Glue logic vs. Spreading Architecture in LFG*

Avery Andrews, ANU, September, 2003
talk given at ALS2003, University of Newcastle, NSW
comments welcome

Although the f-structures of LFG look like an intuitively clear representation of many aspects of meaning, it has seemed somewhat harder than one might expect to connect them to conventional logical forms, over which entailment and other semantic properties and relations can be defined by standard methods. In this talk I will present what I believe to be a relatively easy formulation of the 'glue logic' approach (Dalrymple 1999, 2000), and use it to eliminate the 'restriction projections' used by Andrews and Manning (1993, 1999) in their analyses of scoping modifiers and complex predicates, allowing the analyses to work with the conventional 'locational projections' of Kaplan (1995).

The basic idea is to apply recent work by de Groote (1999) and Perrier (1999) to formulate semantic assembly as a system of 'content flow', whereby the 'meaning constructors' introduced by lexical items (and perhaps grammatical constructions) are construed as 'content processors' which collect inputs from and deliver outputs to various locations in the f-structure. If the content-processors can shift content between different locations in the f-structure, it is natural to ask whether they could shift content between different semantically relevant projections, as required by the Andrews and Manning analyses, and the answer turns out to be that they can.

As a theory of semantic assembly, glue logic is agnostic about the exact nature of the meaning-elements that are being assembled; there is a tradition, of using Montague's Intentional Logic without the intensions in introductory expositions. I will follow this tradition here. First I will talk about basic predicate-argument structure, then quantifiers, and finally, scoping modifiers. I will close with some more general remarks.

1 Predicates and Arguments

In the semantic format we'll be using, the simplest kind of meaning-constructor is one for a proper name. All meaning-constructors consist of two components, on the left, the 'meaning-side', a specification of a meaning, on the right the 'glue-side', a specification of the information needed to control assembly. This will consist of a specification of (Montogovian) semantic type (based on e and t, in the introductory version), combined with f-structural 'locational' information (where the input or output will be collected from/delivered to). So if a proper name such as *Art* is inserted under an N with f-structure g, the resulting constructor will look like this, where the semantic type is subscripted to the location (but is however often omitted, although not in this paper):

(1) *art* : g_e

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

In terms of content-flow, this says that the content *art*, of type e (entity), is delivered to the f-structure g (perhaps the subject of a sentence).

A bit more complex is the meaning-constructor of an intransitive verb, which would collect content of type e from its subject('s f-structure), and deliver content of type t (truth-value) to its own f-structure, which is also that of the clause. This input-output connection is represented with the $\multimap$ symbol, popularly read as 'lollipop', representing implication in linear logic. So the if the f-structure of the whole sentence is f, the constructor for *sleep* will be:

(2) *sleep* : $g_e \multimap f_t$

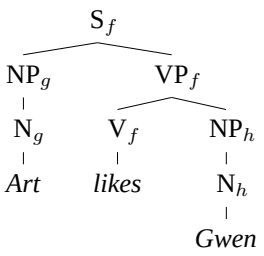
It's an important principle that the semantic type of the meaning-side is fully determined by the structure of the glue side. Here the semantic type has to be a function from type e to type t; this is usually represented partially by writing the meaning side as a lambda-expression, such $\lambda x.\textit{sleep}(x)$. To reduce clutter, I will usually not do this. One could also subscript the meaning-sides with their type: $\textit{sleep}_{e \to t}$.

Now the idea of hookup is to connect *Art*'s e (output) to *sleep*'s e, (input), with the final output coming out at *sleep*'s t. We can diagram the idea of content being delivered to and collected from f-structure locations with a highly explicit picture like this (positions of the meaning and glue sides of the meaning-constructors swapped):

(3)
$$f\!:\!\begin{bmatrix} \text{SUBJ} & g\!:\!\big[\text{PRED} \quad \text{'Art'}\big] \\ \text{PRED} & \text{'Sleep(SUBJ)'} \end{bmatrix} \quad \begin{matrix} e \bullet \\ e \circ \\ t \bullet \end{matrix} \quad \begin{matrix} g_e & : & \textit{art} \\ g_e \multimap f_t & : & \textit{sleep} \end{matrix}$$

We see *Art*'s constructor delivering its content to g (output port represented by a solid dot), and *sleep*'s collecting from g (input represented by circle) and delivering to f. These constructors can then be assembled by adding an arrow to connect the output to the input at g. An important fact which (3) doesn't represent is that there can be multiple input and output ports of the same type at an f-structure; if there are multiple ways of hooking them up, this may result in ambiguity, although it is also possible that constraints to be discussed later will block some of the hookups.

To see a case where the syntactic structure makes a difference, we look at a transitive sentence such as *Art likes Gwen*. We'll follow Montague Grammar and Marantz (1984) in treating a transitive verb as something that takes an e (from the direct object) as input, and yields an intransitive predicate as output, i.e., something of type $e \multimap (e \multimap t)$. So if we have this full syntactic structure:

(4)


$$f\!:\!\begin{bmatrix} \text{SUBJ} & g\!:\!\big[\text{PRED} \quad \text{'Art'}\big] \\ \text{PRED} & \text{'Like(SUBJ, OBJ)'} \\ \text{OBJ} & h\!:\!\big[\text{PRED} \quad \text{'Gwen'}\big] \end{bmatrix}$$

the f-structure of the object will be h, and the constructors we want will be:

(5) $\begin{array}{rcl} \textit{art} & : & g_e \\ \textit{gwen} & : & h_e \\ \textit{like} & : & h_e \multimap (g_e \multimap f_t) \end{array}$

Now the f-structure locational information will control which name's output gets hooked up to which of the verb's inputs, so that the syntactic structure is seen to contribute to the semantic interpretation in the right way.

To move from these ideas to a system that actually does something we need to specify a number of more things. First, how is the f-structure information in the glue-sides produced? The answer is from the lexicon, via the LFG processes of instantiation and functional resolution. In the lexicon, meaning-constructors appear containing expressions using function application and the $\uparrow$-metavariable, which in a lexical item means 'the f-structure correspondent of the c-structure node over me', so that the lexical form of the constructors above will be:

(6) $\begin{array}{rcl} \textit{art} & : & \uparrow_e \\ \textit{gwen} & : & \uparrow_e \\ \textit{like} & : & (\uparrow \text{OBJ})_e \multimap ((\uparrow \text{SUBJ})_e \multimap \uparrow_t) \end{array}$

So when the lexical items are used in structure (4), the uninstantiated constructors instantiate to:

(7) $\begin{array}{rcl} \textit{art} & : & g_e \\ \textit{gwen} & : & h_e \\ \textit{like} & : & (f \text{OBJ})_e \multimap ((f \text{SUBJ})_e \multimap f_t) \end{array}$

And the last step, functional resultion, replaces the functional designator expressions with the f-(sub)structures they generate to produce (5).

Next, we need a formal scheme to officially identify the inputs and the outputs, which we do by defining a notion of 'polarity', as follows:

(8) Polarity Rule 1: An entire glue side has positive polarity.
Polarity Rule 2: If an implication has positive polarity, then its antecedent has negative polarity and its consequent positive.

So by (8), the polarities for the constructors will be:

(9) $\begin{array}{rcl} \textit{art} & : & g_e \\ & & + \\ \\ \textit{gwen} & : & h_e \\ & & + \\ \\ \textit{like} & : & h_e \quad \multimap \quad (g_e \quad \multimap \quad f_t) \\ & & - \quad + \quad - \quad + \quad + \end{array}$

where the polarity of an implication will be written under its implication symbol. Finally, inputs and outputs are characterized as follows:[1]

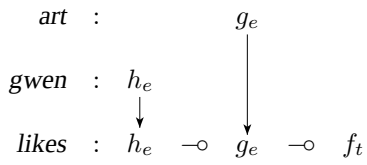
[1]This is the polarity convention used in the LFG literature; unfortunately de Groote (1999) and Perrier (1999) use the opposite.

(10) Inputs and Outputs: a positive literal is an output, a negative literal an input, where a literal is an expression consisting of a location subscripted by a type.

Given polarity, we can formulate the 'Hookup Rule':[2]

(11) Hookup Rule: Every negative literal must by connected to one and only one positive literal, and every positive to one and only one negative, except for one positive literal, the final output, located at the f-structure of the sentence.

Leaving out the polarities to reduce clutter, Hookup gives us:

(12)


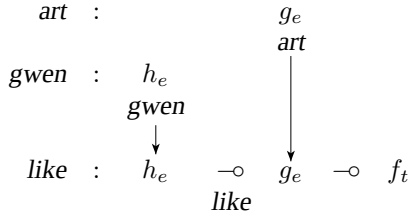
We've also begun leaving out rightmost parentheses (that is, linear implication is taken to be right-associative).

There are further constraints on valid hookup, but they are best formulated in terms of the final ingredient we need, the content-flow rules that produce the logical forms we want. The basic idea is that every formula and sub-formula must receive content, according to a scheme whereby content is assigned initially to certain formulas, and then propagates (monotonically; no overwriting allowed).

We begin by assigning the meaning side of each constructor as content to its (entire) glue-side:

(13) Content Flow Rule 1: The content of a glue side is its meaning side,

The result of appplying this to (12) is (writing the content of an implication under its implication symbol):

(14)


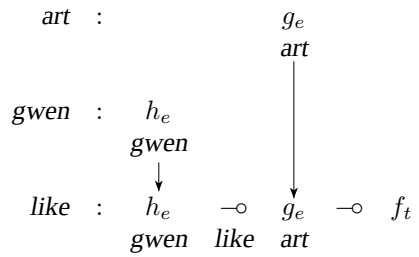
Next we progate content from positive literals (output ports) to negatives that they are linked to (input ports):

(15) Content Flow Rule 2: The content of a positive literal is propagated to the negative that it is linked to.

Two applications of this produce:

[2]Technically, hookups obeying this rule are 'proof structures' (de Groote 1999:4).

(16)

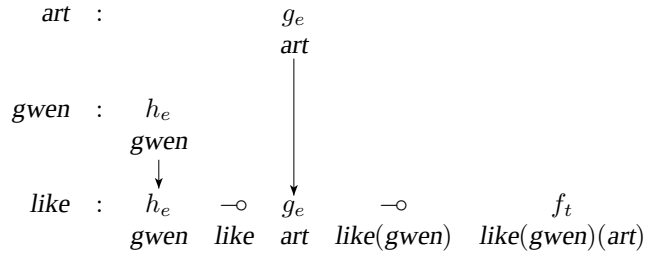


And finally a somewhat more insteresting rule, for calculating the content of the consequent of an implication from that of the implication and its antecedent:

(17) Content Flow Rule 3: If an implication has a as its own content, and b as its antecedent's content, then the content of its consequent is $a(b)$ (the function a applied to the argument b).

This is where the strict correlation between the semantic type of the meaning-side and the structure of the glue-side does its work: if for example the type of the transitive verb constructor here didn't take two type e arguments, the application of (17) wouldn't make sense. But since it is of type $e \rightarrow e \rightarrow t$, everything is fine, and two applications of CFR 3 produce:

(18)



So we've got logical forms being produced by the f-structure.

From some points of view this may not look very illuminating; it simply says that the meaning is what you get by applying a function named *like* to the arguments *gwen* and *art* in succession. But this function might be something with some useful content, such as perhaps an explication along the lines of:

(19) $\lambda Y.\lambda X.$

when X is near Y, X feels something good when X thinks about Y, X feels something good

Then the resultant logical form will be equivalent to (19) with the lambda's removed and (the meanings of) *gwen* and *art* substituted for Y and X respectively, and since the f-structure can be connected to overt phrase structure in a wide variety of ways, as discussed in the LFG literature we will have managed to express the connection between semantic roles and overt positions across a typologically diverse range of languages, as discussed in the LFG literature (Bresnan 2001, Falk 2001, Dalrymple 2000).

But there is one more important issue to deal with, which also address a question you might be asking, what does this have to do with 'logic', anyway? The original formulation of glue-logic was in terms of deductions in a somewhat novel logical

system called 'linear logic':[3] The meaning-constructors were premises for a deduction of which the conclusion was that the meaning of the sentence was such-and-such. 'Hookups' such as we have been discussing have been part of linear logic from the beginning, where they are called 'proof structures', and used to represent classes of proofs (in a restricted version of linear logic), differing only in the order of deductive steps. However it has long been known that conforming to the Hookup Rule is not enough to guarantee that a proof-structure depicts a class of valid proofs: a further 'correctness criterion' is necessary (a proof- structure meeting the correctness condition is a 'proof-net'). The traditional formulations of the correctness criterion are somewhat difficult,[4] but de Groote (1999) proposed a criterion in terms of the propagation of algebraic values, and Perrier (1999) showed how to use logical forms as these algebraic values. For the present purposes, we can formulate this criterion as follows:[5]

(20) Correctness Condition: Every (sub-)formula must receive content according to the content-flow rules, and the content of every constructor must appear in the final output.

The Correctness Condition is needed to exclude certain hookups that don't produce valid readings (and don't correspond to valid deductions in the deductive formulation).

To see an example of a bad hookup excluded by (20), consider the constructors for *Art obviously likes Gwen*. This adverb in some sense modifies the proposition, saying that 'it is obvious that Art likes Gwen', so that we'd want its semantic type to be $t \multimap t$. Furthermore it would appear to take input from and return output to the top f- structure f. So adding it to the set of meaning-constructors would give us:

(21)
$$\begin{array}{rcccccc} \textit{art} & : & & & g_e & & \\ \textit{gwen} & : & h_e & & & & \\ \textit{like} & : & h_e & \multimap & g_e & \multimap & f_t \\ \textit{obviously} & : & & & f_t & \multimap & f_t \end{array}$$

There are two ways of satisfying the Hookup Rule, the sensible way, where the output of *like* goes to the input of *obviously*, and the perverse way, where the output of *obviously* is hooked up to the input of *obviously*. The perverse hookup violates the Correctness Condition.

Perrier's version of the Correctness Condition makes it possible to forget about proof and think of semantic assembly as plumbing (however the deductive view very likely has its uses; for example so far I am finding it to be more natural for thinking about partial semantic assembly).

[3] For a relatively elementary introduction to linear logic for linguists, see Crouch and van Genabith (2000).
[4] See for example the one presented in Fry (1999).
[5] I suspect that it might be possible to simplify this a bit, but will not attempt that here.

2 Quantifiers

The problem I'm going to apply this to is NPs with prenominal 'scoping' adjectives such as *former* and *alleged*, which appear in quantified NPs, so quantifiers, or more precisely, quantificational determiners, is the next issue we'll have to tackle. In our Montogovian framework, these are treated as designating relations between sets. such as overlap, non-overlap, inclusion etc., so that *some knights sleep* is true iff the set of kights overlaps with the set of sleepers, etc. (this is 'generalized quantifiers (Barwise and Cooper 1981)). Since the semantic type of a set-designator is $e{\rightarrow}t$, that of a relation between sets will be:

(22) $(e{\rightarrow}t){\rightarrow}(e{\rightarrow}t){\rightarrow}t$

So what about the locational information? There is a convention to the effect that the first argument of a quantifier is taken to be the nominal content of the NP it appears in (its 'restriction'), the second the predication provided by its environment (its '(nuclear) scope'). The nominal content is of semantic type $e{\rightarrow}t$, but where will its inputs and outputs be located? There is a useful convention, borrowed from HPSG, that the e input is located at the value of an attribute VAR, the t output at the value of an attribute RESTR, both of the NP's f-structure (or its 'semantic projection' in the standard formuluations; we will discuss and argue against the 'semantic projection' at the end of this paper).

So if the f-structure is (23), the instantiated constructor for *knight*, and the first part of the *every* constructor will be as in (24), with polarities indicated to the extent that we have principles to determine them:

(23)
$$f\!:\begin{bmatrix} \text{SUBJ} & g\!:\begin{bmatrix} \text{QUANT} & \text{Every} \\ \text{PRED} & \text{`Knight'} \\ \text{VAR} & gv\!:[\ ] \\ \text{RESTR} & gr\!:[\ ] \end{bmatrix} \\ \text{PRED} & \text{`Sleep(SUBJ)'} \end{bmatrix}$$

(24) *knight* : $\underset{-}{gv_e} \multimap \underset{+}{} \underset{+}{gr_t}$

every : $\underset{?}{(gv_e} \multimap \underset{-}{} \underset{?}{gr_t)} \multimap \underset{+}{} \ldots \underset{+}{}$

The reason we don't know the polarities of the first two literals is that they are antecedent and consequent of a negative conditional, and the rules so far specify polarity only for those of a positive. But it is obvious what the answer has to be: to connect things up we will have to match the first and second literals of the common noun with those of the quantifier, respectively. Therefore the first quantifier literal will have to be positive and the second negative, which can be provided for by this rule:

(25) Polarity Rule 3: If an implication has negative polarity, its antecedent has positive polarity and its consequent has negative.

So now we can do the hookup and set some initial content as follows:

(26) $a.$ gv_e $\multimap$ *knight* gr_t

$b.$ $(gv_e$ $\multimap$ $gr_t)$ $\multimap$ *every* $\ldots$

But to get any further, we need some content for the postive antecedent, which is provided by Content Flow Rule:

(27) Content Flow Rule 4: The content of a positive antecedent is a new symbol, not used elsewhere in the network.

This symbol will function as a variable, to later be lambda-abstracted over.

So by CFR 4 we get:

(28) $a.$ gv_e $\multimap$ *knight* gr_t

$b.$ $(gv_e$ $\multimap$ $gr_t)$ $\multimap$ *every* $\ldots$
x

and by the two propagation rules we have so far:

(29) $a.$ gv_e $\multimap$ *knight* gr_t *knight*(x)
x

$b.$ $(gv_e$ $\multimap$ $gr_t)$ $\multimap$ *every* $\ldots$
x *knight*(x)

So now *every*'s first argument has content at both its antecedent and its consequent, so we need a rule to produce content for the whole thing. This should be something representing a function, since *every* was supposed to be a relation between functions (from entities to truth-values).

Lambda-calculus is the standard method for manufacturing functions, and the propagation rule we need includes a well-formedness criterion for hookups in its formulation:

(30) CPR 3: if ξ is the content of a positive antecedent (of a negative implication), and ϕ is the content of its (negative) consequent, then ξ must appear free in ϕ, and the content of the implication is $\lambda\xi.\phi$.

Cranking through this, what we get for the content of the first (complex) argument of the quantifier is $\lambda x.$*knight*(x). But this is in fact equivalant to just plain *knight*. That seemed like a lot of work for not much result, but there will soon be cases where the lambda can't be eliminated.

So, finally, using the rule CFR 3 for positive implications, we get *every*(*knight*) as the value of the consequent of the main implication, resulting in this (partial) assigment of content to subformulas:

(31)
$$\begin{array}{ccccccc} gv_e & \multimap & gr_t & & & & \\ x & \textit{knight} & \textit{knight}(x) & & & & \\ \uparrow & & \downarrow & & & & \\ (gv_e & \multimap & gr_t) & \multimap & \ldots & & \\ x & \textit{knight} & \textit{knight}(x) & \textit{every} & \textit{every}(\textit{knight}) & & \end{array}$$

The second argument of the quantifier is provided by the surroundings of the quantified NP, its 'scope'. The antecedent of the scope argument will deliver content (a type e variable) to the f-structure of the quantified NP, while the consequent will collect type t content back from some containing f-structure, which by CFR 5 will contain the antecedent variable free. In this simple clause, the only suitable f-structure to locate the consequent input at is f; below we will say a bit more about how the consequent location is determined. So the constructors will be as below, with the effects of CFR4 represented:

(32)
$$\begin{array}{lcccccccccc} \textit{sleep} & : & & & & & g_e & \multimap & f_t & & \\ & & & & & & & \textit{sleep} & & & \\ & & & & & & & & & & \\ \textit{knight} & : & gv_e & \multimap & gr_t & & & & & & \\ & & x & & \textit{knight}(y) & & & & & & \\ & & \uparrow & & \downarrow & & & & & & \\ \textit{every} & : & (gv_e & \multimap & gr_t) & \multimap & (g_e & \multimap & f_t) & \multimap & f_t \\ & & x & & \textit{knight}(y) & \textit{every} & y & & & \textit{every}(\textit{knight}) & \end{array}$$

Now hookup and the content flow rule through the scope argument will produce (first (restriction) argument of the quantifier here abbreviated as . . .):

(33)
$$\begin{array}{lcccccccc} \textit{sleep} & : & & & g_e & \multimap & f_t & & \\ & & & & y & \textit{sleep} & \textit{sleep}(y) & & \\ & & & & \uparrow & & \downarrow & & \\ \textit{every} & : & \ldots & \multimap & (g_e & \multimap & f_t) & \multimap & f_t \\ & & \textit{knight} & \textit{every} & y & \lambda y.(\textit{sleep}(y) & \textit{sleep}(y)) & \textit{every}(\textit{knight}) & \\ & & & & & = \textit{sleep} & & & \end{array}$$

And finally we apply CFR 3 again to produce the final result *every*(*knight*)(*sleep*).

An important question is on what basis we chose f as the location of the second argument's consequent. It turns out that for a first approximation theory of quantifier scope, one can do quite well by letting this location be specified as a variable, which can be identified with any location in the f-structure, as long as the final result is delivered back to the same place, that is, specified with the same variable. Using upper case italics for variables over f-structure, the lexical meaning-constructor for the *every* will now look like this:

(34) *every* : $((\uparrow \text{VAR})_e \multimap (\uparrow \text{RESTR})_t) \multimap (\uparrow_e \multimap H_t) \multimap H_t$

We get (33) by identifying H with f. It turns out that the restriction on hookups imposed by CFR 5 (the content of the consequent must contain the content of the antecedent free) excludes various wrong scope possibilities that tend to be generated

by naive schemes of 'Quantifier Raising', a result discussed at length in Dalrymple et al. (1999).

Here is an example where variable choice of scope-consequent and final result produces an ambiguity, the wide and narrow scope readings of *everyone seems to sleep*; to keep the structure simple the restriction argument of the quantifier is not expressed:

(35) a.
$$f:\begin{bmatrix} \text{SUBJ} & h:\begin{bmatrix}\text{PRED} & \text{'Everyone'}\end{bmatrix} \\ \text{PRED} & \text{'Seem(XCOMP)'} \\ \text{XCOMP} & g:\begin{bmatrix}\text{SUBJ} & h:\begin{bmatrix}\ \end{bmatrix} \\ \text{PRED} & \text{'Sleep(XCOMP')}\end{bmatrix}\end{bmatrix}$$

b.
$$\begin{array}{lcl} (h_e \multimap H_t) \multimap H_t & : & \textit{everyone} \\ h_e \multimap g_t & : & \textit{sleep} \\ g_t \multimap f_t & : & \textit{seem} \end{array} \qquad \begin{array}{lcl} (h_e \multimap H_t) \multimap H_t & : & \textit{everyone} \\ h_e \multimap g_t & : & \textit{sleep} \\ g_t \multimap f_t & : & \textit{seem} \end{array}$$

$$\textit{seem}(\textit{everyone}(\lambda x.\textit{sleep}(x))) \qquad \textit{everyone}(\lambda x.\textit{seem}(\textit{sleep}(x)))$$
$$H = g \qquad H = f$$

Getting the right logical forms out *seem*-structures with functional control is one of the trickier aspects of semantic interpretation in LFG; glue logic does it easily. Although an extended discussion of quantifier scope is beyond the scope of this paper, the basic mechanism is important, because it ultimately solves the 'integration problem' that provided Andrews and Manning's motivation for restriction projections.

3 Scoping Modifiers

The problem that 'scoping modifiers' present for LFG is that the rather flat f-structures that they get don't seem to provide the right guidance for semantic interpretation, which does however seem to be tightly linked to the c-structure in an obvious way (so that various other well-formalized linguistic theories have no issues with these constructions). Complex predicates pose similar issues, as discussed by Alsina (1997) and Andrews and Manning (1993, 1999). The case of scoping modifiers is simpler, so I will discuss it here.

For an example, the pair of NPs:

(36) a. a former alleged file-swapper

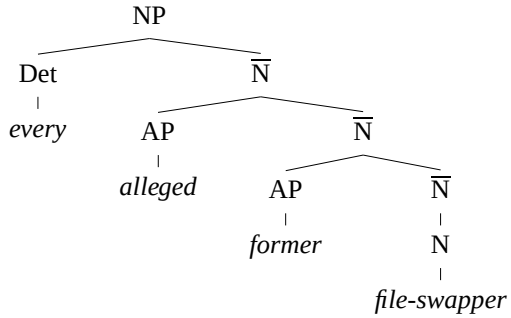
b. an alleged former file-swapper

both get an f-structure like this, where the two adjectives are members of an intrinsically unordered adjunct set:

(37)
$$\begin{bmatrix} \text{QUANT} & \text{EVERY} \\ \text{ADJUNCT} & \left\{ \begin{bmatrix} \text{PRED} & \text{`Former'} \end{bmatrix}, \begin{bmatrix} \text{PRED} & \text{`Alleged'} \end{bmatrix} \right\} \\ \text{PRED} & \text{`File-swapper'} \end{bmatrix}$$

This structure generates the expectation that both of the NPs will be ambiguous, with the same meanings, instead of differing in meaning in the way they clearly do, and which is obviously suggested by the c-structure trees they would plausibly get (Andrews 1983):

(38)



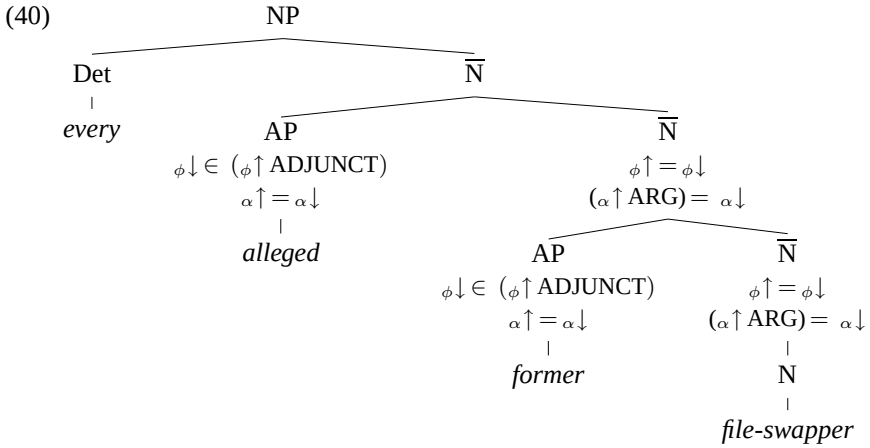
The central component of Andrews and Manning's solution to this sort of problem was an additional projection that reflected the tree-structure more closely, which we called 'a-structure' because it seems suitable to perform at least some of the functions that are often attributed to the somewhat variably defined notion of 'argument structure'. I will now show how glue logic supports an analysis whereby a-structure is retained, but using Kaplan's (1995) original 'locational' notion of projection rather than Andrews and Manning's second innovation, restriction projections.

The basic idea is to have a-structure present different headship relations than f-structure, so that the scoping modifiers are a-structurally heads, with their $\overline{\text{N}}$-sisters being 'a-structural complements' which we construed as values of an a-structure attribute ARG. The form of a-structure proposed for (38) is (39) (PRED-attributes included to make it easier to interpret the structure, although I don't believe they actually exist in a-structure, or perhaps anywhere else):

(39)
$$a{:}\begin{bmatrix} \text{PRED} & \text{`Former(ARG)'} \\ \text{ARG} & b{:}\begin{bmatrix} \text{PRED} & \text{`Alleged(ARG)'} \\ \text{ARG} & c{:}\begin{bmatrix} \text{PRED} & \text{`File-swapper'} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

We can get the f-structure (37) and the a-structure (39) from the tree (38) by annotating the tree as follows, where there are annotations specifying both f- and a- structural information, the arrows pre-subscripted with ϕ or α to indicate whether they are providing information about f- or a- structure, respectively (unannotated nodes assumed to share both f- and a-structure):

(40)



Due to the different relationship to the c-structure, these a-structures will 'respect the tree' (Alsina 1997) as required, but how will we combine their information with that from the f-structure, where grammatical relations control the relationship between NPs and semantic roles? Andrews and Manning showed how restriction projections could be used to solve this problem; here I will show that glue-logic can also do it, essentially by virtue of its capacity to shift content from one location in a structure to another.

The basic idea will be to locate the t-content of NPs at a-structure, but the e-content, which is shared through the levels, at f-structure. Therefore the uninstantiated meaning-constructor for *file-swapper* will be:

(41) *fileswapper* : $(_{\phi}\uparrow \text{VAR})_e \multimap {}_{\alpha}\uparrow_t$

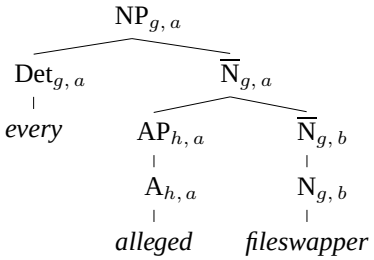
The RESTR attribute can now be abolished,[6] and the constructor of the quantifier adjusted accordingly (I so far see no reason why the quantifier's constructor would have to say what projection the scope t-information is located on, so the H variable doesn't specify this):

(42) *every* : $((_{\phi}\uparrow \text{VAR})_e \multimap {}_{\alpha}\uparrow_t) \multimap ({}_{\phi}\uparrow_e \multimap H_t) \multimap H_t$

For simple NPs such as *every fileswapper*, the analysis will work just as before, except that the noun's consequent output and quantifier's restriction's consequent input passes through an a-structure rather than an f-structure RESTR attribute. But when an operator adjective is present, things will be different. Consider a simple case with one scoping modifier, with c-structure, f-structure and a-structure as indicated below, with the c-structure nodes subscripted first with their f-structure and second with their a-structure correspondent:

[6] As can the VAR attribute, although the explanation of why one can get away with this is somewhat complex.

(43) a.



b.
$$g\!:\begin{bmatrix}\text{QUANT} & \text{EVERY}\\ \text{ADJUNCT} & \left\{h\!:\!\begin{bmatrix}\text{PRED} & \text{`Alleged'}\end{bmatrix}\right\}\\ \text{PRED} & \text{`File-swapper'}\\ \text{VAR} & gv\!:\!\begin{bmatrix}\ \end{bmatrix}\end{bmatrix}$$

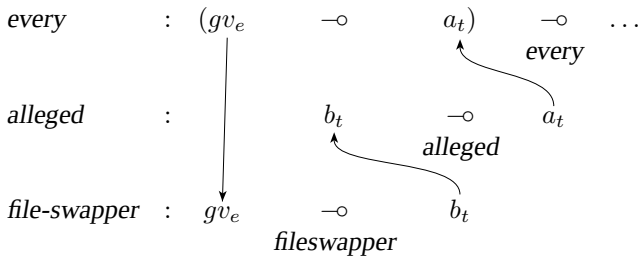
c.
$$a\!:\begin{bmatrix}\text{PRED} & \text{`Alleged(ARG)'}\\ \text{ARG} & b\!:\!\begin{bmatrix}\text{PRED} & \text{`File-Swapper(ARG)'}\end{bmatrix}\end{bmatrix}$$

Now the noun's meaning-constructor will collect as input the variable provided as output by the quantifier, because this is located at f-structure, which is the same for both the quantifier and the noun, and return its output to its a-structure, which is b. But the quantifier collects input back (for its restriction argument) from its own a-structure a, which is different from b. So without some sort of middleman, the flow of content will be blocked. The scoping adjective is of course the middleman: we can construe it as collecting t-type input from the ARG-value b of its own a-structure correspondent a, and outputting the processed result back at a; this effect will be produced by an uninstantiated meaning-constructor like this:

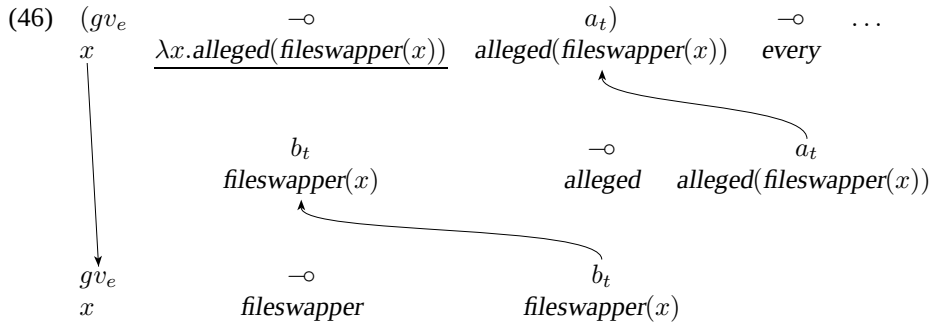
(44) *alleged* : $(_{\alpha}\uparrow \text{ARG})_t \multimap {}_{\alpha}\uparrow_t$

Hooking up the literals produced by lexical insertion and instantiation, we get:

(45)



The noun's output is hooked up to the scoping modifier's input, and the latter's output to the quantifier's first argument's input (a negative consequent), and the content then flows as indicated here, with the resulting value for the quantifier's restriction argument underlined:



So we see that the modifier gets applied to the noun-meaning in the appropriate manner.

Furthermore, it should be clear that if there is more than one scoping modifier, as in (40), the analysis will respect the tree as required, because each modifier will pass up the processed content of its $\overline{\text{N}}$-sister's a-structure to its own a-structure, and the order of application of the modifiers will be determined by the c-structural relations in the tree. Finally, the treatment of the second (scope) argument of the quantifier will solve the integration problem for which restriction projections were proposed, since the content of the quantifier's scope can be determined (at least mostly) by f-structure as discussed at the end of the previous section, independently of how the first argument (restriction) is handled.

4 Final Issues

This completes the exposition of this paper, but there are a number of further issues to be mentioned. First, Andrews (2003) extends the analysis to deal with some more complex types of scoping modifiers, such as *self-proclaimed* and *confessed*, and also regular intersective modifiers, such as *greedy* and *unscrupulous*. Some remarks are also made about developing the same kind of analysis for complex predicates. A downloadable implementation, 'Baby Glue' covers some of these issues as well.[7]

Second, there are various technical differences between the present formulation of glue logic and the currently standard one in Dalrymple (2000). A purely formal difference is that that system uses explicit universal quantification over the f-structural variables representing quantifier scope, whereas here the quantifier is omitted (one can imagine it being floated up to the top level, as is possible if the only connective used is implication (or linear conjunction)). A more substantial difference is that the inputs and outputs are normally delivered not to/from f-structure, but to a further projection off of f-structure, the 'semantic projection'. I see no function being performed by the semantic projection in the literature, and to introduce one would be inconsistent with the proposal here: to avoid restriction projections, it is essential to be able to base semantic interpretation on two different syntactically determined structures, which can't be done if they both somehow have to give rise to a single semantic projection (that kind of integration is basically what restriction projections were devised for).

[7] Available from
`http://arts.anu.edu.au/linguistics/people/averyandrews/software/`

A final issue is that here I've presented content-flow only for linear implication, whereas some glue logic analyses also use linear conjunction ($\otimes$, read 'tensor'); as discussed in Andrews (2003), content flow can be extended to $\otimes$, although it is not clear to me at this point whether $\otimes$ is truly required for natural language semantic assembly.

Bibliography

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

Andrews, A. D. 2003. Glue logic, projections, and modifiers. URL: `http://arts.anu.edu.au/linguistics/People/AveryAndrews/Papers`.

Andrews, A. D., and C. D. Manning. 1993. Information-spreading and levels of representation in LFG. Technical Report CSLI-93-176, CSLI, Stanford University, Stanford, CA. URL: `http://www-nlp.stanford.edu/ manning/papers/`.

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

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

Bresnan, J. W. 2001. *Lexical-Functional Syntax*. Oxford: Blackwell.

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. 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`.

Falk, Y. N. 2001. *Lexical-Functional Grammar: An Introduction to Parallel Constraint-Based Syntax*. Stanford University: CSLI Publications.

Fry, J. 1999. Proof nets and negative polarity licensing. In M. Dalrymple (Ed.), 91–116.

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

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

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`.