dalrymple 2001), like this:

(36) *Somebody* : $(g_e \multimap \%H_p) \multimap \%H_p$

The intended interpretation is that the local variable places no constraint on what piece of f-structure its literal corresponds to, as long as all literals with that local variable correspond to the same piece of f-structure. This idea could be expressed a bit more explicitly by reformulating the constructor like this, where $\tau$ is the mapping from meaning-constructor literals to semantic types:

(37) *Somebody* : $(X \multimap Y) \multimap Z$

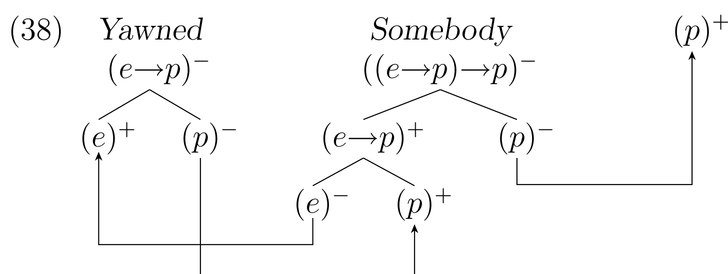
$$\sigma(X) = g \qquad \tau(X) = e$$
$$\sigma(Y) = \sigma(Z) \qquad \tau(Y) = p$$
$$\tau(Z) = p$$

The function of the universal quantifiers in standard glue can thus be taken over by the workings of LFG correspondence architecture, not with any specific empirical consequences, as far as I know.

In particular, there is no effect on the basic glue result (Dalrymple et al. 1997) that the logically coherent possibilities for quantifier interpretation can be obtained without imposing any extrinsic constraints: the correctness criterion does it all (although of course it wouldn't account for purely grammatical constraints on scope).

## 2.3 Lopping, $\eta$-reduction, and Common Nouns

One goal of this subsection is to integrate common nouns into the analysis, but to do this we need to first consider the relationship between 'lopped' and 'non-lopped' analyses. The issue is that for a sentence such as *somebody yawned*, we have both the lopped analysis of (22), and this unlopped one:

(38)



If we convert these proof-nets to regular logical form notation, they become:

(39) a. *Somebody*(*Yawned*)

b. *Somebody*($\lambda x.$*Yawned*($x$))

These are related by the lambda-calculus rule of $\eta$-conversion (or reduction, if one is only using it to derive the shorter formula from the longer one):

(40) $F \equiv_\eta \lambda x.F(x)$

This equivalence says that if you take a function $F$, apply it to a variable $x$ of appropriate type, and then lambda-bind that, you just have $F$ again.

In terms of the syntax of the formulas, what $\eta$-conversion does (and also $\beta$-conversion, which we'll briefly discuss later) is to group them into equivalence classes, so that the 'real objects' in the system of formulas are the classes of equivalent formulas, not their particular orthorgraphic representatives. So, via $\eta$-conversion, (a) and (b) of (39) are supposed to be two representations of the same thing.

The notion of $\eta$-conversion extends naturally to proof-nets; in the sequent notation for proof-nets we an formulate it concisely as follows (applying recursively):

(41) $\ldots\ (a \multimap b)^+\ \ldots\ (a \multimap b)^-\ \ldots$

$\Updownarrow$

$\ldots\ (a \multimap b)^+\ \ldots\ (a \multimap b)^-\ \ldots$

Having to deal with equivalance classes of proof-nets is not an entirely welcome step, since part of the motivation of proof-nets was to replace certain equivalence classes of

proofs with single objects, in the form of proof-nets. However, proof-nets only reduce the role of equivalance classes, and may not manage to eliminate them entirely.[14] In any event, this equivalence class actually looks more natural over proof-nets than over standard formulas, since it can be seen as a decision to regard two matching axiom-links as a single one, hence the greater thickness of the link in (b).

Usually, in proof-net-based work, people don't bother with lopping/$\eta$-reduction, but instead work with formulas in $\eta$-expanded form, to the extent required in order to connect only literals with axiom-links. In linguistics, we can do as we like, and I see no reason to even have a policy on the matter. When we look at the connections to logic more closely, we'll see that there's a version of $\eta$-conversion for conventional proofs, as well as for proof-nets and lambda-terms (as would be expected, since these are all just different representations of the same thing).

Now we're ready to look at how quantificational determiner such as *every* combines with a common noun, such as *dog*. Semantically, common nouns are taken as being of type $e{\rightarrow}p$, so that quantificational determiners are then of type $(e{\rightarrow}p){\rightarrow}(e{\rightarrow}p){\rightarrow}p$, as mentioned above. The common noun $e$'s and $p$'s must obviously be in some way localized to the f-structure of the NP that the noun appears in, and here we will implement this by simply associating both with the f-structure of that NP. Therefore the uninstantiated meaning-constructors for *every* and *dog* will be:

(42) $\mathit{Every} : (\uparrow_e \multimap \uparrow_p) \multimap (\uparrow_e \multimap \%H_p) \multimap \%H_p$
$\mathit{Dog} : \uparrow_e \multimap \uparrow_p$

By LFG, if the quantifier and common noun are introduced in the usual way in the same NP, the $\uparrow$'s in both constructors will instantiate to the same f-structure, and, by the polarity rules, the $e$ and $p$ in the first component of the quantifier will be negative and positive respectively, and vice-versa for the $e$ and $p$ of the common noun. So these constructors, instantiated and partially hooked up, can be depicted in either $\eta$-expanded form as (a), or $\eta$-reduced as (b):

(43) a. $g_e \multimap g_p \quad (g_e \multimap g_p) \multimap (g_e \multimap \%H_p) \multimap \%H_p$
*Dog* *Every*

b. $g_e \multimap g_p \quad (g_e \multimap g_p) \multimap (g_e \multimap \%H_p) \multimap \%H_p$
*Dog* *Every*

The result is a combined structure which behaves exactly like the quantificational pronouns we discussed in the previous subsection.

Standard glue makes use of the 'dummy attributes' VAR and RESTR to locate the $e$ and $p$ literals of the common noun constructor, and these are furthermore placed on a special 'semantic projection', symbolized with $\sigma$, which comes off the f-structure. This projection however has none of the empirical properties of a real projection, such as

[14] Although progress is continually be made, e.g. Hughes (2005).

a distinctive pattern of sharing across other structural levels (Andrews and Manning 1999), and is unnecessary for making the analysis work, so we abandon it here.[15] The important point is that thanks to the correctness criterion, the outgoing link from the negative $e$ in the first argument of the quantifier must be part of a dynamic path that returns back to the positive $p$ there, and can't go anywhere else, such as perhaps the positive $p$ in the second argument. There is furthermore nothing in the theory that says that semantic information of different types, such as $e$ and $p$, can't be associated with the same f-structure location. So the dummy attributes themselves do no work, and the projection they're supposed to be on has no properties, so the conclusion is that both should be dispensed with, unless one or both of these deficiencies is remedied.

It should be evident that if we combine the composite (43) with other constructors as in the previous subsection, we get semantic assemblies for sentences such as *every dog seems to bark*, *every man loves some woman*, etc., just as with quantificational NP such as *somebody* and *everybody*.

The next thing we'll look at is adjectives. For 'scoping' adjectives such as *alleged*, *former*, *self-styled*, etc., the basic treatment is pretty straightforward. Such adjectives appear to apply as 'operators' to the common-noun following them. Some, such as *alleged* and *former*, could be treated as being of type $p{\rightarrow}p$, but others, such as *confessed* and *self-styled*, appear to clearly apply to arguments of type $e{\rightarrow}p$, so the obvious semantic type for them would be $(e{\rightarrow}p){\rightarrow}e{\rightarrow}p$ (note that this would look more symmetrical if we included the parentheses around the final $e{\rightarrow}p$ subsequence, omitted by the convention for the right-associativity of $\rightarrow$). So the constructors for *confessed criminal* would be:

(44) $\mathit{Confessed} : (\uparrow_e \multimap \uparrow_p) \multimap \uparrow_e \multimap \uparrow_p$
$\mathit{Criminal} : \uparrow_e \multimap \uparrow_p$

It should be clear how these will fit together to produce an assembly that acts like a single common noun.

But, unfortunately, it isn't satisfactory to rest with such a simple picture of adjectives, for a number of reasons. Perhaps the simplest and most empirically pressing is that fact that this analysis doesn't allow for the combinations of adjectives with adverbs, as originally noted in the context of HPSG by Kasper (1995), and applied to the LFG by Dalrymple (2001). I won't go into this here; hopefully the reader will be able to comprehend the issues from Dalrymple's discussion. A more subtle problem is that 'scoping' adjectives such as *alleged* have somewhat different properties than 'intersective' ones such as *large*, which need to be dealt with somehow, and that the former have their scoping possibilities determined by the c-structure in ways that aren't immediately accounted for by standard LFG architecture (see Andrews (2004a) for some discussion in an LFG+glue context), and an account of their properties seems likely to require significant involvement of the internal structure of the meaning-side.

Consider for example a constraint discussed in Andrews (2004b), to the effect that while we have adjectives such as *alleged* with meanings such that 'an alleged murderer'

[15] See Andrews (2004a) and Kokkonidis (to appear) for further discussion.

is someone who someone has said is a murderer (perhaps in some special format, such as a declaration to the police), there is no adjective similar to **alleger*, such that 'an alleger murderer' is someone who has said that someone (else) is a murderer. Meaning-constructors for the attested *alleged* and impossible **alleger*, as well as the somewhat similar *confessed*, with expanded meaning-sides, might be (we've assumed a typical variable & two proposition treatment of the meaning-side of quantification):

(45)
$$\begin{array}{ll} \textit{alleged}: & \lambda P.\lambda x.Some(y, Person(y), Say(y, P(x))) : (\uparrow_e \multimap \uparrow_p) \multimap \uparrow_e \multimap \uparrow_p \\ *\textit{alleger}: & \lambda P.\lambda x.Some(y, Person(y), Say(x, P(y))) : (\uparrow_e \multimap \uparrow_p) \multimap \uparrow_e \multimap \uparrow_p \\ \textit{confessed}: & \lambda P.\lambda x.Say(x, P(x)) : (\uparrow_e \multimap \uparrow_p) \multimap \uparrow_e \multimap \uparrow_p \end{array}$$

The possible meanings seem to share the property that if you apply the composite $A(N)$ to some entity $a$, then the proposition $N(a)$ seems to in some sense a component of resulting proposition (although not an entailment). So, if our proposition is that John is a confessed murderer, then 'John is a murderer' is in a sense part of the content (it's what John says he his). At this point I don't know how to formalize this kind of constraint in the lambda-calculus (although there is some work on paths and reduction that seems like it might be relevant), but it would seem to involve the internal structure of the meaning-side in some way, and to be beyond what can be said by merely stating its type.

I think there are further indications that some aspects of the internal structure of meaning-sides belong in the grammatical representation, such as the fact that quantification is always extensional: in each world, the truth of $\mathcal{D}(P)(R)$ where $\mathcal{D}$ is a quantificational determiner, and $P$ and $R$ are one-place predicates, depends only on the extensions of $P$ and $R$ in that world, something which doesn't follow from the glue-type of quantificational determiners, but might be made to follow from some story about what the internal structure of their meaning-sides had to look like. But this is a topic beyond the scope of this paper.

## 2.4 The Architecture Reviewed

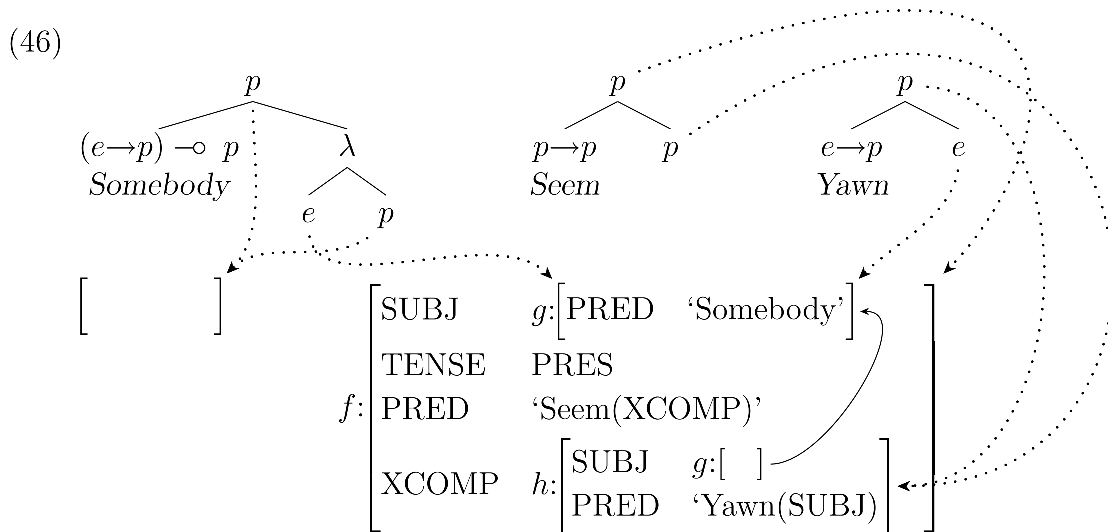
There is a significant similarity between the present conception of glue-semantics, and the way in which c-structure is related to f-structure. One way of looking at annotated phrase-structure rules is as small chunks of phrase-structure, consisting of a mother and immediate daughters, together with information about the relationships between their f-structure correspondents. Constructors in the format of (45) are similar in being partial structures with f-structure correspondence information. They are also similar in that there is an essentially binary convention on how they are to be put together. Given a collection of tree-pieces admitted by a phrase-structure grammar, one can regard the mothers as being objects with negative polarity, the daughters as objects with positive, and the task as being to connect each negative to a positive subject to the requirement that the phrase-structure categories match, with an additional positive of category S

being supplied, to express the idea that we're trying to produce a structure of type S.[16]

LFG differs from HPSG and Minimalism (but is similar to some but not all forms of Categorial Grammar), in being 'surface driven', in the sense that the assembled annotated c-structure determines the more abstract portions of the syntactic structure, by the f-description solution method in the case of the f-structure, and linear logic proofs for the semantics. On the present presentation, the semantic assembly can be presented as a matter of assembling pieces in a manner somewhat reminiscent of phrase-structure generation.

For the instantiated meaning-constructors produced by the annotated phrase-structure tree can be seen as a collection of structure-pieces, with constraints on a correspondence relation to the f-structure. A diagram of this situation for *somebody seems to yawn* is:

(46)




In this picture, the incomplete information about the f-structure correspondents of the $p$-literals in the quantifier constructor is represented by having these point to an empty piece of f-structure.

So far, the construction of the logical form looks a bit like that of f-structure itself, but there is a significant difference. F-structures are found by merely solving the equations provided by the annotated c-structure, with no further constructive step. For the logical forms, however, there is a further step, which is the linking of the literals. This is free, subject to the constraints that:

(47) a. Only literals with the same semantic type and f-structure correspondent can be merged

b. The correctness criterion must be satisfied

The correctness criterion allows two possible matches for the positive $p$ of *Somebody*, the (negative) outputs (top nodes) of either *Seem* or *Yawn*. These choices will force

[16] Phrase-structure rules construed in the way are in fact extra-logical axioms in a non-commutative linear logic with conjunction as the only connective.

the identification of the empty brackets with either the outer or the inner f-structure, respectively. From this choice, the wide and narrow scope logical forms emerge deterministically.

This version of LFG architecture has an interesting resemblance to Minimalism. In Minimalism, things start with a 'numeration', which is a collection of lexical-item occurrences, which are then assembled into an overt form and its interpretation by a computation. In this LFG architecture, the annotated c-structure mechanism delivers a collection of pieces reminiscent of a numeration, which also must be assembled, but only to deliver the interpretation, and subject to fairly strong constraints. In spite of these differences, the theme of assembly of things very much like lambda-terms is shared.

Furthermore, Andrews (to appear) explores the possibility, in the context of OT-LFG, of using a more autonomous assembly process for glue, whereby entries are chosen from a 'semantic lexicon' consisting of meaning-constructors paired with f-structure information. This produces a pairing of logical forms with f-structures, and full linguistic structures are produced by combining these with conventional c-structure-f-structure pairs whose f-structures are sufficiently similar. Something of this nature seems to be a conceptual necessity in order to have a theory of semantic interpretation for OT-LFG, and it might be worth considering for standard LFG as well. The initial collection of lexical items would then be exactly equivalent to a Mininalist numeration, althought the techniques employed in the computation would be quite different.

## 3 Proof-nets and Deductions

So far I have been asserting that the present proof-net based approach is equivalent to the standard deductive ones, but in spite of the fact that this is well-covered in the literature, I think it might be worth discussing briefly here, since the relevant literature is certainly not written for linguists. But, at this point the math/logic level probably goes up a notch, with a rather temporary remission at section 4. So this would be a reasonable stopping point for people without much time or background.

The basic point is that each proof-net represents a class of proofs, which are deemed to be 'essentially the same' proof, in that they differ only in certain details which are seen as 'bureaucratic', such as the order of application of certain rules (similar in spirit to whether the subject or object NP is expanded first, in a classical rewriting-system implementation of phrase-structure rules, so that phrase-structure trees can be seen as a sort of proof-net formalism for context-free grammar derivations[17]).

They were originally developed for proofs in a superfically rather user-unfriendly proof-format called the 'Gentzen sequent calculus', and indeed for an even more arcane version of that, the one-sided sequent calculus.[18] However for the very limited logic we're

[17]Observation by Jon Cohen, 'That Logic Blog', `http://thatlogicblog.blogspot.com/2006_04_01_archive.html` (April 13, 2006, 'Topologic').

[18]See Crouch and van Genabith (2000) and Moot (2002) for useful discussion. The biggest problem with the one-sided sequent calculus for linear logic is that it depends on the intuitively opaque 'par' connective (multiplicative disjunction), which obeys straightforward logical rules, but has no easily

using here, it's easy to connect them to the much more intuitive Natural Deduction format, used in much recent glue work such as Asudeh (2004, 2005).

Natural deduction for implication logic consists of a general framework for constructing deductions as trees (or, alternatively, lists or sequents), together with two rules, governing the 'introduction' and 'elimination' of the implication symbol, here '$\multimap$', since the logic we're using is Linear Logic (more on this shortly).

One tree rule says that you can make a deduction by writing down a postulate, say, $A$, producing a one-step deduction. Then the rule of $\multimap$-elimination says that if you have one deduction ending in $A$ (the minor premise), and another ending in $A \multimap B$ (the major premise), they can be combined as two branches of a tree whose root (conclusion) is $B$. Graphically:[19]

(48) $$\dfrac{\begin{array}{c}\vdots\\ A\end{array} \qquad \begin{array}{c}\vdots\\ A \multimap B\end{array}}{B}\ \multimap \text{elim}$$

The other rule, implication introduction, says that if we have derived a conclusion $B$ from a deduction that has $A$ as an assumption, we can derive the concusion $A \multimap B$ from the assumptions for $B$, with $A$ removed. This is called 'discharging' the assumption $A$, and is standardly represented by putting superscripted brackets around the discharged assumption, repeating the superscript in the notation for the introduction-rule:

(49) $$\dfrac{\begin{array}{c}[A]^i\\ \vdots\\ B\end{array}}{A \multimap B}\ \multimap \text{intro}^i$$

Note that the other assumptions involved in the production of $B$ are omitted in this somewhat informal notation.

Now for Linearity in the logic. This can be seen as a matter of how one handles the issue of 'types' versus 'tokens' of the assumptions. Suppose we have the assumption 'Bush is a moron', and (actually) using it and perhaps others can deduce either, 'The USA is in trouble', or 'Europe is toast'. In Classical Logic, it is possible to go on to conclude 'The USA is in trouble and Europe is toast' (i.e. both conclusions at once), but in Linear Logic you can't, since, having been used in to produce one part of the conclusion, the Bush assumption isn't available for use in the other (we're assuming it's needed for both; linear logic also obeys relevance, in that every assumption of an argument must be used in the production of the conclusion). Indeed, to derive the combined conclusion, we need two copies of the Bush assumption, one of which must be

understandable intuitive interpretation.

[19]Most presentations of this rule would use one of the standard symbols for ordinary implication, such as $\rightarrow$ or $\supset$, but since we're basically doing linear logic, we'll use the linear one.

dropped (along perhaps with others) if we only want to derive one of the components. This sensitivity to tokens as opposed to types of assumptions strikes many people as being extremely bizarre when they first hear about it, but it makes good sense once one notices that someone interested in the structure of reasoning and arguments might well want to know how many times a given assumption was used to derive a conclusion, which clearly amounts to tracking tokens rather than types of assumptions.

And regardless of its motivations, this idea fits very nicely into the tree-format for a deduction, since the assumption-tokens used will appear at the leaves of the tree, as many times as an assumption is actually used. The constraints of linear logic then amount to the requirement that the deduction be a proper tree, both connected, and with no 'mergers' of the branches (each assumption appearing independently at the end of a branch).

However, in glue deductions,[20] we don't just use deductions, but 'labelled deductions', in which the propositions are associated with a symbolic structure called a 'label', which for glue is some sort of lambda-term representing the meaning. The deduction rules operate on the labels as follows:

(50)

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

This includes the proviso that if an assumption is going to be discharged, its label must be a variable. Asudeh (2004) provides numerous examples of labelled deductions (in a system slightly more complex that what we have here, so far). The original motivations for labelling deductions included such facts as (a) the lambda-term labels provide a convenient record of the essential features of the deduction that produces the final term (b) the propositional component tracks the types of the terms, if we want to operate in the typed lambda-calculus (the 'formulas as types idea'). Independently of the original motivations, deduction-labelling can be and often is used in a fairly freewheeling way to make deductions produce results that are useful for various purposes.

And so the next step is to show how to use a proof-net to construct a natural deduction proof of the conclusion (final positive formula) of the net. What I'll show here is based on Perrier's (1999) method of 'maximal labelling' of proof-nets, itself derived from de Groote's (1999) algebraic formulation of the correctness criterion, which Perrier calls 'minimal labelling' (since it involves a minimal amount of labelling to check correctness, whereas maximal labelling gives as much information as can be collected on a trip through the net).

All these labelling-schemes are based on the idea of assigning values to the nodes of the proof-net, that is, the subformula-tokens, in a manner determined by polarity

[20] In the 'new glue' format of Dalrymple et al. (1999); in the earlier 'old glue' format, things were different.

assignments and the net-structure. What this one will do is assign to each node a natural deduction proof, produced by applying one of the rules to the proofs associated with other nodes in the net. The input to the rules is a proof-net constructed from the meaning-constructors, where the right-hand (glue-side) is construed as a logical formula, the left-hand (meaning-side) as its label. The rules are as follows:[21]

(51) a. To an entire constructor, assign itself as a one-step deduction.

b. If $A$ is connected to $B$ by an axiom-link, assign to $B$ whatever proof gets assigned to $A$.

c. If a negative implication is assigned a proof ending in $f : A \multimap B$, and its (positive) antecedent is assigned a proof ending in $a : A$, assign to its (negative) consequent the proof derived from these two by $\multimap$ elim, whose last line is $f(a) : B$.

d. To a negative antecedent $A$ of a (positive) implication, assign the one-step deduction $x : A$, where $x$ is a variable not previously employed in the calculation (this is an assumption which will have to be discharged).

e. If the (negative) antecedent of a positive implication is assigned $x : A$, and its (positive) consequent is assigned a proof ending in $b : B$, this proof must have $x : A$ as an assumption, and then assign to the implication the proof whose last line is $\lambda x.b : A \multimap B$, which is produced from these by discharging the assumption $x : A$ with $\multimap$ elim.

From the first three rules, it shouldn't be too difficult to derive this as the meaning-deduction for *Ernie yawned*:

(52)

$$\frac{\mathit{Ernie} : g_e \qquad \mathit{Yawned} : g_e \multimap f_p}{\mathit{Yawned}(\mathit{Ernie}) : f_p} \multimap \text{elim}$$

And likewise for slightly more complex examples such as *Bert likes Ernie* (note that we're here taking the f-structure+semantic type information on the glue-side as a kind of composite proposition-letter whose internal structure we're not interested in).

The last two rules are used to derive lambda-abstracts such as are used in the analysis of wide-scope reading of quantifiers. The deduction corresponding to the wide-scope reading of *somebody seems to yawn* is for example ($\multimap$ elim steps left unlabelled due to obviousness):

[21] Readers who already know something about the subject might notice that the construction of entire proof-trees isn't actually necessary, since this is encoded in the form of the lambda-term that is the label of the last step of the deduction.

(53)

$$\dfrac{\dfrac{\dfrac{\dfrac{[x : g_e]^1 \quad \mathit{Yawn} : g_e \multimap h_p}{\mathit{Yawn}(x) : h_p} \quad \mathit{Seem} : h_p \multimap f_p}{\mathit{Seem}(\mathit{Yawn}(x)) : f_p}}{\lambda x.\mathit{Seem}(\mathit{Yawn}(x)) : g_e \multimap f_p}\multimap \text{intro}^1 \quad \mathit{Everybody} : (g_e \multimap f_p) \multimap f_p}{\mathit{Everybody}(\lambda x.\mathit{Seem}(\mathit{Yawn}(x))) : f_p}$$

Proofs can be seen to have a version of $\eta$-conversion, as follows. Suppose we want to do a deductive assembly of *somebody yawned*. The obvious thing to do is:

(54)

$$\dfrac{\mathit{Somebody} : (g_e \multimap f_p) \multimap f_p \quad \mathit{Yawned} : g_e \multimap f_p}{\mathit{Somebody}(\mathit{Yawned}) : f_p}$$

This is what would be produced by the corresponding lopped proof-net. The unlopped one would produce this somewhat longer derivation:

(55)

$$\dfrac{\mathit{Somebody} : (g_e \multimap f_p) \multimap f_p \quad \dfrac{\dfrac{[x : g_e]^1 \quad \mathit{Yawned} : g_e \multimap f_p}{\mathit{Yawned}(x) : f_p}}{\lambda x.\mathit{Yawned}(x) : g_e \multimap f_p}\multimap \text{elim}^1}{\mathit{Somebody}(\lambda x.\mathit{Yawned}(x)) : f_p}$$

The right branch here is an example of a ‘detour’, a sequence of steps that doesn’t produce anything that we don’t already have, from the propositional point of view the statement $g_e \multimap f_p$, from the point of view of the lambda-labels, *Yawned* or its $\eta$-equivalent $\lambda x.\mathit{Yawned}(x)$. Identifying and eliminating such detours through ‘proof-normalization’ is a major concern of Proof Theory.[22]

There’s another kind of detour in proofs, which corresponds to the most important lambda-calculus rule, $\beta$-reduction. This occurs when we follow an abstraction/$\multimap$ intro with an application/$\multimap$ elim:

(56)

$$\dfrac{\dfrac{\begin{matrix}[x : A]^1 \\ \vdots\ \alpha \\ b : B\end{matrix}}{\lambda x.b : A \multimap B}\multimap \text{intro}^1 \quad a : A}{(\lambda x.b)(a) : B}$$

[22]See Restall (in preparation) for an introduction.

From the propositional point of view, it should be evident that we can get the same result from the undischarged assumptions ($A$ and whatever else is involved in the deduction of $B$) with the much simpler (57), where the notation $Z[a/x]$ means '$Z$, with free instances of $x$ replaced by $a$':

(57)
$$\begin{array}{c} a : A \\ \vdots\ \alpha[a/x] \\ b[a/x] : B \end{array}$$

Observe that the term assigned to the conclusion is exactly what $\beta$-reduction specifies to be the $\beta$-reduced form of $(\lambda x.b)(a)$.

The proofs corresonding to our proof-nets turn out to always be fully $\beta$-reduced: the terms they produce/correspond to never provide an opportunity to apply $\beta$-reduction. This would be a big problem were it not for the fact that every lambda-term has a fully reduced '$\beta$-normal form', to which no further $\beta$-reduction can apply. This is the lambda-calculus correspondent of the fact that $\beta$-detours can always be eliminated from proofs. As a result, the correspondence between our proof-nets and Natural Deduction proofs is one-to-one but not onto: to each proof-net there corresponds a unique Natural Deduction, but not vice-versa. A one-to-one correspondence can be obtained by adding 'cut links' to proof-nets; these are used in Categorial Grammar to express lexical decomposition (de Groote and Retoré 1996), but we won't look into that here.

Another significant fact lies in the relationship between $\eta$ and $\beta$-expansion. Using $\beta$-conversion, there is no limit to the complexity of the detouring proofs/formulas that can be manufactured from a $\beta$-normal starting point (note that we're omitting the types here, to reduce clutter):

(58)
$$\begin{array}{c} f(a) \\ (\lambda x.f(x))(a) \\ (\lambda y.(\lambda x.f(x))(y))(a) \\ \vdots \end{array}$$

You can also do this with $\eta$-conversion, but after the first step you get something that isn't $\beta$-normal:

(59)
$$\begin{array}{c} f \\ \lambda x.f(x) \\ \lambda y.(\lambda x.f(x))(y) \end{array}$$

So if $\beta$-conversion is available, $\eta$-conversion isn't needed for very much, in fact, it can be limited to cases where the inner formula isn't itself an abstraction. This is why

$\eta$-conversion isn't seen as anywhere near as important as $\beta$-conversion, and people mostly don't bother to formalize or worry about lopping.

A final point is that the full construction of proofs, as we have implemented it, is actually unnecessary. The structure of the proof can in fact be recovered from the label of its last step (that's what the labels were originally invented for). For discussion of this, see for example Girard et al. (1989) (the earlier chapters of which are in fact quite accessible). Perrier's maximal labelling is just the assignment of these lambda terms only to the proof-net nodes, without the redundant construction of the corresponding conventional proof-tree. It can also be noted that the dynamic graph introduced earlier indicates the order in which values must be assigned for these calculations to work, and the correctness criterion is the requirement that must be satisfied in order for the procedure to assign an appropriate value to every node in the net.

So far, we have presented proof-nets as exact equivalents to $\beta$-reduced deductions, but there is one area where there is a slight technical divergence, which is the scope-ambiguity of quantifiers. The standard LFG treatment involves linear-universal quantification over f-structure locations, so that the meaning-constructor for *Everybody* would be (omitting the 'semantic projection'):

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

Kokkonidis (to appear) shows how this can be construed as first-order quantification, and discusses various issues related to its history in glue. Proof-net based approaches to glue, mine as well as the original Fry (1999), don't handle this quantification explicitly, but in essence assume that things will work out if we omit the quantifier, and treat the variable it binds as a Prolog variable that unifies with whatever it is combined with by axiom-linking, so that if $H_p$ is linked to $f_p$, all instances of $H$ will specialize to $f_p$ (only locations can be variables, not semantic types). It seems obvious that this should work, but it might be good if it was looked at more explicitly from the point of view of proof-net theory, in particular, proof-net constructions for linear universal quantification.

So as long as our glue logic is purely implicational (as is the 'core fragment' of Dalrymple et al. 1999), we therefore get quite a simple relationships between proof-nets and the corresponding proofs. If 'tensors' are introduced, things get somewhat more complicated from a technical point of view, as we shall see soon, but the essential point remains valid.

Before moving on, I'd like to point out some further connections between glue semantics and work over the last several decades in mathematics and logic.[23] We can think of the initial collection of meaning-constructors as being a collection of formulas in fully distinct proposition-letters, from which, of course, no conclusions follow. Our task is to find a substitution-instance of this collection from which something does follow,a task very similar to applying Meredith's rule of Condensed Detachment. A correct proof-net is furthermore equivalent to a 'minimal theorem' of the logic being used (Intuitionistic

[23]Thanks to Bob Meyer for pointing me in the direction of some of this material.

Implicational Linear Logic, aka BCI, so far). That is, a theorem that isn't a substitution instance of any other theorem. For the linking of two literals in the proof-net is basically the same thing as replacing two formerly distinct proposition letters with one, resulting in a sequent with the 'two-property', that each proposition-letter appear only twice, once in a negative and once in a positive position. The two-property appears to have first been investigated by Jaśkowski (1963),[24] and formulas and sequents possessing it have the interesting property of tending to have unique proofs, up to the usual normalizations.[25] Finding such a minimal theorem is also the same thing as finding an 'allowable graph' in the sense of Kelly and Mac Lane (1971) (if we combine all the left-hand members of the sequent into a single tensor-product). But this linking isn't done completely freely, but subject to certain constraints, as expressed by the type and f-structure location information. This can be construed as a requirement that we need a sequent that sits over the obvious f-structure and semantic type sequents in the substitution-instance lattice. So for *Somebody seems to yawn*, the three relevant sequents would be:

(61)
$$\begin{array}{cccc}
(x \multimap w) \multimap z & u \multimap v & y \multimap q & \vdash r \\
(g \multimap X) \multimap X & h \multimap f & g \multimap h & \vdash f \\
(e \multimap p) \multimap p & p \multimap p & e \multimap p & \vdash p \\
\textit{Somebody} & \textit{Seems} & \textit{Yawn} &
\end{array}$$

A solution is a substition-instance of the first line that is a minimal theorem of IILL/BCI (and therefore has the two-property), that has the lower two lines as substitution instances. So the two that there are could be represented as:

(62)
$$\begin{array}{cccc}
(x \multimap w) \multimap z & u \multimap w & x \multimap u & \vdash z \\
(x \multimap w) \multimap z & z \multimap v & x \multimap w & \vdash v \\
\textit{Somebody} & \textit{Seems} & \textit{Yawn} &
\end{array}$$

The graphic representation with the links is clearly a lot easier to deal with intuitively, but the purely propositional representation might have uses as well.

In spite of their connections to earlier work, it is clear that it is the simple and efficient techniques for assessing the correctness of proof-nets developed in Linear Logic that makes them a suitable candidate for use in linguistics. It remains to be seen whether anything linguistically useful will follow from the further involvements of the ideas that have been briefly mentioned above.

[24]Which I haven't managed to look at yet, because somebody seems to have lost or stolen the library's copy of the journal volume.

[25]For example, they don't get additional proofs when the constraints of Linearity are removed from the logic (e.g. Babaev and Solov'ev (1982), Aoto (1999)).

## 4 Tensors and Anaphora

For some initial motivation for tensors, consider the situation of somebody who is unconvinced by Marantz's (1984) arguments that multi-place predicates are curried, and instead prefers the more usual treatment found in introductory logic, where the subject and object of a transitive verbs are in effect packed into a 'complex subject' in the form of an ordered pair, to which the verb then applies as a predicate. In type-theory, such an analysis can be implemented by introducing an additional type-constructor, 'product', or 'pairing', usually symbolized with '$\times$', so that if $a$ is a type and $b$ is a type, then $a \times b$ is the type of pairs whose first component is $a$ and whose second is $b$. We can then use a type such as $e \times e \rightarrow p$ (assuming the convention that $\times$ binds tighter than $\rightarrow$) for transitive verbs, rather than $e \rightarrow e \rightarrow p$ (if we're using a formal semantics based on set theory,[26] it will follow that these two types do essentially the same job, although one might claim to have empirical linguistic or other kinds of reasons for preferring one for the other).

To use such an analysis in glue, we need to use the 'tensor' connective of linear logic (multiplicative conjunction), usually symbolized as '$\otimes$'.[27] A meaning-constructor for a transitive verb using the tensor might be (assuming the convention that $\otimes$ binds tighter than $\multimap$ ):

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

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

We can extend proof-nets to include tensors by adding tensor nodes in addition to the implication nodes we already have, with the polarity-assignent convention that a tensor and its components all get the same polarity. For positive tensors, the appropriate orientation in the tree-format is tensor over components (and the dynamic graph links are from each of the components to the tensor). So the tree-representation of (63) would then be (f-structure information omitted):

(64)

$$(p)^-$$

$$(e \otimes e \multimap p)^- \qquad (e \otimes e)^+$$

*Kill*

$$(e)^+ \qquad (e)^+$$

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

However we'll require some additional machinery if we want to do work in the meaning-side, for example lexical decomposition. Suppose we want to express the idea that *kill*

[26] Or indeed any 'Cartesian Closed Category'

[27] Perhaps because the tensor product construction in linear algebra is an important example obeying the essential rules governing multiplicative conjunction (Troelstra (1992), ch 9; MacLane (1971:157)).

means *cause to die* (or some more empirically tenable complexification of this position). Our *Kill*-meaning here applies to a pair of type $e \times e$, but for the decomposition we need to dissect the pair into its two components, and feed each to a different predicate. This can be done with the projections $\pi_1^{e,e}$ and $\pi_2^{e,e}$, which take extract the first and second members, respectively, of something of type $e \times e$ (but this requires going beyond linear logic). With the types of the variables notated as superscripts where they are bound, the decomposed meaning-side might be:

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

(In real life, people wouldn't bother to notate all of the types explicitly, as long as they could be determined from context.)

The use of input (positive) tensors for arguments has been a persistent minority option in glue (especially in the 'old glue' notation, where the projections aren't needed to build the meanings), although I think it could be weakly challenged on the basis of requiring more arbitrary decisions than the curried alternative. Marantz's arguments provide some rationale for deciding what arguments should be 'composed first' (or, more plausibly in a framework such as LFG, which tries to keep operations as order-free as possible, structurally closer to the predicate in the underlying representation), but there is no comparable way to argue about which component of the tensor should be first, or how the arguments of 3-or-more-place predicates should be packaged.

A more interesting question arises when we consider the fact that if we have input tensors, we might also expect to have output (negative polarity) tensors. Asudeh (2004) discusses various uses to which these might be put, but the only worked out and persistently popular example is anaphora, although there are alternatives, as discussed in Kokkonidis (2005) and Lev (2006b).

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

On the other hand, (a) their absence seems to actually make the deeper math more difficult (Mints 1981) (b) the counterparts to the tensors in systems very closely related to glue (such as the products in closed cartesian categories, and the dot-composites in categorial grammar) play a conceptually central role in those systems, so that their absence from glue might be seen as peculiar. So, without endorsing any specific position on the desireability of including tensors, we present here a discussion of their use in anaphora.

A common feature of all of the analyses of anaphora is that in order to taken as analyses of anaphora specifically, it's necessary to consider the internal structure of

the meaning-side, not just the glue-side. We'll start by illustrating this point by a tensor-free analysis of anaphora discussed by Lev (2006a), whose basic mechanism in fact seems well-motivated for reciprocals (Lev 2006c), although more problematic for reflexive and ordinary pronouns. Then we'll consider a popular analysis with tensors, first proposed in Dalrymple et al. (1997), and later taken up by Asudeh (2004).

## 4.1 Anaphora without Tensors

All glue analysis of anaphora make use of an ANTecedent attribute of the anaphoric element, discussed in detail in Dalrymple (1993). The value of this attribute is the f-structure of some other element of the syntactic structure of the sentence, which is supposed to intuitively be the 'supplier' of the meaning of the anaphoric element. So consider the f-structure of a sentence such as *John washed himself*, with the ANT grammatical function included:

(66)
$$f{:}\begin{bmatrix} \text{SUBJ} & g{:}\begin{bmatrix}\text{PRED} & \text{`John'}\end{bmatrix} \\ \text{TENSE} & \text{PAST} \\ \text{PRED} & \text{`Wash(SUBJ, OBJ)'} \\ \text{OBJ} & h{:}\begin{bmatrix}\text{PRED} & \text{`Pro'} \\ \text{ANT} & [\ \ ]\end{bmatrix} \end{bmatrix}$$

If the instantiated meaning-constructors were just:

(67)
$$\begin{array}{rcl} \textit{John} & : & g_e \\ \textit{Himself} & : & g_e \multimap h_e \\ \textit{Washed} & : & h_e \multimap g_e \multimap f_e \end{array}$$

with *Himself* merely piping the content of the subject to the object position, things wouldn't work out, because the verb's demand for an input from the subject wouldn't get satisfied.

This problem can be solved by a trick, which is to build a constructor that accepts a two-place predicate, and produces a one-place predicate (basically a diagonalization operator). The type of a 2-place predicate will be $e{\rightarrow}e{\rightarrow}p$, and that of a 1-place predicate $e{\rightarrow}p$, so that the semantic type information of the constructor will be $(e \multimap e \multimap p) \multimap e \multimap p$. What about the f-structure information? The two-place predicate will take input located at the pronoun and antecedent f-structures, so these will be the f-structure locations of the type $e$ 'sub-arguments' of the first argument of the constructor, and the antecedent is the ulimate supplier of the content that's being piped around, so putting everything together we get:

(68) $\textit{Himself} : ((\uparrow \text{ANT})_e \multimap \uparrow_e \multimap H_p) \multimap (\uparrow \text{ANT})_e \multimap H_p$

With this new constructors, we get a correct proof-net (writing the constructors vertically, since they won't fit on one line):

(69)

$$\begin{aligned}
\textit{Johns} &: g_e \\
\textit{Himself} &: (h_e \multimap g_e \multimap f_p) \multimap g_e \multimap f_p \\
\textit{Washed} &: h_e \multimap g_e \multimap f_p
\end{aligned}$$

You can check that this satisifes the correctness criterion, but what does it mean?

Redoing it as a tree-format logical form isn't very illuminating, except that it does bring out the idea of the reflexive as being a sort-of-quantifier-like operator:

(70)

$f_p$
$g_e \rightarrow f_p$   $g_e$ *John*
$(h_e \rightarrow g_e \rightarrow f_p) \rightarrow g_e \rightarrow f_p$ *Himself*   $\lambda$
$h_e$   $\lambda$
$g_e$   $f_p$
$g_p \rightarrow f_p$   $g_p$
$h_p \rightarrow g_p \rightarrow f_p$ *Washed*   $h_p$

But to see this as being an analysis of reflexivation, we need a meaning-side which expresses the idea that whatever argument is supplied to the operated-on predicate plus *Himself* is supplied to both arguments of the former. Such as:

(71) $\lambda P^{e \rightarrow e \rightarrow p} \lambda x^e . P(x)(x)$.

If we do the $\beta$-reductions that then become possible for (70), we get the final desired result *Washed*(*John*)(*John*).

An important feature of this treatment is that it requires a combination of two different kinds of logic, the linear logic that does the basic assembly, and something more powerful to produce the actual reading. This is because linear lambda-terms cannot produce the effect of copying the bound variable $x$ in (71) into the two different argument positions where it appears. The nature of the connections between the two logics isn't much discussed in the LFG glue literature, although it is sometimes handled more explicitly in categorial and type-logical grammar. One question that arises is whether it is possible to use proof-nets for the meaning-sides, and resultant $\beta$-reductions; the answer is yes, although I am not sure of the best way to set things up.[28] An advan-

[28] de Groote and Retoré (1996) for example use the standard embedding of (noncommutative) linear logic into intuitionistic logic, although this requires the use of various cumbersome proof-nets constructions such as weakening and promotion boxes.

tage of using a proof-net-like structure rather than a conventional lambda-term is that the annoyances of specifying free vs. bound variables, etc. and 'change of variables' ($\alpha$-conversion) would no longer be required.

So we have a tensor-free analysis works for the case discussed, and various other ones, it has at least two major drawbacks as a treatment of anaphora:

(72) a. It seems to have very dim prospects as a treatment non-bound anaphora

b. It gives rise to certain spurious structural ambiguities, as discussed by Lev (2006a).

For example *Ernie probably likes himself* would be either:

(73) a. $\mathit{Probable}(\mathit{Self}(\mathit{Ernie}, \lambda yx.\mathit{Like}(x, y)))$

b. $\mathit{Self}(\mathit{Ernie}, \lambda yx.\mathit{Probable}(\mathit{Like}(x, y)))$

For reciprocals, similar ambiguities are sometimes genuine (Lev 2006c), but in the case of reflexives, there seems to be no hint of either an intuitive structural or interpretational ambiguity. This might simply be because the two analysis $\beta$-reduce to the same thing, but it's still a point to worry about.[29]

On the other hand, the good features of this analysis are:

(74) a. it fits into the purely implicational fragment of Intuitionistic Linear Logic.

b. it uses a form of constructor that is independently motivated for reciprocals (Lev 2006c).

## 4.2 Anaphora with Tensors

The other analysis we'll look at, using tensors, was formulated in the 'old glue' format by Dalrymple et al. (1997), and used in new glue by Asudeh (2004). The idea here is to use a negative tensor as output of the reflexive constructor, which is then:

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

Suitably instantiated, this can be combined with other constructors into a (sequent-style) proof-net like this:

(76) $g_e \quad g_e \multimap g_e \otimes h_e \quad h_e \multimap g_e \multimap f_p \quad \vdash \quad f_p$

*John* $\quad \lambda x.[x, x]$ $\quad$ *Washed*

*Himself*

[29] And note that, unlike some other spurious ambiguities noticed by Lev, this one can't be suppressed by requiring the reflexive to take its scope at the lowest possible c-structure node.

This is correct, although the correctness criterion has be strengthened to require the dynamic graph (links running from negative tensor to its components) for to be both connected and non-cyclic, rather than one or the other (de Groote 1999).

And an issue arises as to how to read it as a logical form. If we try to apply our previous scheme for reading proof-nets as lambda-terms, a difficulty arises. The constructor for *Washed* renders straightforwardly as:

(77)
$(f_p)^-$
$(g_e{\rightarrow}f_p)^-$ $(g_e)^+$
$(h_e{\rightarrow}g_e{\rightarrow}f_p)^-$ $(h_e)^+$
*Washed*

But assembling *John* and *Himself* yields a configuration that we haven't produced rules for dealing with:

(78)
$(g_e \otimes h_e)^-$
$(g_e \multimap g_e \otimes h_e)^-$ $(g_e)^+$
*Himself*
$(g_e)^-$
*John*

The problem of course being the composite output.

Clearly, we want to run links from the two components of this output to the two inputs of *Washed* (that's what the proof-net does), but the result is not a tree:

(79)
$(f_p)^-$
$(g_e{\rightarrow}f_p)^-$ $(g_e)^+$
$(h_e{\rightarrow}g_e{\rightarrow}f_p)^-$ $(h_e)^+$
*Washed*
$(g_e \otimes h_e)^-$
$(g_e \multimap g_e \otimes h_e)^-$ $(g_e)^+$
*Himself*
$(g_e)^-$
*John*

(We really ought to expand the tensor into tree-form, as a little root-down tree with the full formula as the root and the two components as branches, but we won't bother with this.)

Although (79) isn't a tree, it isn't too hard to come up with a way of reading it as a kind of logical form. The idea is that the axiom-links from the first and second

components of the output tensor are treated as being like first and second projections, so that (79) is read as if it were:

(80) $\textit{Washed}(\pi_1(\textit{Himself}(\textit{John})))(\pi_2(\textit{Himself}(\textit{John})))$

The main difference being that in a linear format, we need to use two copies, rather than two links to a shared substructure.

This idea can be expressed with an extension of Perrier's maximal labelling rules (Perrier himself doesn't extend his system to tensors). The implicational rules, as originally formulated, are (these are just the rules for proof-construction (51), with everything but the construction of the label of the conclusion of each proof removed):

(81) a. To an entire constructor, assign its meaning-side

b. If $A$ is connected to $B$ by an axiom-link, assign to $B$ whatever gets assigned to $A$.

c. If a negative implication is assigned $f$, and its (positive) antecedent is assigned a proof ending in $a$, assign to its (negative) consequent the $f(a)$

d. To a negative antecedent $A$ of a (positive) implication, assign $x$, where $x$ is a variable not previously imployed in the calculation (this is an assumption which will have to be discharged).

e. If the (negative) antecedent of a positive implication is assigned $x$, and its (positive) consequent is assigned $b$, then $b$ must contain $x$, and assign to it the implication $\lambda x.b$.

For tensors, we add the following two rules:

(82) a. If $A \otimes B$ is a negative tensor to which the $z$ has been assigned, assign $\pi_1(z)$ to the first component ($A$), and $\pi_2(z)$ to the second component ($B$).

b. If $A \otimes B$ is a positive tensor where $A$ has been assigned $x$ and $B$ $y$, then assign $[x, y]$ to $A \otimes B$.

These rules will then assign (80) as value to the final output of (79).

This analysis shares with the previous one the property of not being construable as an analysis of any kind of coreference, unless we take into account the internal structure of the meaning-side of the reflexive pronoun constructor. As far as glue itself is concerned, it simply says that the subject and object of the sentence are first and second components of something produced as output by something that applies to the subject (or, more generally, antecedent of the pronoun). The linear logic of glue is not capable of copying the reference of the antecedent so as to supply these copies to different positions; this requires the intervention of an additional and less constrained logic that can manipulate the internals of the meaning-sides.

But this treatment does have the advantage that it doesn't introduce spurious ambiguities with operators such as *probably*. Introducing this adverb leads to this proof-net

(83) $g_e \quad g_e \multimap g_e \otimes h_e \quad h_e \multimap g_e \multimap f_p \quad f_p \multimap f_p \quad \vdash \quad f_p$

*John* $\quad \lambda x.[x, x]$ *Himself* $\quad$ *Washed* $\quad$ *Probably*

which is equivalent to this more conventionally notated logical form:

(84) *Probably*(*Washed*($\pi_1$(*Himself*(*John*)))($\pi_2$(*Himself*(*John*))))

However, these proof-nets don't have such a clear relation to Natural Deduction. Readers of Asudeh (2004) may well have noticed that there, projections aren't used, but rather a **let**-term constructor, so that instead of (80) one would get:

(85) **let** *Himself*(*John*) **be** $x \otimes y$ **in** *Washed*($y$)($x$)

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

(86) **let** $[a, b]$ **be** $x \otimes y$ **in** $c \;\; \Rightarrow_\beta \;\; c[a/x, b/y]$

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

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

But of course glue *per se* doesn't have such access, so as far as glue is concerned, the logical form is the unreduced (85).

The literature appears to provide no guidance as to the choice between these two techniques. The projections are used without comment in the Type-Logical Grammar literature (e.g. Moot 2002), whereas the LFG literature seems to use **let**, with equally little discussion of the alternatives.

One aspect of the choice seems to be that projections fit more naturally to the use of proof-nets, while the **let**-constructors more closely follow the grain of Natural Deduction. That there is a problem with **let** and proof-nets can be seen from the fact in an adverbial example such as (83), we have a single proof-net, but have to choose between two different-looking **let**-terms:

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

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

This difference corresponds to a difference in the possible proofs in a Natural Deduction formulation.

The (labelled) Natural Deduction rules for ⊗ are:

(89) a.

$$\frac{a : A \qquad b : B}{[a, b] : A \otimes B} \otimes\text{intro}$$

(not actually used in Asudeh (2004), but included for symmetry)

b.

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

The proofs produced by these rules will show the same choice as in the pair of proof-terms (88), for the reason that we have to make a choice about when to replace the pair of variables $x, y$ in the argument positions of *Washed*, and this can be done either before or after the application of *Probably*.

At this point it looks as if proof-nets and natural deductions are coming adrift rather badly, but there is a fix at hand, motivated by the fact that if the first argument of **let** is a product itself produced by linear logic, the two formats of (88) will $\beta$-reduce to the same thing, and should therefore be regarded as being the same term. This is achieved by means of a 'commuting conversion', which defines an equivalence relation between terms related as those in (88). See Mackie et al. (1993:317) (conversion 4) for a formulation (as usual, in a much more complex environment than required for glue, since meaning assembly doesn't appear to need the linear logic units).

So, although the **let** constructor participates in the same structural ambiguities as the Natural Deductions, and so corresponds better to them than to proof-nets, this doesn't matter very much, because the two structures are to be equated by a commuting conversion. One might think of proof-nets as having the advantage here, in that commuting conversions aren't required (for this particular application, although there are others where they are still found useful), but on the other hand their coresonding proof-terms are more redundant in presentation, with the duplications, and also require a more complex concept of linear binding.

The reason for this is that if we have bound anaphora to a quantificational NP:

(90) Everybody washed himself

the projection-based logical form will contain two instances of the bound variable, one inside each projection:

(91) $\mathit{Everybody}(\lambda x.\mathit{Washed}(\pi_1(\mathit{Himself}(x)))(\pi_2(\mathit{Himself}(x)))))$

But this duplication of the variable is an artifact of the linear representation: in all of the proof-net formats, there will be a single link from the negative $e$ of the quantifier to the positive of *Himself*.

The tensor analysis is somewhat less hopeless for cross-sentential anaphora than the one of the previous section, but still doesn't seem very promising, due to the fact that in order to get the anaphora in a discourse such as *Somebody/John came in. He sat down*, we'd need to retrospetively restructure the first sentence by inserting a tensor, in order to process the second, as well as insert the second within the scope of the quantifier, in the existential version. It is therefore not a big surprise that there's been considerable work on developing methods related to dynamic logic and discourse representation theory in glue, which involves mechanisms that go beyond glue itself, and so won't be discussed here (see Kokkonidis (2005), Lev (2006a) for recent discussion).

I will however make a general remark. Much work on anaphora in glue, such as Crouch and van Genabith (1999) and Dalrymple (2001), has devoted considerable effort to trying to fit the phenomena of anaphora with the mechanisms of linear logic. But anaphora doesn't display the distinctive properties that linear logic easily accounts for, which is resources that must be used once and once only. When a referent is introduced, it doesn't have to be referred to again, and but it always can be, as many times as desired. There aren't any linguistic formats for NPs that either require or forbid later reference as a matter of syntax, let alone demand a specific number of later references (as opposed to NPs such as *nobody*, to which later reference might not make sense, and is therefore excluded on conceptual grounds). In the DRT-based approach suggested by Kokkonidis, and in slightly modified form by Lev, on the other hand, most of the work of accounting anaphora is done by a different component, a discourse representation theory which functions behind the scenes from the point of view of the grammatically controlled meaning-compositions controlled by glue. This would seem like a fundamentally more promising approach. But the whole approach seems to have a long way to go before it is up to speed on standard issues such as paycheck sentences, antecedent-contained deletion, etc. It's reasonable to hope that the flexibility of glue will provide good solutions to these problems, but they haven't been worked out.

## Bibliography

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

Andrews, A. D. 2004b. Unificational glue. URL: `http://arts.anu.edu.au/linguistics/People/AveryAndrews/Papers`.

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

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

Aoto, T. 1999. Journal of logic, language and information. 8:217–242.

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

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

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

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.

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

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

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

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

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

Dalrymple, M. 2001. *Lexical Functional Grammar.* Academic Press.

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

Dalrymple, M., R. M. Kaplan, J. T. Maxwell, and A. Zaenen (Eds.). 1995. *Formal Issues in Lexical-Functional Grammar.* Stanford CA: The Center for the Study of Language and Information.

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

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

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

de Groote, P., and C. Retoré. 1996. On the semantic reading of proof-nets. In G. G.-J. Kruijff and D. Oehrle (Eds.), *Formal Grammar*, 57–70, FOLLI Prague, August. URL: `citeseer.ist.psu.edu/degroote96semantic.html`.

Fry, J. 1999. Proof nets and negative polarity licensing. In M. Dalrymple (Ed.), *Syntax and Semantics in Lexical Functional Grammar: The Resource-Logic Approach*, 91–116.

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

Hughes, D. 2005. Simple multiplicative proof nets with units. `http://arxiv.org/abs/math.CT/0507003`.

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

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

Kasper, R. 1995. Semantics of recursive modification. URL: `ftp://ling.ohio-state.edu/pub/HPSG/Workshop.Tue.85/Kasper/`. (handout for HPSG Workshop Tuebingen, June 1995).

Kelly, G., and S. Mac Lane. 1971. Coherence in closed categories. *Journal of Pure and Applied Algebra* 1:97–140.

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*. URL: `http://csli-publications.stanford.edu`.

Kokkonidis, M. to appear. First order glue. *Journal of Logic, Language and Information*. URL: .

Kuhn, J. 2001. Resource sensitivity in the syntax-semantics interface and the german split np construction. In W. D. Meurers and T. Kiss (Eds.), *Constraint-Based Approaches to Germanic Syntax*. Stanford CA: CSLI Publications.

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

Lev, I. 2006a. Anaphora in glue semantics. URL: http://www.stanford.edu/~iddolev/papers/gmemo_anaphora.pdf.

Lev, I. 2006b. Introduction to the syntax-semantics interface using glue semantics. URL: http://www.stanford.edu/~iddolev/papers/gmemo_intro.pdf.

Lev, I. 2006c. On the syntax-semantics interface of overt and covert reciprocals. URL: http://www.stanford.edu/~iddolev/papers/ling233b_quants.pdf.

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

MacLane, S. 1971. *Categories for the Working Mathematician*. Springer Verlag.

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

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

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

Morrill, G. 2005. Geometry of language and linguistic circuitry. In C. Casadio, P. J. Scott, and R. Seely (Eds.), *Language and Grammar*, 237–264. CSLI Publications.

Partee, B. H., A. ter Meulen, and R. E. Wall. 1993. *Mathematical Methods in Linguistics*. Kluwer. Corrected second printing.

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

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

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

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