Glue Logic, Projections and Modifiers*

Avery Andrews, ANU, May 17, 2003

2nd draft

In this paper I will show how the 'glue logic' that has been developed to help provide a formal semantics for LFG (Dalrymple 2000) can be used to integrate semantically relevant information that is distributed across multiple projections, thereby permitting the kinds of analysis proposed in Andrews and Manning (1993, 1999) to use conventional LFG projections (Kaplan 1995), rather than the 'restriction projections' proposed by Andrews and Manning. In addition to dispensing with one theoretical innovation by means of a more aggressive use of another, I also hope to make the latter innovation, glue logic, a bit more accessible by using an alternative presentation in terms of 'content flow', based on the the work on proof-net correctness criteria by de Groote (1999) and Perrier (1999). There will therefore be an *ab initio* exposition of glue-logic.

The elimination of restriction projections will however involve a slight change in how glue-logic operates in LFG: the semantic projection will have to be abolished, with semantic content being located directly at both f-structure and the a-structure level proposed by Andrews and Manning. I believe that the content-flow interpretation made it easier to see how to dispense with restriction-projections: if one is thinking of content as being passed between different places in a structure, it then becomes natural to wonder if it can be shifted between different projections, and the answer to this question is clearly 'yes'. Nevertheless, the content-flow interpretation of glue-logic is a matter of presentation rather than empirical substance, and is an independent issue from multiple projections and the elimination of the semantic projection.

The term 'glue logic' refers to the use of a special logic to control the process of 'semantic assembly' whereby the meanings of words (and perhaps in some cases at least, grammatical constructions) are assembled to give a meaning for the whole utterance. A very important point is that glue logic doesn't say anything directly about the meanings, and is therefore not a meaning-specification language, but merely a tool for specifying how the meanings, whatever they are, are put together (although the technique of assembly does carry certain implications about the nature of the meanings).

The lexical and constructional meanings are initially presented as 'meaning constructors', which in the standard interpretation serve as premises for a deduction; if from the premises a conclusion can be derived that meets certain constraints, this conclusion is taken to specify a meaning for the sentence. On the 'content-flow' interpretation presented here, the meaning-constructors can be viewed as 'content processors', and the process of semantic assembly consists of hooking them up subject to certain constraints, such that content for the whole sentence is produced.

The use of glue-logic for semantic assembly is closely related to the conception of semantic types introduced into modern linguistics by Montague and his associates, made out of the basic types e (entity) and t (truth-value) and perhaps others, combined with function-formation (lambda-abstraction), to produced derived types for intransi-

*I am indebted to Ash Asudeh and Mary Dalrymple for discussion some of the issues considered, all errors are of course due to me.

tive and transitive verbs, quantifiers and various kinds of modifiers.[1]

1 Proper Names and Predicates

Proper names are for example seen as producing e-type content, while intransitive verbs consume e-content and produce t-content, and so are functions from e to t, symbolized $e \rightarrow t$. To get the content of a sentence produced by combining a proper name and an intransitive verb, we plug the name's e-output into the verb's e-input, and collect the resulting t-output. Transitive verbs are treated as functions from e to intransitive verb meanings, that is, of type $e \rightarrow (e \rightarrow t)$, which we can write as $e \rightarrow e \rightarrow t$ by treating $\rightarrow$ as right-associative. In terms of content-processing, they take in two streams of e-content to produce one stream of t-content.[2]

To start converting this imagery into an actual account, one thing we need to do is allow for syntactic structure to control semantic assembly; since this application of glue logic is to LFG, we do this by supposing that the inputs and outputs are tied to specific locations in the f-structure (actually, quantifiers turn out to be a partial exception to this). A proper name for example produces its output at the f-structure corresonding to the N node that the name is introduced in, while an intransitive verb collects its e-input at its f-structure subject, and produces its t-output at the f-structure corresponding to the V it's introduced under. I will symbolize this locational information by writing f-structure-designators followed by '$\sim$' in front of the semantic type, e.g. '$f \sim e$' for 'e-content at f-structure f'. When writing the meaning-constructors, we also use the linear implication symbol '$\multimap$' instead of ' $\rightarrow$ ', so an intransitive verb 'signature' might be '$g \sim e \multimap f \sim t$'. Due to the origin of this scheme in logic, this kind of combination is called an 'implication'; the material to the left of the $\multimap$ the 'antecedent', to the right, the 'consequent'. The actual form of a meaning-constructor will then consist of:

(1) a. a 'left hand side' or 'meaning term' symbolizing the content.

b. a 'right hand side', 'glue term' or 'signature' specifying the information needed to control semantic composition.

Before proceeding, I'll mention two nonstandard features of this presentation. First, content is normally located at a 'semantic projection' that comes off f-structure, rather than at f-structure itself. I see no clear motivation in existing work for the semantic projection, and it is incompatible with the use of multiple projections, to be introduced later, so I omit it here. Furthermore it is common to omit the type information, but since it is sometimes important, I will almost always include it.

Since the syntactic theory we're working in is LFG, meaning-constructors are specified in the lexicon using the $\uparrow$-arrow metavariable to refer to 'the f-structure corresponding to the node I am inserted under' (this will be modified later, to accomodate

[1] We'll follow tradition here and ignore matters of intensionality.

[2] In the 'Intuitionistic Implicational' fragment of Linear Logic, which we'll be working in here, it is a basic constraint that there be only one stream of output content. To deal with anaphora, as analysed in Dalrymple (2000) and references cited there, we need more than one, and require the somewhat broader 'Multiplicative' fragment. A content-flow interpretation has not yet been developed for the multiplicative fragment, although de Groote's correctness criterion, which this interpretation is based on, is applicable.

multiple projections), plus the full resources of the f-structure designator notation to specify f-structures. So the meaning-constructors for a proper name and intransitive verb might look like this:

(2) *uther* : $\uparrow \sim e$
sleep : $(\uparrow \text{SUBJ}) \sim e \multimap \uparrow \sim t$

If these are used in a syntactic structure, we replace the f-structure designators with variables tagging the f-structures, getting a set of instantiated constructors such as:

(3) *uther* : $g \sim e$
sleep : $g \sim e \multimap f \sim t$

The latter is what semantic assembly works on.

To formalize assembly, we first need a notion of 'polarity' to characterize the inputs and outputs. There are unfortunately two opposite conventions for this in circulation; we'll use the one used by Gupta and Lamping (1998), since it seems more intuitive for the content-processing interpretation.[3]

The glue terms of meaning-constructors are all assumed to have 'positive polarity', meaning that from the perspective of hooking up the network, they are content-emitters.[4] Then, for implications, we have the following polarity-assignment rules:

(4) a. The antecedent of a positive implication is negative.

b. The consequent of a positive implication is positive.

Applying these principles to some constructors for a transitive verb sentence, we get polarity assignments like this (writing the polarity of an implication under its $\multimap$ symbol):

(5) *art* : $g \sim e$
$+$

gwen : $h \sim e$
$+$

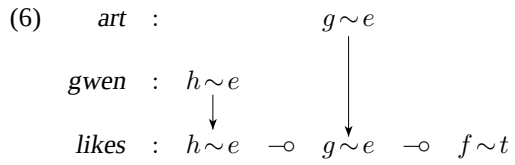
likes : $h \sim e \quad \multimap \quad (g \sim e \quad \multimap \quad f \sim t)$
$- \qquad + \qquad - \qquad + \qquad +$

To help with understanding the polarity pattern with the *likes* constructor, I've written in its rightmost parentheses, even though these will normally be omitted in accord with right associativity.

Now to do the assembly, we connect positive and negative 'literals' (the basic terms, consisting of f-structure label and semantic type, connected with '$\sim$') with arrows going from the positives to the negatives, until there is one positive left over, which should be located at the f-structure for the whole sentence. Doing this to (5) yields (6):

[3]However, unfortunately, de Groote and Perrier, whose work this is an application of, use the opposite convention.

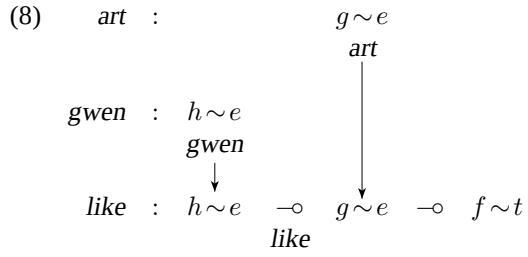
[4]de Groote and Perrier's opposite interpretation makes sense looking at the network from outside, where the constructors take meaning up from the environment.

(6)


To get a representation for the resulting content, we procede as follows. First, we assign content to each entire glue-term as follows:

(7) The content of the glue-side (entire glue term) of a meaning-constructor is its meaning-side.

Using this rule, (6) becomes (8):

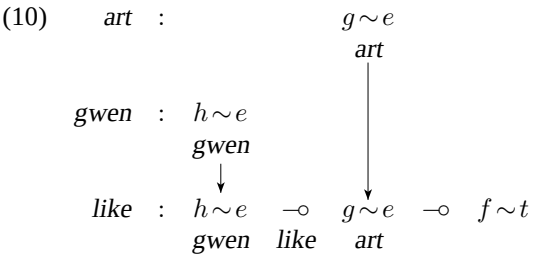
(8)


Then we specify rules whereby content flows along arrows, and also is assigned to subterms of the glue terms, starting with these:[5]

(9) a. The content of a positive literal propagates to a negative that it is linked to.

b. If the content of an implication is a, and of its antecedent b, then the content of its consequent is $a(b)$ (the value of the function a applied to b as an argument).

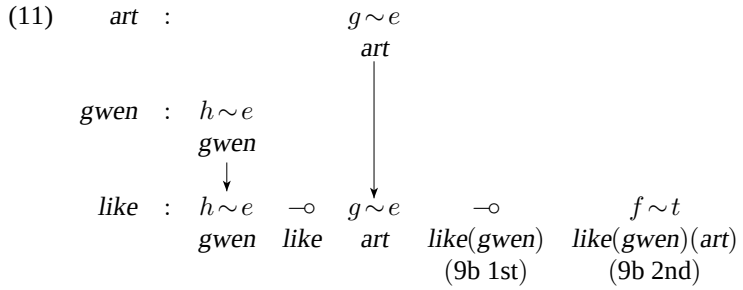
Now rule (a) get us (10):

(10)


Then 2 applications of (b) produce the final result:

[5]Perrier (1999) showed how to use content-propagation rules to produce logical forms.

(11)



This produces a Montague-style representation, assuming that the verb applies first to the semantic value of the object (c.f. Marantz 1984); if we want to use some other notation to represent lexical content, we can use lambdas, so the meaning-side for the 'like' constructor might for example be represented as $\lambda y.\lambda x.\mathit{like}(x, y)$, which would yield a final result of $\mathit{like}(\mathit{art}, \mathit{gwen})$.

Unfortunately, just hooking up positive to negative literals isn't sufficient, as we can see by considering the constructors for a sentence involving a sentence adverb such as *obviously*. This adverb seems to map a propositional meaning, located at the f-structure of the clause it appears in, onto another propositional meaning, located at the same place. So we might get something like this:

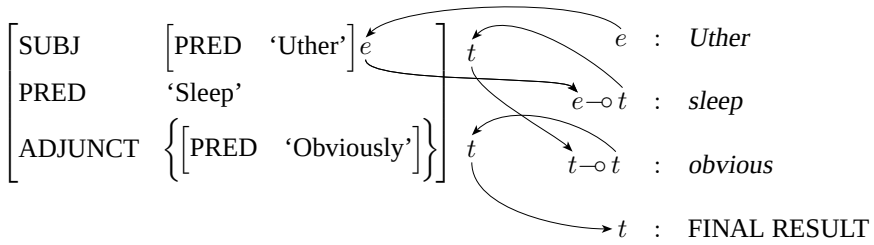
(12)
$$\begin{array}{rcl} \mathit{uther} & : & g\sim e \\ \mathit{sleep} & : & g\sim e \multimap f\sim t \\ \mathit{obvious} & : & f\sim t \multimap f\sim t \end{array}$$

You can see that alongside of the sensible hookup that delivers $\mathit{obvious}(\mathit{sleep}(\mathit{uther}))$ as the content of the sentence, there's a perverse hookup where the output of *obvious* is fed into its own input, and the output of *sleep* is the unconnected 'final' output. Fortulately, we can use the content-propagation rules to formulate a criterion for excluding the bad hookup (de Groote 1999). There are actually several ways that this can be done; the simplest is:

(13) The content of every meaning-constructor must appear in the final result

This excludes the perverse hookup and but allows the proper one.

The treatment of *obviously* illustrates an important feature of the approach, which is that there is no limit on the number of output-input connections that can be made at a given location in f-structure: here the entire f-structure has two, getting input from the verb and sending output to *obviously*, and getting input from *obviously* and sending the final output. It might be useful to illustrate this with a diagram which explicitly indicates how the f-structures are serving as connection points for the inputs and outputs:

(14)


One can think of the meaning-constructors as having input and output 'ports' located at the various f-structure position; assembly consists of connecting inputs to outputs with hoses that are rather short, and therefore unable to connect inputs and outputs located at different f-structures. The f-structural origins and destinations of the arrows are fixed by the meaning-constructors, but how they are paired up at a given f-structure is an option of assembly (however quantifiers get to choose source and destination, as we shall see below).

The existence of multiple connections at one f-structure corresponds to the 'updating' of meaning-assignments in the standard presentation, whereby an assignent such as $s \rightsquigarrow$ *sleep*(*uther*) gets replaced in the course of the deduction with $s \rightsquigarrow$ *obvious*(*sleep*(*uther*)) (c.f. Crouch and van Genabith 1999:120). I think the multiple connections view is 'more in the spirit' of LFG, since it uses the addition of connections rather that the updating of assignments, and so is more conceptually monotonic.[6]

This general approach can clearly extended to complex sentences, including those with functional control. First consider a *seem* f-structure:

(15)
$$h\text{:}\begin{bmatrix} \text{SUBJ} & g\text{:}\begin{bmatrix}\text{PRED} & \text{'Uther'}\end{bmatrix} \\ \text{PRED} & \text{'Seem(XCOMP)'} \\ \text{XCOMP} & f\text{:}\begin{bmatrix}\text{SUBJ} & g\text{:}\begin{bmatrix}\ \end{bmatrix} \\ \text{PRED} & \text{'Sleep'}\end{bmatrix}\end{bmatrix}$$

The fact that *Uther*'s f-structure is doubly connected into the f-structure of the whole sentence doesn't present any problem for glue-logic, since the uninstantiated constructors can simply be:

(16) *uther* : $\uparrow \sim e$
sleep : $(\uparrow \text{SUBJ}) \sim e \multimap \uparrow \sim t$
seem : $(\uparrow \text{XCOMP}) \sim t \multimap \uparrow \sim t$

which, for appropriate c- and f-structures, will instantiate out to:

(17) *uther* : $g \sim e$
sleep : $g \sim e \multimap f \sim t$
seem : $f \sim t \multimap h \sim t$

[6] But the assignment-updating view is also monotonic in the sense that the new assigned value always contains the material provided in the old one.

These will link up smoothly, not caring about how many ways *Uther* is linked into the structure.

try turns out to require a bit more work, including more rules for value-assignment. Unlike *seem*, *try* assigns a semantic role to its subject, which is of type e; there is furthermore an issue as to what the second argument is: is it semantically a proposition (type t, as implicit in the traditional transformational analysis), or an 'action' (type $e \rightarrow t$, as implicit in the 'VP complement' analysis)? There is furthermore the syntactic issue of whether it involves functional or anaphoric control. If we treat it as involving functional control, and wish to remain within the implicational fragment, the only available analysis is the one proposed in Asudeh (**?**, **?**), where from the point of view of semantic composition, the complement is a 'semantic VP' (it is however possible to have the underlying semantic primitive relate an entity and a proposition, by means of internal application of the meaning of the VP to that of its understood subject). The VP complement is of semantic type $e \rightarrow t$, and the other argument, the tryer, if of course of type e; a workable uninstantiated constructor is:

(18) *try* : $((\uparrow\text{SUBJ}) \sim e \multimap (\uparrow\text{XCOMP}) \sim t) \multimap (\uparrow\text{SUBJ}) \sim e \multimap \uparrow \sim t$

With an f-structure such as (19) below, this would result in the instantiated constructor set (20):

(19) h: $\begin{bmatrix} \text{SUBJ} & g\text{:} \begin{bmatrix} \text{PRED} & \text{'Uther'} \end{bmatrix} \\ \text{PRED} & \text{'Try(SUBJ, XCOMP)'} \\ \text{XCOMP} & f\text{:} \begin{bmatrix} \text{SUBJ} & g\text{:} \begin{bmatrix} \ \end{bmatrix} \\ \text{PRED} & \text{'Sleep'} \end{bmatrix} \end{bmatrix}$

(20) *uther* : $g \sim e$
sleep : $g \sim e \multimap f \sim t$
try : $(g \sim e \multimap f \sim t) \multimap g \sim e \multimap h \sim t$

What we need to make happen is that the entire meaning-constructor of *sleep* serve as the first argument of *try*, so that that of *Uther* can serve as the second argument of *try*.

And since *sleep* has negative e at g, and positive t at f, we will need the opposite polarities for the e and t in the first argument of *try*, and we will need an appropriate value-assignment rule. It's not hard to provide the polarity rules: the one we have for implications is applying to positive ones (whole meaning-constructors, and their consequents), whereas this implication is negative (because it's the antecedent of a positive implication). So these polarity rules for negative implications appear to be needed:

(21) a. The polarity of the antecedent of a negative implication is positive.

b. The polarity of the consequent of a negative implication is negative.

It should now be obvious how to do a hookup, so how do we calculate the content?

On a 'semantic VP analysis', the content we want for the first argument of *try* is $\lambda x.\textit{sleep}(x)$, equivalent to just *sleep*. We can get it with the following two principles (a. due to de Groote, b. to Perrier):

(22) a. The value of a positive antecedent is a new symbol, not used previously in the calculation of values.

b. If the value of the antecedent of a negative implication is ξ, and the value of its consequent is Φ, then ξ must occur free in Φ, and the value of the whole implication is $\lambda\xi.\Phi$.

The calculation of values would start with an arrangement like this:

(23)
$$\begin{array}{lcl}
\textit{uther} & : & \begin{array}{c} g\sim e \\ + \\ \textit{uther} \end{array} \\[2ex]
\textit{sleep} & : & \begin{array}{ccc} g\sim e & \multimap & f\sim t \\ - & + & + \\ & \textit{sleep} & \end{array} \\[2ex]
\textit{try} & : & \begin{array}{ccccccccc} (g\sim e & \multimap & f\sim t) & \multimap & g\sim e & \multimap & h\sim t \\ + & - & - & + & - & + & + \\ & & & \textit{try} & & & \end{array}
\end{array}$$

After hooking up *sleep* and *try*, and calculating values, we would get (arbitrarily assuming x as the new symbol assigned as value to the postitive $g\sim e$ position of *try*):

(24)
$$\begin{array}{lcl}
\textit{uther} & : & \begin{array}{c} g\sim e \\ + \\ \textit{uther} \end{array} \\[2ex]
\textit{sleep} & : & \begin{array}{ccc} g\sim e & \multimap & f\sim t \\ - & + & + \\ x & \textit{sleep} & \textit{sleep}(x) \end{array} \\[2ex]
\textit{try} & : & \begin{array}{ccccccccc} (g\sim e & \multimap & f\sim t) & \multimap & g\sim e & \multimap & h\sim t \\ + & - & - & + & - & + & + \\ x & \lambda x.\textit{sleep}(x) & \textit{sleep}(x) & \textit{try} & & \textit{try}(\textit{sleep}) & \\ & = \textit{sleep} & & & & & \end{array}
\end{array}$$

It is now straightforward to finish up the connections so as to get the final result *try*(*sleep*)(*uther*). If we want a more traditional representation with *try* relating an individual and a proposition, we can use a meaning-term for *try* such as $\lambda P.\lambda x.\textit{try}(x, P(x))$, which will yield *try*(*uther*, *sleep*(*uther*)).

2 Quantifiers

Quantifiers is a likely place for people to lose the plot with glue logic (I did, more than once). The way we think about (count) quantifiers (more precisely, quantificational determiners) is as relations between sets, which are produced as the extensions of one-place predicates. *some* relates two sets if they have a nonempty overlap, *every* if the

first is contained in the second, etc. Thus we use the same technique as we did with *try*, but for both arguments, so that the semantic type of a quantificational determiner is $(e \to t) \to (e \to t) \to t$.

Next, we need to identify the relevant locations for content to be delivered to and collected from. By convention, common nouns are treated as accepting e-input at an attribute of (the f-structures of) nouns called VAR, and producing t output at another nominal attribute called RESTR.[7] We'll follow this convention for now. And as an aid to intelligibility, in instantiated structures we'll refer to say g's VAR-value as gv, g's RESTR-value as gr, etc.

So the instantiated constructor for a common noun such as *knight* might be:

(25) *knight* : $(gv \sim e) \multimap gr \sim t$

For a proper match-up, the constructor for *every* will have to start out this way:

(26) *every* : $(gv \sim e \multimap gr \sim t) \multimap \ldots$

The $gv \sim e$ position in the quantifier will now be positive, and able to connect to the matching $gv \sim e$ position in the nominal, and vice-versa for the two $gr \sim t$ positions. And, by our value-calculation rules, the value assigned to the consequent of *every*'s main connective (represented in (26) by . . .) will be *every*(*knight*), as illustrated in the following partial structure:

(27)
$$\begin{array}{ccccccc} g \sim e & \multimap & g \sim t & & & \\ x & \textit{knight} & \textit{knight}(x) & & & \\ \uparrow & & \downarrow & & & \\ (g \sim e & \multimap & g \sim t) & \multimap & \ldots & \\ x & \textit{knight} & \textit{knight}(x) & \textit{every} & \textit{every}(\textit{knight}) & \end{array}$$

The next step is a bit more challenging; what we need to do is create a one-place predicate from the syntactic environment of the quantified NP, so that for example in a sentence such as *every knight sleeps*, the one-place predication tested for containing the extension of *knight* will be *sleep*, whereas in *every knight seems to sleep*, there will be an additional possibility, $\lambda x.\textit{seem}(\textit{sleep}(x))$.

For the e-type argument, the specification will obviously be $g \sim e$: the positive of the quantifier sends to the negative at the subject-position of the verb. We then need to collect t-content to send back to the quantifier, from some location containing the subject. One might think that that some kind of complicated machinery was required to do this, perhaps involving functional uncertainty, but actually something quite simple will suffice for basic coverage: just use a variable (understood as universally quantified) for the f-structure position, both of the second input to the quantifier and its final output. So a final formulation of the meaning-constructor for *every* might be:

(28) *every* : $((\uparrow \text{VAR}) \sim e \multimap (\uparrow \text{RESTR}) \sim t) \multimap (\uparrow \sim e \multimap H \sim t) \multimap H \sim t$

In a one-clause context, the rest of our structure might be start out like this:

[7] In the standard presentation, these attributes are located on the semantic projection rather than f-structure itself.

(29)
$$\begin{array}{lllll} (g\sim e & \multimap & f\sim t & & \\ & \textit{sleep} & & & \\ \ldots\ (g\sim e & \multimap & f\sim t) & \multimap & f\sim t \\ & & & \textit{every}(\textit{knight}) & \end{array}$$

Finishing the hookup and value-propagations, we would get:

(30)
$$\begin{array}{lllllll} & g\sim e & \multimap & f\sim t & & & \\ & y & \textit{sleep} & \textit{sleep}(y) & & & \\ & \uparrow & & \downarrow & & & \\ \ldots & (g\sim e & \multimap & f\sim t) & \multimap & f\sim t & \\ & y & \textit{sleep} & \textit{sleep}(y) & \textit{every}(\textit{knight}) & \textit{every}(\textit{knight})(\textit{sleep}) & \\ & & & & & \text{(FINAL OUTPUT)} & \end{array}$$

If we want a more conventional-looking representation with explicit variable-binding, we can get that too, by formulating the *every*-constructor like this:

(31) $\lambda P.\lambda Q.\textit{every}(x, P(x), Q(x)) \quad : \quad ((\uparrow\text{VAR})\sim e \multimap (\uparrow\text{RESTR})\sim t)\multimap (\uparrow\sim e \multimap H\sim t) \multimap H\sim t$

With this constructor, the final result will be $\textit{every}(x, \textit{knight}(x), \textit{sleep}(x))$.

A worry might be that we choose a wrong f-structure to identify H with; the only possiblity here is g. But if you check this out you'll see that this doesn't allow an acceptable value-assignment in accord with the rules, because nothing will produce t-content at g that contains the variable that is assigned to g as e-content. In general turns out that (21b) suppresses *a priori* hopeless readings (a point developed at length in Dalrymple et al. 1999).

Now what about a more complex structure with scope ambiguity, such as that of *every knight seems to sleep*? Here the instantiated constructors might be:

(32)
$$\begin{array}{rcl} \textit{knight} & : & g\upsilon\sim e \multimap g r\sim t \\ \textit{every} & : & (g\upsilon\sim e \multimap g r\sim t) \multimap (g\sim e \multimap H\sim t) \multimap H\sim t \\ \textit{sleep} & : & g\sim e \multimap f\sim t \\ \textit{seem} & : & f\sim t \multimap h\sim t \end{array}$$

The hookup of the first (compound) argument of *every* to *knight* proceeds as before, but now we have two viable choices to identify H with, f and h. If you calculate the values, you see that the first corresponds to a narrow-scope reading, the second to wide. In terms of content-flow, with the narrow scope reading, the quantifier collects the output of the *sleep*-predication from f, compares it with the *knight* predication, and then collects the result and sends it for processing by *seem*. With the wide-scope reading, the output of *sleep* proceeds onward to *seem*, and then the result of *seem*-*sleep* ($\lambda x.\textit{seem}(\textit{sleep}(x))$) is compared to *knight*.

So *seem* works out, but what about *try*? What we need to be sure of here is that we get only wide scope, not narrow. This turns out to be the case. The instantiated constructors might be as below, where we've already combined *every* and *knight*:

(33) *every*(*knight*) : $(g \sim e \multimap H \sim t) \multimap H \sim t$
sleep : $g \sim e \multimap f \sim t$
try : $(g \sim e \multimap f \sim t) \multimap g \sim e \multimap h \sim t$

The attempt to construct a narrow-scope reading will fail, because if we hook up *sleep* to the $e \rightarrow t$ argument of *every*(*knight*), there won't be any way to finish the hookup, since the negative $f \sim t$ of *try* won't be able to get matched up to anything whose content contains the value of the positive antecedent $g \sim e$. But for the wide-scope reading, *sleep* serves as the $e \rightarrow t$ argument of *try*, the quantified NP sends its e 'subsidiary' output, say, y, to the e argument of *try*(*sleep*), the result *knight*(y) is fed back to the 'paired subsidiary input' (negative $H \sim t$, paired with the quantified NP's positive $e \sim t$), and the final result emerges as *every*(*knight*)(*try*(*sleep*)), which is what is wanted. Or, with more traditional meaning terms, we'd get *every*(x, *knight*(x), *try*(x, *sleep*(x))).

Various more complex problems of quantifier scope are handled correctly by the theory, such as the celebrated five readings of *Every representative of a company saw a sample* (Dalrymple et al. 1999). This is significant, because naive schemes of quantifier scope interpretation by 'extraction' tend to produce linguistically impossible structures in which quantifiers are applied to structures that don't contain the variables they bind.

3 Connections to Deduction

Although we have been presenting this account of glue logic within a 'content flow' interpretation, it is useful to get some grasp on the deductive formulation as well, especially for understanding ways in which several constructors can be thought of as combining into one, by means of the logical rule of Modus Ponens. There is a correspondence between the structures of proof nets, which are what we get by hooking up the positive and negative literals in accordance with the rules, and deductive steps. The simplest-to-understand equivalence is to the 'Gentzen sequent' formulation of deductions, but this formulation tends to strike people as rather unintuitive (see Crouch and van Genabith (2000) for details). Here I will informally sketch some useful connections between proofnet hookups and 'natural deduction' proofs using Modus Ponens and Hypothetical Conditional (assuming an arbitrary premise, drawing some conclusions, and finally discharging the premise to derive an implicational result without assumption). These are also known as Implication Elimination and Introduction; such a system is presented accessibly and used in Asudeh and Crouch (2002).

One useful fact is that in the intuitionistic implicational fragment, every meaning-constructor is of the following form:

(34) $A_0 \multimap \ldots A_n \multimap C$
where C is not an implication, and it is possible for there to be no A_i (e.g. a proper name constructor).

We can call the A_i the 'arguments' and C the '(final) result'.

So one useful result is the following:

(35) Correctly connecting every literal of a constructor M_1 with every literal in the first argument of another constructor M_2 is equivalent to applying Modus Ponens to produce a 'derived constructor' whose glue side is the consequent of the

first implication of M_2, and whose meaning side is the result of applying the meaning-side of M_2 (a function) to the meaning side of M_1 (the argument).

Indeed, we have already illustrated this principle with the application of the *every* constructor to the *knight* constructor to produce an *every*(*knight*) constructor.

For a sentence with subject quantification, such as *every knight sleeps*, the intransitive verb meaning can be used in the same way as the second argument of the quantificational determiner, but object quantification, or wide-scope *seem*, requires a different treatment. We illustrate the former case because we haven't diagrammed it yet. Here are possible instantiated constructors for *Gwen likes every knight*, with polarities and initial values assigned, and the object quantifier and determiner already assembled. Since the initial values are shown, we don't need to include the left-hand-sides (meaning terms):

(36)

$$\begin{array}{c} g\sim e \\ + \\ \textit{gwen} \end{array}$$

$$\begin{array}{ccccccc} (h\sim e & \multimap & H\sim t) & & \multimap & & H\sim t \\ + & - & - & & + & & + \\ & & & & \textit{every}(\textit{knight}) & & \end{array}$$

$$\begin{array}{ccccccc} h\sim e & \multimap & g\sim e & & \multimap & & f\sim t \\ - & + & - & & + & & + \\ & \textit{likes} & & & & & \end{array}$$

The (only) hookup works as follows: the quantifier's positive $h \sim e$ is assigned a novel symbol, say x, by (21a), and is then connected to the verb's negative $h \sim e$, and *gwen*'s positive $h \sim e$ would also be connected to the verb's negative $h \sim e$, leading to $\textit{like}(x)(\textit{gwen})$ as the value of the verb's final output. This gets piped into the quantifier's negative $H \sim t$ (thereby identifying H with f), resulting in final output of $\textit{every}(\textit{knight})(\lambda x.\textit{like}(x)(\textit{gwen}))$ for the quantifier and the whole sentence (where the quantifier's output goes, since $H = f$).

The step of creating a novel value corresponds to the deductive step of making an assumption, the first step in the deductive strategy of Hypothetical Deduction, or Implication Introduction. Propagation steps where the propagated value contains this value as a free variable correspond to deduction steps that depend on this assumption, and finally lambda-binding (21b) corresponds to the final stage of the strategy, discharging the hypothesis, by introducing an implication/function. Observe how the constraint in (21b) that the value of the antecedent must appear free in the value of the consequent corresponds to the constraint on Implication Introduction whereby a premise that is discharged must have actually been used in the production of the consequent.

A final point is how to satisfy a non-first argument of a predicate: what is the deductive counterpart of drawing a line from a positive e to the second argument of a two-place predicate, while the first is still unconnected?

(37) *lance* :
$$g \sim e$$
$$+$$
lance
$$\downarrow$$

like :
$$\begin{array}{ccccccc} h \sim e & \multimap & g \sim e & \multimap & f \sim t \\ - & + & - & + & + \\ & \textit{like} & & & \end{array}$$

This corresponds to using Implication Introduction to temporarily satisfy the first argument, then Modus Ponens to satisify the second argument (that is, drawing a line to the now-first argument of the verb), and finally discharging the hypothesis to derive $\lambda y.\textit{like}(y)(\textit{lance})$ as the content of the partially assembled result. We are now ready to take on the analysis of prenominal modifiers.

4 Prenominal Modifiers

Since the glue-terms of common nouns are of the form $gv \sim e \multimap gr \sim t$, we expect prenominal modifiers to act as if they had glue terms such as $(gv \sim e \multimap gr \sim t) \multimap gv \sim e \multimap gr \sim t$, however these are to be produced. Of the various kinds of prenominal modifiers, one that has been rather problematical for LFG are the 'scoping adjectives' such as *purported*, *alleged*, *former* and *self-styled*, which seem to apply as operators to all of the material between themselves and the head N, and optionally to a continuous segment of the material after it:

(38) a. An alleged Russian racketeer

b. an alleged Racketeer from Russia

c. a self-styled former entrepreneur from Mountain View

d. a former self-styled entrepreneur.

The problem for LFG is that all of these various modifiers would be standardly analysed as members of a set-valued ADJUNCT attribute of the f-structure of the NP, leaving no clear account of the rather obvious way in which the interpretation seem to depend on the constituent structure (Andrews 1983).

Andrews and Manning (1993) proposed to account for this by introducing an additional projection, which they called 'argument-structure', wherein the prenominal modifiers are in effect the heads (with the same arguments structure correspondents as their containing $\overline{\text{N}}$s), and the argument-structures of the following $\overline{\text{N}}$s bear an attribute called ARG. Using the subscript notation for projections introduced in Kaplan (1995), we might have a rule like this for expanding $\overline{\text{N}}$:

(39) $\overline{\text{N}} \quad \rightarrow \quad \begin{array}{c} \text{AP} \\ \uparrow_\alpha = \downarrow_\alpha \\ \downarrow_\phi \in (\uparrow_\phi \text{ADJUNCT}) \end{array} \qquad \begin{array}{c} \overline{\text{N}} \\ (\uparrow_\alpha \text{ARG}) = \downarrow_\alpha \\ \uparrow_\phi = \downarrow_\phi \end{array}$

Note how the AP is the head w.r.t the α-projection (a-structure projection), while the $\overline{\text{N}}$ is the head for the ϕ-projection (f-structure projection).

Now in the full structure for, say, *an alleged racketeer*, we would have f- and a-structures like this, where I've written PRED-features into both to make it clearer what the relationships are (although I suspect that PRED-features as standardly understood don't exist at all, and their presence in f-structure will be seen to cause problems):

(40) a. f-structure:

$$g:\begin{bmatrix}\text{SPEC} & \text{INDEF}\\ \text{ADJUNCT} & \left\{h:\begin{bmatrix}\text{PRED} & \text{'Alleged'}\end{bmatrix}\right\}\\ \text{PRED} & \text{'Racketeer'}\end{bmatrix}$$

b. a-structure:

$$a:\begin{bmatrix}\text{PRED} & \text{'Alleged'}\\ \text{ARG} & b:\begin{bmatrix}\text{PRED} & \text{'Racketeer'}\end{bmatrix}\end{bmatrix}$$

To benefit from the articulation of the a-structure, which more closely follows the relevant features of the c-structure, our basic trick will be to locate the t-type content at the a-structure rather than the f-structure.

This allows us to dispense with the RESTR attribute, so that we can reformulate N constructors as follows:

(41) *racketeer* : $(\uparrow_\phi \text{VAR}) \sim e \multimap \uparrow_\alpha \sim t$

Quantifier constructors will have to be correspondingly altered to something like this:

(42) *a* : $((\uparrow_\phi \text{VAR}) \sim e \multimap \uparrow_\alpha \sim t) \multimap (\uparrow_\phi \sim e \multimap H_\phi \sim t) \multimap H_\phi \sim t$

(Assuming without justification that the t content of sentences is still located at the f-structure). And for the operator adjectives such as *alleged*, something like this will do:

(43) *alleged* : $(\uparrow_\alpha \text{ARG}) \sim t \multimap \uparrow_\alpha \sim t$

In the structure partially depicted in (40), the constructors of *alleged*, *racketeer* and the quantifier will instantiate out to:

(44) *alleged* : $b \sim t \multimap a \sim t$
racketeer : $g v \sim e \multimap b \sim t$
a : $(g v \sim e \multimap a \sim t) \multimap (g \sim e \multimap H_\phi \sim t) \multimap H_\phi \sim t$

It is now possible to hook up the positive $g v \sim e$ of the quantifier to the negative one of *racketeer*, then *racketeer*'s positive $b \sim t$ to *alleged*'s negative one, and finally the latter positive $a \sim t$ to the quantifier's negative one. As a result, the consequent of the main $\multimap$ of the quantifier will get the value $a(\lambda x.\mathit{alleged}(\mathit{racketeer}(x)))$, which is what we want.

alleged is plausibly of the basic semantic type $t \to t$, which in this environment can act as if it were of type $(e \to t) \to (e \to t)$ (leaving in some unnecessary parentheses

to emphasize that this is the type of a mapping from properties to properties), but some other scoping adjectives, such as *confessed*, must basically be of this latter type. So a confessed racketeer is somebody who has confessed that they are or at least have been a racketeer. A plausible constructor, using the meaning of the verb to produce that of the adjective, would be:

(45) $\lambda P.\lambda x.\mathit{confess}(x, P(x)) \quad : \quad (((\text{ADJUNCT} \uparrow_\phi) \text{VAR}) \sim e \multimap (\uparrow_\alpha \text{ARG}) \sim t) \multimap$
$((\text{ADJUNCT} \uparrow_\phi) \text{VAR}) \sim e \multimap \uparrow_\alpha \sim t$

Here we've used iofu to take advantage of the relative 'flatness' of the f-structure (and have assumed that set-membership is transparent to iofu designators); if all of the prenominal modifiers are members of the adjunct set, then there won't be a problem with the e-content for nesting examples such as *former self-styled racketeer*, etc. Having multiple projections thus lets us have our cake one way for the e-content, and in a different way for the t-content. In a suitable structure, the constructors for *a confessed racketeer* will instantiate to:

(46)
$$\begin{array}{rcl} \mathit{racketeer} & : & gv \sim e \multimap b \sim t \\ \mathit{a} & : & (gv \sim e \multimap a \sim t) \multimap \\ & & \quad (g \sim e \multimap H_\phi \sim t) \multimap H_\phi \sim t \\ \lambda P.\lambda x.\mathit{confess}(x, P(x)) & : & (gv \sim e \multimap b \sim t) \multimap gv \sim e \multimap a \sim t \end{array}$$

delivering a final result of $\mathit{a}(\lambda x.\mathit{confess}(x, \mathit{racketeer}(x)))$, which is what is wanted.

We have now achieved the main goal, of getting an analysis of scoping adjectives with conventional rather than restriction projections, but there is still plenty to do. One important task is to integrate the treatment of these adjectives with the 'intersective' adjectives such as *big* and *Swedish*, treated on the basis of much previous work in Dalrymple (2000).[8] These kinds of adjectives can occur predicatively as well as prenominally, and in prenominal position seem to specify a predication that applies conjunctively with that of the head nominal:

(47) a. A big elephant walked in.

b. The elephant that walked in was big.

c. A Swedish student yawned.

d. The student that yawned was Swedish.

Simple prenominal occurrences can be dealt with by a constructor with the same glue term as *confessed*, and a meaning-term such as for example $\lambda P.\lambda x.\mathit{and}(\mathit{big}(x), P(x))$.

But a problem, noted by Kasper (1995), arises when such adjectives are modified by 'proposition-modifying' adverbs such as *possibly*, *obviously* and *presumably*:

(48) a. a possibly Swedish student yawned

b. a presumably large elephant broke into the storehouse

c. an obviously daft proposal was endorsed.

[8]These are different subclasses, gradable vs. nongradable, but this difference is irrelevant here.

The problem is that the adverbs appear to modify the content of the adjective, but not the head noun: a possibly Swedish student is a student whose nationality isn't known with certainty, not someone whose studenthood is in doubt.

Within LFG, the only presently extant solution to this problem is to have two constructors for these adjectives (Dalrymple 2000:265-269, mirroring Kasper's HPSG solution). One is of semantic type $e \rightarrow t$, and provides something for the propositional adverbs to work on, while the second converts the $e \rightarrow t$ constructor, perhaps as modified by an adverb, to an intersective $(e \rightarrow t) \rightarrow (e \rightarrow t)$ one.

There are unfortunately quite a number of ways to implement an analysis of this kind. One issue is whether or not the second constructor should be introduced in the lexicon, or 'syncategorematically' by the PS rules. It's unclear what the empirical effects of this difference might be, although the syncategorematic version wouldn't need to use iofu, while the lexical version would. Another issue is related to an empirical phenomenon, which is the status of recursive adverbial modification. Kasper notes that there can be recursive adverbial modification in these constructions:

(49) a. a deliberately obviously insulting proposal

b. an apparently deliberately obviously insulting proposal

He proposes, and Dalrymple adopts, a left-branching analysis whereby the semantically topmost adverb is the most deeply embedded: *an* [[[[*apparently*] *deliberately*] *obviously*] *insulting*] *proposal*. But this treatment ignores the fact that with adverbs modifying sentences rather than adjectives, non-degree modification by other adverbs seems to be impossible:

(50) a. They (*deliberately) obviously insulted the envoy.

b. They very obviously insulted the envoy.

Degree modification is arguably rather different from other kinds of modification, involving a degree argument of gradable adjects, as discussed briefly by Dalrymple, citing more extensive work by Kennedy (1999) and others.

Setting aside degree modification, the fact that recursive adverbial modification applies only to adjectives suggests that the recursion is a characteristic of the AdjP rather than of the AdvP, allowing us to propose a right-branching structure where the c-structure nesting is more in accord with the scope:

(51) an [apparently [deliberately [obviously [insulting]]]] proposal

Indeed, if we have a rule of the general form AdjP $\rightarrow$ AdvP AdjP, the syntactic structure and semantic composition of these adverbs can be parallel to that of the scoping adjectives, which they resemble. This would be a theoretical improvement, because ultimately we want the 'rules' to be to the greatest extent possible the products of principles; this goal will be advanced if rules with similar semantic effects are formally similar as well.

A right-embedding rule employing an α-projection, similar to the one for scoping adjectives, could look like this:

(52) AdjP → AdvP AdjP
$\uparrow_\alpha = \downarrow_\alpha$ $(\uparrow_\alpha \text{ARG}) = \downarrow_\alpha$
$\downarrow_\phi \in (\uparrow_\phi \text{ADJUNCT})$ $\uparrow_\phi = \downarrow_\phi$

This has the same overall form as our rule for introducing scoping adjectives, except it expands AdjP rather than $\overline{\text{N}}$, and introduces an a-structure head/f-structure adjunct AdvP rather than an AdjP. In line with the literature on $\overline{\text{X}}$-theory, it would clearly be plausible to increase the resemblance by treating AdjP and AdvP as variants of some kind of AP category, distinguished by an additional feature which would be dependent on whether the AP was modifying an $\overline{\text{N}}$ or not.

The resemblance extends to the meaning-constructors for the adverbs, which would have glue terms such as $(\uparrow_\alpha \text{ ARG}) \sim t \multimap \uparrow_\alpha \sim t$, identical to what was proposed for *alleged*. If the modified adjective's first meaning constructor is given the glue-term $((\text{ADJ} \uparrow_\phi) \text{ VAR}) \sim e \multimap \uparrow_\alpha \sim t$, then the adjective and adverb will combine to yield a result of the form $(f \text{ VAR}) \sim e \multimap a \sim t$, where f is the f-structure of the whole NP, and a is the a-structure of the AdjP. Likewise, right-branching stacks of adverbially modified AdjPs will work out to deliver an appropriate meaning-constructor for the topmost AdjP.

So now we have to combine the meanings of the AdjP and the $\overline{\text{N}}$. One question is what is the a-structure integration of the $\overline{\text{N}}$ and its modifying AdjP and daughter $\overline{\text{N}}$? An obvious idea is that it should be the same as that found with the scoping adjectives: the AdjP would be the head and the $\overline{\text{N}}$ the ARG. But this won't combine properly with our proposed treatment of the adverbs, since these are the a-structure heads of the AdjPs, taking the a-structure of their sister AdjP as ARG-value, and this proposal would put the head $\overline{\text{N}}$ under the scope of the adverb, which is definitely not what we want.

Fortunately there is some additional counter-evidence. Andrews and Manning (1993) point out that prenominal intersective/degree adjectives can appear in what Matushansky (2002:13) calls 'asyndetic coordination', in sequence separated by comma-pause, which is not available for scoping adjectives:

(53) a. a ruthless, unscrupulous property developer

b. *a former, self-proclaimed moral guardian

Matushansky adduces some evidence from degree modification that such adjective sequences ought to form some sort of constituent:

(54) a. a very/less cold rainy day

b. an utterly ruthless unscrupulous property developer

The claim is that the degree modifiers modify both adjectives. I find this rather dubious, and more significantly, note the absence of such an apparent constituent in any other position, including postnominal modification:

(55) books yellow with age *(and) moldy with neglect were scattered around the deserted cabin.

So I will continue to follow Andrews and Manning (1993) in supposing that the sequences of AP's are being introduced in a flat structure as multiple sisters to $\overline{\text{N}}$.

The effect we need to produce is that a collection of $e \rightarrow t$ semantic types are combined 'conjunctively' to produce an $e \rightarrow t$ semantic type. We can acheive this by imitating the analysis of coordination in Asudeh and Crouch (2002). In their analysis, the meaning of the final conjunct is taken as a 'seed value', of which the others function as modifiers. We can use the meaning of the head $\overline{\text{N}}$ as seed by having it be the a-structure head as well as the f-structural one. Then for the AdjPs, we can use a 'syncategorematic' meaning-constructor introduced by the PS- rules, such as the following:

(56) $\lambda Q.\lambda P.\lambda x.\mathit{and}(P(x), Q(x))$:
$((\uparrow_\phi \text{VAR}) \sim e \multimap \downarrow_\alpha \sim t) \multimap$
$((\uparrow_\phi \text{VAR}) \sim e \multimap \uparrow_\alpha \sim t) \multimap (\uparrow_\phi \text{VAR}) \sim e \multimap \uparrow_\alpha \sim t$

This looks rather intimidating, but all it says is 'take the meaning of the daughter, and use it as an intersective modifier of the meaning of the mother.' If this constructor is associated with the Kleene-starred AdjP in a rule like:

(57)

$\overline{\text{N}}$	$\rightarrow$	AdjP*	$\overline{\text{N}}$
		$\downarrow_\phi \in (\uparrow_\phi \text{ADJUNCT})$	$\uparrow_\phi = \downarrow_\phi$
			$\uparrow_\alpha = \downarrow_\alpha$

then the meanings of the AdjPs and the daughter $\overline{\text{N}}$ will combine as desired to produce an appropriate meaning for the mother $\overline{\text{N}}$.

It would also be possible to reformulate the constructor (56) so that it would work as a second constructor introduced in the lexicon, as in Dalrymple's analysis; this would require introducing some functional uncertainty and perhaps use of inverse projections (depending on whether and how the AdjP's a-structure was integrated into that of the containing $\overline{\text{N}}$).

This is as far as I will take the development of prenominal adjectives here; the next step would be to investigate degree modification, especially the sorts of phenomena studied by Bresnan (1973) and many others since, most recently Matushansky (2002). But what we have seen so far suffices to show how glue logic can do the work proposed for restriction projections by Andrews and Manning (1999).

An important point is that this analysis requires a significant change in how glue-logic-based semantic assembly operates: in the standard conception, it assembles things found on a single projection, the semantic projection. On this assumption, we face the problem of integrating semantically-relevant information present on different projections, as discussed by Andrews and Manning, and problems arise for which restriction projections provide a solution. It is only by abandoning the semantic projection and allowing meaning to be located at both f- and a-structure projections (and perhaps other places as well) that we can use glue-logic to solve the integration problem, and revert to the standard locational projections of Kaplan (1995). In the next and last section of the paper I will take a quick look at some of the issues which arise in extending the present approach to the analyses of complex predicate constructions in Andrews and Manning (1999).

5 Complex Predicates

Complex predicates are constructions in which several apparent surface verbs appear to function as one unit for some syntactic purposes; in particular they seem to share a single array of grammatical relations. One problem they pose for LFG is that their semantic interpretation sometimes seems to respect a c-structure tree (Alsina 1997) that is considerably more articulated that the f-structure. This phenomenon can be illustrated by the following examples from Catalan:

(58) a. Li acabo de fer llegir la carta
Him.DAT I.finish of make read the map
'I finish making him read the map.'

b. Li faig acabar de llegir la carta
Him.DAT I.make finish of read the map
'I make him finish reading the map.'

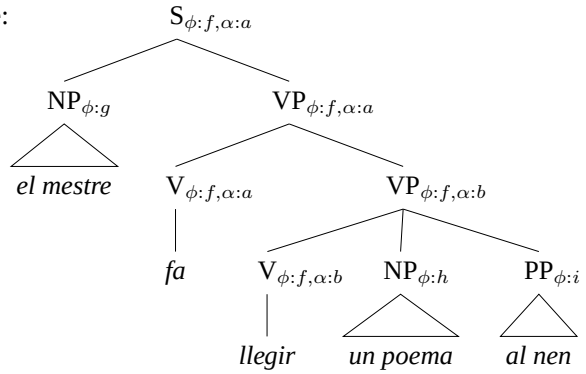
Manning (1992) and Alsina (1993) argue for a rightward-branching c-structure for this construction in Romance, in which each V is followed by a VP dominating the remaining Vs, as well as any NP or PP objects at the end. Andrews and Manning (1999) argued that restriction projections were necessary for a satisfactory analysis of these constructions, but since the situation is quite similar to that with scoping modifiers, one would expect a similar glue logic alternative to be possible. This appears to be the case, but there are some additional factors with complex predicates, such as linking rules and the behavior of adverbs.

We illustrate the approach by representing the tree and relevant skeletal f- and a-structures for a typical Catalan example:

(59) el mestre fa llegir un poema al nen
the teacher makes read a poem by-the child
The teacher makes the child read a poem

The tree-nodes are subscripted with indexes for their f- and a-structure correspondents in the structures below:

(60) a. c-structure:



b. f-structure:

$$f\!:\begin{bmatrix} \text{SUBJ} & g\!:\begin{bmatrix}\text{SPEC} & \text{DEF}\\ \text{PRED} & \text{`Mestre'}\end{bmatrix}\\ \text{PRED} & \text{`Far-llegir' (??)}\\ \text{OBJ2} & h\!:\begin{bmatrix}\text{SPEC} & \text{INDEF}\\ \text{PRED} & \text{`Poema}\end{bmatrix}\\ \text{OBJ} & i\!:\begin{bmatrix}\text{SPEC} & \text{DEF}\\ \text{CASE} & \text{DAT}\\ \text{PRED} & \text{`Nen}\end{bmatrix}\end{bmatrix}$$

c. a-structure:

$$a\!:\begin{bmatrix}\text{PRED} & \text{`Far'}\\ \text{ARG} & b\!:\begin{bmatrix}\text{PRED} & \text{`Llegir'}\end{bmatrix}\end{bmatrix}$$

Note the problem with the PRED-feature of the main f-structure f: if we assume that both the causative and causee verb are contributing a PRED-feature, then these are expected to clash; since in this presentation the PRED-features play no clear theoretical role, I've just combined them with a hyphen (my serious proposal is that PRED-features as standardly conceived don't exist, and that what they roughly correspond to in reality[9] is not located in f-structure). I've also assumed an analysis whereby dative objects in Romance are taken to be dative OBJ, combining aspects of Simpson's (1991) treatmeant of case in Warlpiri, and Alsina's (1996) analysis of Romance. Nothing is supposed to depend on this.

What does matter is how to get a reasonable semantic interpretation out of the two feature-structures. We retain the idea that the nominal e content is delivered to the f-structure nodes, however to make use of the a-structure, we'll clearly have to suppose that the t-content delivered by verbs goes to a-structure rather than f-structure, which seems reasonable since the t-content of common nouns goes there too. So our next problem is to provide a linking theory. The arguments of *Llegir* are the OBJ and the OBJ2, so we want things to work out as if there were the following instantiated meaning-constructor:

(61) *llegir* : $h\!\sim\!e \multimap i\!\sim\!e \multimap b\!\sim\!t$

Fa on the other hand is a bit more complicated, because the normal interpretation of such a sentence would have *al nen* serving both as the acted-upon argument of *fa* and the understood subject of *llegir*; this is basically the same idea as the meaning-constructor for *try*, except that the semantically shared argument is the object rather than the subject: the following constructor will do:

(62) *far* : $(i\!\sim\!e \multimap b\!\sim\!t) \multimap i\!\sim\!e \multimap g\!\sim\!e \multimap a\!\sim\!t$

How are these constructors to be produced?

[9]Morphologically relevant 'tags' which indicate for example that the verbs 'motional go' (*John went to Braidwood*) and 'expressive go' (*John went 'oops'*) have the same morphology.

There are various possibilities; one is to use the argument-structure scheme from Andrews and Manning (1999), in particular the TERMS-list idea (Andrews and Manning discuss and argue against some alternatives). This proposal is that intermediate between grammatical relations and semantic roles is an ordered list of argument-positions, accessed as the value of the a-structure attribute TERMS, with the empirical properties proposed in Manning (1996) and other work. Ordinary verbs have a 'closed' TERMS-list that can't be extended, but the lexical entries of 'light verbs' (those whose semantically subordinate verbs form a complex predicate rather than serve as a complement) have open-ended ones that combine with those of their ARG-values to produce a combined TERMS-list for the whole structure. The idea of combining TERMS-lists is motivated by the observation that the arguments of complex predicates typically take grammatical relation arrays similar or identical to those of single predicates; the ordering of arguments imposed by the TERMS-list seems to provide a useful intermediary stage between semantic roles and grammatical relations.

Positions on the TERMS-list are conveniently notated as TERMS.1, TERMS.2, etc. So the uninstantiated constructor for *llegir* would now become:

(63) *llegir* : $(\uparrow_\alpha \text{TERMS.2}) \sim e \multimap (\uparrow_\alpha \text{TERMS.1}) \sim e \multimap \uparrow_\alpha \sim t$

This would function in the context of a revised a-structure for (59):

(64) $$a:\begin{bmatrix} \text{PRED} & \text{'Far'} \\ \text{TERMS} & < t_1, t_2, \ldots > \\ \text{ARG} & b:\begin{bmatrix} \text{PRED} & \text{'Llegir'} \\ \text{TERMS} & < t_2, \ldots > \end{bmatrix} \end{bmatrix}$$

where the $< t_2, \ldots >$ portion of the a-TERMS-list is supposed to be the same object as the entire b-TERMS-list. To get the desired combination of the TERMS-lists, we can put the following equation in the lexical entry of *far*:

(65) $(\uparrow_\alpha \text{TERMS TAIL}) = (\uparrow_\alpha \text{ARG TERMS})$

(assuming a LISP/Prolog-style right-branching tree representation of TERMS-lists). Now the following uninstantiated meaning-constructor will work for *far*:

(66) *far* : $((\uparrow_\alpha \text{TERMS.2}) \sim e \multimap (\uparrow_\alpha \text{ARG}) \sim t) \multimap$
$(\uparrow_\alpha \text{TERMS.2}) \sim e \multimap (\uparrow_\alpha \text{TERMS.1}) \sim e \multimap \uparrow_\alpha \sim t$

What remains is to get the e-content from the f-structure to the a-structure; the obvious way to do it in this context is to use meaning-constructors as linking rules. The following collection will do the job (semantic type information omitted due to irrelevance):

(67) $\lambda x.x$: $(\uparrow_\phi \text{SUBJ}) \multimap (\uparrow_\alpha \text{TERMS.1})$
$\lambda x.x$: $(\uparrow_\phi \text{OBJ}) \multimap (\uparrow_\alpha \text{TERMS.2})$
$\lambda x.x$: $(\uparrow_\phi \text{OBJ2}) \multimap (\uparrow_\alpha \text{TERMS.3})$

To develop this approach properly, we would have to investigate how these constructors would get added to those for the sentence, an issue we won't examine here.

Instead we will look at the interpretation of adverbs, in particular the somewhat surprising fact that adverbs in complex predicates are frequently capable of modifying either the upper or the lower predicate, even when their c-structure position appears to be within the lower VP (Alex Alsina p.c., cited in Manning (1992) and Andrews and Manning (1999:55, 126)):

(68) a. He fet beure el vi a contracor a la Maria.
I have made drink the wine against x's will to the Mary
'I have made Mary drink the wine against her/my will.'

b. Volia tastar amb molt d'interès la cuina tailandesa
I wanted to taste with much interest the cuisine Thai
'I wanted to taste Thai food with much interest.'
(with *with much interest* most naturally modifying *want*)

This actually works out better with restriction projections than in the present approach, but we can still get a tolerable analysis by using inverse projections. In order for there to be an ambiguity, the adverb should presumably be attached to the structure as an f-structural ADJUNCT, which will give it a relationship to both verbs. But it then must be capable of applying to the t-value of either of the two a-structures that will correspond to that f-structure under the composite relation $\phi^{-1}\alpha$ (if f is the f-structure that the adverb is ADUNCT of, then $\phi^{-1}(f)$ is any c-structure node c that has f as its f-structure correspondent, and $\alpha(c)$ is that c-structure node's a-structure correspondent). So a constructor like this will work (assuming that *a contracor* maps 'actions' (semantic VPs) onto actions:

(69) *acontracor* : $((A\,\text{TERMS.1}) \sim e \multimap A \sim t) \multimap (A\,\text{TERMS.1}) \sim e \multimap A \sim t$
where $A = \alpha(\phi^{-1}((\text{ADJUNCT} \uparrow_\phi)))$

This is rather awkward; hopefully further work will reveal a cleaner way to it. One possibility would be to have the adverbial meaning-constructor work on the a-structure of the AdvP:

(70) *acontracor* : $((\uparrow_\alpha \text{TERMS.1}) \sim e \multimap \uparrow_\alpha \sim t) \multimap (\uparrow_\alpha \text{TERMS.1}) \sim e \multimap \uparrow_\alpha \sim t$

There would then be an additional annotation, perhaps introduced syncategorematically with the AdvP, which identified the AdvP's a-structure with some a-structure in $\alpha(\phi^{-1}(\phi(\uparrow))$:

(71) VP $\rightarrow$ V ... AdvP ...
$\downarrow_\alpha = \alpha(\phi^{-1}(\phi(\uparrow)))$

Yet another possibility would be to have each VP introduce a pair of meaning-constructors, one shifting the content from the VP's a-structure to it's f-structure, another shifting it back:

(72) $\lambda x.x$: $\uparrow_\alpha \multimap \uparrow_\phi$
$\lambda x.x$: $\uparrow_\phi \multimap \uparrow_\alpha$

This allows for two classes of modifiers, one type constrained by the a-structure, another only by the f-structure, and able to have its application freely interleaved with the former type.

So there are too many possibilities, and too little empirical guidance on which one to choose, but perhaps more attention to the literature on adverb syntax and typology would provide some illumination.

Appendix A: Anaphora

In this section I briefly sketch an approach to extending the content-flow interpretation of glue logic to anaphora. The treatment of anaphora seems to me to be rather more difficult than that of basic predicate-argument structure and quantification, and it may be that content-flow doesn't make it very much easier, but it still seems worthwhile trying to work out how it could be done. We basically adapt the algebraic formulations of de Groote (1999) to produce logical forms, without proving the results that show that we have a correctness criterion for proofnets (if we want one, we can just use de Groote's). Unlike the main text of this paper, this appendix does not purport to be readable independently of Dalrymple (2000).

The treatment of anaphora involves a new class of locations for content-delivery, which are represented as n-tuples of f-structures, enclosed in angle-brackets. E.g. $< g, h >$. The values of such locations are then n-tuples of meanings, for ordinary pronouns they'd be e-type meanings, such as say $< X,$ *david* $>$ (variables introduced by certain quantifiers such as 'some' get to function in effect as names in later discourse. In a deductive formulation, two location-value pairs are equivalent if they can be mapped onto each other by applying the same permutation to both sides; this effect can be captured in the content-flow interpretation by allowing one location to be linked to another if the f-structure-list in one can be permuted into the f-structure-list of the other, the values being permuted likewise. This implies that the distinct locations are actually multi-sets, rather than n-tuples. Considerable use is made of list-variables, like C, something like $< g, C >$ denotes a list with first member g and remainder C(it might be better to write something like $< g|C >$, following Prolog conventions). In linking, these variables can be seen as unifying like Prolog variables.

On the glue side, we'll also have pairs joined by $\otimes$, where the first member is always an ordinary f-structure, the second an f-structure sequence; corresponding to these we'll have pair-values where the first member is an f-structure value and the other a list.

A successful hookup will now start with a context list amongst the ordinary constructors, represententing the initial context for the sentence, and conclude providing values for one resultant context list and a meaning for the whole sentence.

A final point is that pronouns are assumed to have referents determined by the value of their ANTECEDENT attribute, whose value is an f-structure chosen from the context list, the ultimate effect is that the semantic value of the pronoun comes up identical to that of its antecedent.

I'll go through the issues that arise in extending value-calculation by working through the use of the constructor for *someone* Dalrymple (2000:303-306). A sen-

tence such as *someone arrived* can be used with no preceeding context, so the initial context list can be $<>$. On the other hand after it has been said, we can have a pronoun referring to the person who arrived, so the resultant context-list contains a referent for someone. An initial set of instantiated constructors might look like this (I've slightly modified the meaning-side of the *someone*- constructor):

(73) $<>$: $<>$

$\lambda C.\lambda S.a(X, person(X), s(S([X, < X, C >]))), x(S([X, < X, C]))$

: $C_0 \multimap ([g \sim e \otimes < g, C_0 >] \multimap [H \sim t \otimes C_1]) \multimap [H \sim t \otimes C_1]$

arrive : $g \sim e \multimap f \sim t$

We can begin by hooking up the initial context to the first argument of *someone*. The effect of this is to identify C_0 with $<>$, so that the constructor-set is in effect further instantiated to:

(74) $<>$: $<>$

$\lambda C.\lambda S.a(X, person(X), s(S([X, < X, C]))), x(S([X, < X, C]))$

: $<> \multimap ([g \sim e \otimes < g >] \multimap [H \sim t \otimes C_1]) \multimap [H \sim t \otimes C_1]$

arrive : $g \sim e \multimap f \sim t$

Note that although this instantiation is in effect caused by the connection of the literals, there is no direct literal-connection relationship between the two instances of C_0 on the glue-side of *someone*.

Now we have to deal with the rather complex second argument of *someone*. The polarity principles need to be extended to deal with $\otimes$, and they are rather obvious; the two components of a positive tensor are both positive, of a negative, negative. This corresponds to the idea that a something assigned as value to a positive tensor will be split into two components, each available at one of the premises, while negative tensor will combine values presented to its premises in order to satisfy a request for a value at its conclusion. Therefore we will have arrows heading out from $g \sim e$ and $< g >$ in the (positive) antecedent, and into the $H \sim t$ and C_1 of the (negative) consequent. The first members get hooked up to the literals of *arrive*, while the two contexts get hooked up to each other:

(75) $<>$: $<>$

$\lambda C.\lambda S.a(X, person(X), s(S([X, < X, C]))), x(S([X, < X, C]))$

: $<> \multimap ([g \sim e \otimes < g >] \multimap [H \sim t \otimes C_1]) \multimap [H \sim t \otimes C_1]$

arrive : $g \sim e \multimap f \sim t$

The final $H \sim t$ and C_1 of *someone* are then our unconnected final outputs.

So that's the hookup, what about the values? There's really two aspects of this, one that we want the values to be useful logical forms, the other that we want to use them to provide a correctness criterion for the hookup. I haven't proved anything about the

latter point; one could always simply use de Groote's scheme for that. The utility of logical forms on the other hand can't be proved, but only demonstrated in a practical manner. The big challenge is what to do about complex arguments such as the second one of *someone*. By the value-calculation rule so-far, we expect that the antecedent of this argument, being a positive antecedent of a condition, will be assigned as value a novel symbol, say T. The associated glue-term is $g \sim e \otimes <>$, so we need to produce values for the first and second component of the tensor, and it would be good to deal with cases where the context-list isn't empty. The crucial idea is that the variable T is going to be of a type which guarantees that we can perform various operations on it, in particular we can extract its regular meaning component with the function s, and it context component with the function c (corresponding to the left and right square-root operators of de Groote, *sem* and *cxt* of Dalrymple (2000)). When the context is an n-tuple with members, we will also be able to get at those members with projection functions, which will just be designated with subscripts, e.g. $c(T)_1$ for the first member of the context-list (which wouldn't be defined for $<>$, but would be for $< g >$.

So the total effect is that the value of the positive $g \sim e$ in *someone* will be $s(T)$, and of $< g >$, $c(T)$, or equivalently, $< c(T)_1 >$. The propagation rules then entail that the negative $H \sim t$ of *someone* gets the value *yawn*$(s(T))$, while the negative $< g >$ gets the value $c(T)$. Then the value calculation rules for negative tensors say that we build a value for the tensor from those of the right and left components by making a pair with those components (corresponding to de Groote's algebraic rule that left and right square roots of m multiply to produce m). Finally, by lambda-abstraction, the value of the entire conditional is calculated to be:

(76) $\lambda T.[$*yawn*$(s(T)), c(T)]$

This is what is supplied as argument to the lambda-bound S in the *someone* constructor, where it gets applied to the argument $[X, < X, C >]$, equivalent to $[X, < X >]$ in this case, so the output of this is then:

(77) $[$*yawn*$(X), < X >]$

To finally get the value for *someone*, we subsitute the s component of this into the appropriate position in the quantificational expression, and return the context component unaltered. The resulting pair becomes the value of the final $[H \sim t \otimes < g >]$ in the glue-term of *someone*, with the meaning and context components getting distributed to the two subterms by the positive tensor rule.

There are various more complex cases to consider, one of the more complicated ones is when we have a pronoun in the scope of the quantifier, but not bound by it, perhaps as in *somebody*$_i$ *forgot his*$_j$ *name* (explaining why a delegation of peasants petitioning the Beloved Leader were all executed). The variables in the context-list terms give us (unfortunately, it seems to me) an option in how to hook things up; we can feed the initial context into the quantifier, and deal with the pronoun while processing the quantifier's scope, or do the pronoun first and then the quantifier. The former option is the more confusing one, so we'll discuss it. To provide an antecedent for the pronoun, the initial context-list must be non-empty, say it's $<$ *kim* $>:< o >$. Hooking this up to the first argument of *someone* then gives us a positive antecedent for the second

argument of the form $g \sim e \otimes < g, o >$. Now one might mistakenly think that the o here would get the value *kim* (perhaps the whole thing would have value $< X,$ *kim* $>$), but this would be wrong. Rather the value is $[s(T), < c(T)_1, c(T)_2 >]$. Now when we crank throught to compute the value of the consequent of the conditional, the pronoun is processed using $c(T)_2$ as the semantic value for the pronoun. Only when the semantic value of the entire argument (a function, represented by S on the meaning-side of the *someone*-constructor, does the context o get associated with *kim.* Another point is that in processing the pronoun meaning-constructor (77) (Dalrymple 2000:302), we'll have to result to the trick of permuting the items on the context list, to get the reference to *kim* into first position.

I think this shows that content-propagation can be applied to anaphora, whether it will be helpful is another matter.

Bibliography

Alsina, A. 1993. *Predicate Composition: A Theory of Syntactic Function Alternations*. PhD thesis, Stanford University.

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

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

Alsina, A., J. Bresnan, and P. Sells (Eds.). 1997. *Complex Predicates*. Stanford, CA: CSLI Publications.

Andrews, A. 1983. A note on the constituent structure of modifiers. *Linguistic Inquiry* 14:695–7.

Andrews, A. D., and C. D. Manning. 1993. Information-spreading and levels of representation in lfg. Technical Report CSLI-93-176, CSLI, Stanford University, Stanford, CA. Available at: `http://www.sultry.arts.usyd.edu.au/cmanning/papers`.

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

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

Bresnan, J. W. 1973. Syntax of the comparative clause construction in English. *Linguistic Inquiry* 4:275–343.

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

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

Dalrymple, M. 2000. *Lexical Functional Grammar*. Academic Press.

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

Dalrymple, M., J. Lamping, F. Pereira, and V. Saraswat. 1999. Quantification, anaphora and intensionality. In Dalrymple (Ed.), 39–90.

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

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

Kaplan, R. M. 1995. The formal architecture of LFG. In M. Dalrymple, R. M. Kaplan, J. T. Maxwell, and A. Zaenen (Eds.), 7–27.

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).

Kennedy, C. 1999. *Projecting the Adjective*. New York: Garland Press. originally UMass Amherst dissertation, 1997.

Manning, C. D. 1992. Romance is so complex. Technical Report CSLI-92-168, CSLI, Stanford University, Stanford, CA.
URL: http://www.sultry.arts.usyd.edu.au/cmanning/papers.

Manning, C. D. 1996. *Ergativity: Argument Structure and Grammatical Relations*. Stanford, CA: CSLI Publications.

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

Matushansky, O. 2002. *Movement of Degree/Degree of Movement*. PhD thesis, MIT. available from MITWPL.

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

Simpson, J. H. 1991. *Warlpiri Morpho-Syntax*. Dordrecht: Kluwer Academic.