Unificational Glue

Avery D Andrews, ANU

February 2005

draft; comments welcome

In this paper I will describe a tentative proposal for using unification rather than lambda-calculus to effect semantic assembly in LFG glue logic. Some motivations for trying this out are:

(1) a. The resulting relationship between f-structures and 'logical forms' is more in accordance with the general form of correspondence-based grammatical architectures.

b. The system is inherently more restrictive (in effect, it requires (the equivalents of) higher order lambdas to be linear).

c. The system permits the formulation of some plausible monotonicity constraints which aren't straightforwardly formulable in the standard lambda-based system.

None of this constitutes anything like an argument that the idea is right, but I think it does show that it is worth a bit of exploration. A side-effect is that it highlights some subtle unclarities in the interpretation of glue-logic, as I'll discuss shortly.

The technique I will employ is to have the meaning-constructors consist of a unifiable structure as their meaning-side, and, on the glue-side, a conventional glue-term supplemented with indexes to substructures of the meaning-side. When constructors combine, indexes will be identified, forcing unifications of the meaning-side structures to produce the assembled result. This is similar to the standard technique for faking beta-reduction with unification in Prolog, and suffers from the same limitations. Whether this is an empirical drawback or virtue is presently unclear, but it does constitute a difference from the standard lambda-based technique.

I will present the approach using both proof-nets (Fry 1999, Andrews 2004) and natural deduction (Asudeh 2004), and will assume a basic familiarity with the workings of both approaches. A significant issue is that unificational glue makes no sense at all unless one accepts a level of 'logical form', a level at which quantifier and related scope ambiguities are structurally represented. Standard lambda-based glue does not require such a commitment:

we can regard the assembly of meaning-constructors to be merely a technique for picking a model-theoretic interpretation of the f-structure. But the use of unification for assembly implies that the things being assembled are structures, that is, inherently syntactic entities, and the approach makes no sense unless these are thought of as being psychologically realized (which I would take to mean that they are putative representations of mental structures and/or events, rather than the other way around, as some philosophers of linguistics seem to sometimes assume).

1 Predicates and Arguments

We introduce the basics by showing how simple structures of predicates and arguments work. The essential ideas can be illustrated by the meaning-constructors for the noun *Fido* and the intransitive verb *barks*, for a structure where the subject's f-structure is g, the entire f-structure f:

(2) $$[\mathit{Fido}]_{\alpha} \;:\; \alpha_{g,e}$$

$$[\mathit{Bark}([\ ]_{\beta})]_{\gamma} \;:\; \beta_{g,e} \multimap \gamma_{f,t}$$

The meaning-side format is 'semantic structures', which you can think of as convenient abbreviations for some system of feature-structures expressing meaning (such as for example those of Jackendoff), where some of the substructures are tagged with Greek letters, written as subscripts to square brackettings, some of which are empty (these are tyically unification sites). You can think of the brackets as reminders that these things are supposed to have a decomposition into features, supporting useful unifications.

The glue-side format is the same as in standard (new-style, as introduced in Dalrymple et al. (1999), and also presented in Dalrymple (2001) and other recent work), except that the 'literals' (basic expressions) contain Greek letter tags indexing (sub-)structures of the semantic structure as well as the usual f-structure location and semantic type information. It is convenient to vacillate as to whether the tags or the f-structures are treated as the main item or a subscript: if we are primarily interested in the results of the unifications, the former is more convenient (as in (2), while if we're interested in how the syntax controls assembly, the latter is.

There is a significant issue about the significance of the type information and how it should be represented. The type information is seen as

necessary for technical reasons, to block certain bad readings that would otherwise be produced [GET PRINT REFERENCE FOR THIS]. But this raises the question of whether it is a strictly technical device, or has some more general significance, such as implementing a form of semantic filtering. This intrinsically desireable property would require that the types match linguistically plausible ontological categories for the meanings of linguistic expressions, something which seems plausible, but has not been worked on in any systematic way, let alone actually worked out.

If the types are ontological categories, then we expect that these types will be reflected in some manner in the structure of the meaning-sides (perhaps by means of some kind of typing of the feature structures, or perhaps by means of attributes constraining possible unifications), which means that we shouldn't have to redundantly represent the typing information on the glue-side, unless we want to do this in order to have a place where all the information is represented that indicates whether an acceptable semantic composition exists, but doesn't say what the resulting meaning is. The upshot of all this is that we will often omit type information from the glue-side.

The proof-net presentation of glue goes particularly nicely with the use of unification for assembly, since the addition of links to the developing net can be seen to drive unification of substructures of the meaning-side, in a way somewhat analogous to how establishing c-structure relations drives unification of portions of f-structure. Proof-nets work by implementing semantic assembly as a process of connecting the glue-side literals (atomic formulas, consisting of location, tag and perhaps type information) with arrows, subject to certain well-formedness constraints for the resulting hookup.[1] We'll present these shortly; the simplest is that the type and location information of connected literals must match (the matching of type can be taken as an aspect of a requirement that the result of assembling the meanings make sense), and for the constructors of (2) there is only one hookup that satisfies this condition:

(3)
$$\begin{array}{lcl} [\mathit{Fido}]_{\alpha} & : & \alpha_{g,e} \\ & & \downarrow \\ [\mathit{Bark}([\ ]_{\beta})]_{\gamma} & : & \beta_{g,e} \multimap \gamma_{f,t} \end{array}$$

[1] Fry (1999) presents the original version of these constraints, Andrews (2004) a possibly easier formulation due to de Groote (1999) and Perrier (1999). An unfortunate feature of the unificational approach is that the semantic content-based version of the correctness criterion formulated by Perrier and used by Andrews won't work for unificationally-processed contents, so we have to use one of the others.

(The directionality of the arrow will be explained shortly). Staring at the picture, it is hopefully a reasonably obvious idea that we are supposed to interpret the connection from $\alpha_{g,e}$ to $\beta_{g,e}$ as an instruction to identify, that is unify, the semantic (sub-)structures tagged α and β. Doing this transforms the structure to (surviving index chosen arbitrarily):

$$\begin{array}{lcl} [\mathit{Fido}]_\beta & : & \beta_{g,e} \\ & & \downarrow \\ [\mathit{Bark}([\mathit{Fido}]_\beta)]_\gamma & : & \beta_{g,e} \multimap \gamma_{f,t} \end{array} \quad (4)$$

and identifies the final output γ with an appropriate assembled structure. We assume a constraint that the final consequent of the meaning-constructor must be coindexed with the entire meaning-side rather than a substructure thereof.

In a natural deduction implementation, this assembly is accomplished by means of the rule of modus ponens, or 'implication elimination' ($\multimap_{elim}$), by means of a deduction that looks like this:

$$\frac{[\mathit{Fido}]_\alpha : \alpha_{g,e} \qquad [\mathit{Bark}([\ ]_\beta)]_\gamma : \beta_{g,e} \multimap \gamma_{f,t}}{[\mathit{Bark}([\mathit{Fido}]_\beta)]_\gamma : \gamma_{f,t}} \multimap_{elim} \quad (5)$$

To effect the unification on the meaning-side, we suppose that in order to apply $\multimap_{elim}$, we need to find an identifcation (equalizer) for the tags that makes the premise on the left identical to the antecedent of the premise on the left; the meaning-side of the result is the the result of unifying the two 'indexed' f-structures (an indexed f-structure being a function from a set of indexes into a feature-structure; indexed f-structures are partially ordered in an obvious way by a subsumption relation, and the unification of two indexed f-structures is the minimal indexed f-structure, if any, subsumed by both).

To formulate the general rule, we need a bit of notation; we'll say that Φ^A is an f-structure indexed by the set A, and that if ι is a mapping from one index set A to index set B, and $C \subseteq A$, then $\iota(\Phi^C : \mathcal{F})$ is what you get from (the meaning-constructor) $\Phi^C : \mathcal{F}$ by replacing each tag in C, on both the meaning and glue-sides, with its image under ι. Now if $\multimap_{elim}$ is going to apply to the meaning-constructors $\Phi^A : \mathcal{A}$ and $\Psi^B : \mathcal{B} \multimap C$, there must be an equalizer $<\iota, \kappa>$ mapping A and B, respectively, onto some index-set C, such that $\iota(\mathcal{A}) = \kappa(\mathcal{B})$, and we can say that the meaning-side of the result is then $\iota(\Phi^A) \sqcup \kappa(\Psi^B)$. Schematically:

(6) $$\frac{\Phi^A : \mathcal{A} \qquad \Psi^B : \mathcal{B} \multimap \mathcal{C}}{\iota(\Phi^A) \sqcup \kappa(\Psi^B) : \kappa(\mathcal{C})} \multimap_{elim}$$

where $\iota(\mathcal{A}) = \kappa(\mathcal{B})$

2 Second Order Arguments

A considerably more sophisticated application for glue (and a generally harder problem for semantic interpretation of LFG and other syntactic theories) is quantification. Intuitively, the issue is how to get the desired readings of sentences such as:

(7) Everybody didn't yell

out of f-structures such as:

(8) $$f:\begin{bmatrix} \text{SUBJ} & g:\begin{bmatrix} \text{PRED} & \text{`Person'} \\ \text{QUANT} & \text{EVERY} \end{bmatrix} \\ \text{PRED} & \text{`Yell(SUBJ)'} \\ \text{POL} & \text{NEG} \\ \text{TENSE} & \text{PAST} \end{bmatrix}$$

The problem is similar in spirit to head-switching, because we want the arguably syntactically embedded negation and quantifier to function more like syntactic heads, but it is more complex, due to the ambiguities as indicated in these fairly conventional 'logical forms':

(9) a. $\mathit{Every}(x, \mathit{Person}(x), \neg \mathit{Yell}(x))$

b. $\neg \mathit{Every}(x, \mathit{Person}(x), \mathit{Yell}(x))$

and the intricacies of quantifier-scope. A major original selling point of glue logic, for example, was its ability to correctly deliver the five available readings of:

(10) Every representative of a company exhibited a product

rather than producing six, as tends to happen with naive 'Quantifier extraction' algorithms (Dalrymple et al. 1997).

One of the many difficult issues posed by quantification is a choice between what might be called an 'Aristotelian' versus a 'Fregean' representation:

(11) a. Aristotelian: $\mathit{Every}(\mathit{Dog}, \mathit{Bark})$

b. Fregean: $\mathit{Every}(x, \mathit{Dog}(x), \mathit{Bark}(x))$

In the Aristotelian approach, the quantifier (or, more precisely, quantificational determiner) represents a relation between (the extensions of) two predicates, while in the Fregean approach, it binds a variable which appears in two propositions. Whatever the relative empirical merits of the two techniques, it is an interesting feature of unificational glue that it strongly motivates (although doesn't actually require) a Fregean approach. The reason is that the Aristotelian approach requires the use of lambda-abstraction in order to represent the quantification of non-subjects (or, more generally, quantification over more than one grammatical relation of the clause); for example something like (12) for *Fido likes everybody*:

(12) $\mathit{Every}(\mathit{Person}, \lambda x.\mathit{Like}(\mathit{Fido}, x))$

But making extensive use of lambda's in the meaning-representations undercuts the motivation for trying to avoid them in the treatment of semantic composition.

Furthermore, the Aristotelian treatment seems to lead to an unattactive complexity in the present unificational approach, which are avoided in the Fregean. To see this, consider first a preliminary Fregean account of the quantifier *everybody*. We'll assume a glue-term of the same form as would be used in regular glue, to be combined with an intransitive predication. The proof-net hookup will be as below, where the antecedent of the quantifiers first argument as ?? as its tag, because we haven't settled how the for quantifiers are supposed to work:

(13) $[\mathit{Every}(x, [\mathit{Person}(x)], [\ ]_\alpha)]_\beta \ : \ (??_{g,e} \multimap \alpha_{H,t}) \multimap \beta_{H,t}$

$[\mathit{Yell}([\ ]_\gamma)]_\delta \ : \ \gamma_{g,e} \multimap \delta_{f,t}$

The unification of α and δ is obviously appropriate, and we also need to unify γ with something, and the obvious choice is a tag connected to the quantified variable, so that our revised structure will be:

(14) $[\mathit{Every}(x_\epsilon, [\mathit{Person}(x)], [\ ]_\alpha)]_\beta \ : \ (\epsilon_{g,e} \multimap \alpha_{H,t}) \multimap \beta_{H,t}$

$[\mathit{Yell}([\ ]_\gamma)]_\delta \ : \ \gamma_{g,e} \multimap \delta_{f,t}$

This will unify out to yield

(15) $[\mathit{Every}(x, [\mathit{Person}(x)], [\mathit{Yell}(x)])]$

as final output, which seems appropriate.

But at this point we've made a significant divergence from standard lambda-based glue. On that account, in the content-flow interpretation of Andrews (2004), the $\epsilon_{g,e}$ position in the glue term would be associated with a novel free variable as content, say, Z, not appearing elsewhere in the structure. Via the rules of content-propagation, this would result in $\alpha_{H,t}$ getting the content $\mathit{Yell}(Y)$, whence, via the lambda-abstraction content rule for negative implications, $(\epsilon_{g,e} \multimap \alpha_{H,t})$ gets $\lambda Y.\mathit{Yell}(Y)$, equivalent by η-reduction to plain old *Yell* (the square brackets would be absent from a conventional glue treatment). The deductive version is essentially the same, with Implication Elimination triggering function application, Implication Introduction lambda-abstraction.

Standard glue is thus quite compatible with an Aristotelian treatment, which can use for *everybody* a meaning-constructor like this:

(16) $\lambda P.\mathit{Every}(\mathit{Person}, P) \;:\; (g_e \multimap H_t) \multimap H_t$

We can also implement the Fregean, like this:

(17) $\lambda P.\mathit{Every}(x, \mathit{Person}(x), P(x)) \;:\; (g_e \multimap H_t) \multimap H_t$

But for unificational glue, things aren't so simple. If we try to use a meaning-constructor such as (16), the problem arises of what content to provide to the first literal, and how to integrate it into the final result.

This is not an issue in lambda-based glue, because, as it were, the network of connections itself, or the structure of the deduction, has a constructive capacity to build a lambda-term, effectively using the variable that it itself has introduced at an earlier stage (corresponding to the proof-steps of introducing and discharging an assumption). But in the unificational approach, all the network can do is prescribe patterns of identification for pre-existing structures, so the only way I can think of to get an Aristotelian treatment is to introduce the lambda-abstraction into the meaning-constructor itself, producing something like:

(18) $[\mathit{Every}([\mathit{Person}], \lambda x_\epsilon.[\ ]_\alpha)]_\beta \;:\; (\epsilon_{g,e} \multimap \alpha_{H,t}) \multimap \beta_{H,t}$

This analysis isn't clearly wrong, but it is however not so attractive on grounds of simplicity and elegance, especially for quantificational determiners, which would always require two lambda-abstracts, one of which (for the nominal) would always be vacuous (eliminable by η reduction).

The entire discussion raises a further issue for standard glue of whether we want to think of the assembly notations provided by glue (lambda-abstraction and function-application) as part of the meaning-language, or as a different language, used to provide assembly instructions for expressions in the meaning-language. Either interpretation is posssible, but in unificational glue, the question doesn't arise: assembly expressions are tags which are clearly extraneous to the meaning-language *per se*, forcing unifications when they get equated, producing the ultimate meaning-expressions.

It is worth noting that quantification is where the unificational contents become unable to support a content-flow-based correctness criterion. In the criteria developed by de Groote (1999) and Perrier (1999), the antecedents of negative (in LFG polarity terms, positive in theirs) implications receive as content an novel symbol, corresponding to the introduction of a provisional hypothesis, which is removed by division (deGroote) or lambda-abstraction (Perrier) at the content-assignment step that corresponds to the discharge of the provisional hypothesis. In the unificational version, there is nothing representing discharge, and therefore no way to model this feature of a well-formed deduction. Therefore, in order to detect well-formed proofnets, we need to use one of the other correctness criteria; when we get to tensors, we'll see that if we want to use some kind of content-flow, it will have to be deGroote's rather than Perrier's.

In a deductive formulation, some things are a bit different. In the first place, *everybody yelled* can be done by a single application of $\multimap_{intr}$, forestalling most of the previous discussion.[2] But we can get the complexity back by introducing a negation operator that will participate in a scope ambiguity, as in for example the wide-scope reading of *everybody* in *everybody didn't yell*. The overall form of a deduction for this will be:

[2]We could get the equivalence to this with proof-nets if we allowed identical subformulas of opposite polarities to be directly connected by axiom links, assuming that this possibility precluded connecting any of the proper subformulas of the subformulas so-connected.

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{[g]^1 \qquad g \multimap f}{f}\multimap_{elim}
\qquad\qquad f \multimap f
}{f}\multimap_{elim}
}{g \multimap f}\multimap_{intr,1}
\qquad\qquad (g \multimap f) \multimap f
}{f}\multimap_{elim}
\tag{19}
$$

To build the meanings with unificational glue, we have to figure out what the meaning should be for the introduced hypothesis $[g]^i$, and what the meaning-operation ought to be for $\multimap_{elim}$. The appropriate answers would appear to be, respectively, an empty f-structure, and no change (however a question about types arises, which we will consider shortly).

With contents added, the tree above $\multimap_{intr}$ is, with some arbitrary decisions being made about the tags:

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{[[\ ]_\alpha : g_\alpha]^1 \qquad [\mathit{Yell}([\ ]_\beta)]_\gamma : g_\beta \multimap f_\gamma}{[\mathit{Yell}([\ ]_\beta)]_\gamma : f_\gamma}\multimap_{elim}
\qquad\qquad [\mathit{Not}([\ ]_\delta)]_\epsilon : f_\delta \multimap f_\epsilon
}{[\mathit{Not}([\mathit{Yell}([\ ]_\beta)]_\delta)]_\epsilon : f_\epsilon}\multimap_{elim}
}{[\mathit{Not}([\mathit{Yell}([\ ]_\beta)]_\delta)]_\epsilon : g_\beta \multimap f_\epsilon}\multimap_{intr,1}
}{[\mathit{Not}([\mathit{Yell}([\ ]_\beta)]_\delta)]_\epsilon : f_\epsilon}\multimap_{elim}
\tag{20}
$$

The arbitrariness of the decisions resides in that fact that nothing fixes a choice of tag-equalizers for the $\multimap_{elim}$ steps; it doesn't matter which ones we choose (as long as they are minimally aggressive ones that fit the left premise to the antecedent of the right premise). We'll also see in a moment that we have to keep track of them, or at least the ones applied to the left premises, so our $\multimap_{elim}$ notations in fact need to be tagged with the equalizers used.

As becomes apparent when we consider what is involved in applying the $\multimap_{intr}$ step (continuing (20), repeating its conclusion):

$$
\dfrac{[\mathit{Not}([\mathit{Yell}([\ ]_\beta)]_\delta)]_\epsilon : f_\epsilon}{[\mathit{Not}([\mathit{Yell}([\ ]_\beta)]_\delta)]_\epsilon : g_\beta \multimap f_\epsilon}\multimap_{intr,1}
\tag{21}
$$

The problem is how are we supposed to know that the introduced antecedent should have the tag β? The answer is by tracking the effects of the equalizers used by the applications of $\multimap_{elim}$ on the original tag α; applying them in succession we get β, so that's what we use for the tag.

The derivation finishes with a relatively spectacular instance of $\multimap_{elim}$, where Φ abbreviates the conclusion of (21):

$$(22)\quad \frac{\Phi : g_\beta \multimap f_\epsilon \qquad [Every(x_\zeta, Person(x), [\ ]_\eta)]_\theta : (g_\zeta \multimap f_\eta) \multimap f_\theta}{[Every(x_\zeta, Person(x), [Not([Yell([\ ]_\beta)]_\eta)]_\theta : f_\theta} \multimap_{elim}$$

Note that the equalizer used here is $<\{\beta \mapsto \beta, \epsilon \mapsto \eta\}, \{\zeta \mapsto \beta, \eta \mapsto \eta, \theta \mapsto \theta\}>$.

We can reduce the bookkeeping needed for $\multimap_{intr}$ by treating the unifications induced by $\multimap_{elim}$ as destructively applying to the entire deduction; then the tag of the introduced antecedent will be the same as that used in the hypothesis being discharged. This seems to be somewhat contrary to the 'spirit of deduction', where one would tend to think of hypotheses as being unaltered by use, but won't cause any trouble in a linear system, where they only get used once.[3]

So that's how the system is supposed to work for purely implicational meaning-constructors. I will now make two observations about it. The first is that that consistently with the expectations of parallel correspondence architecture, we get a correspondence between f-structure and the semantic structures produced in assembly. This will not in general be 1-1 from f-structure to semantic structure (since for example negatives and quantifiers involve multiple semantic levels corresponding to one syntactic level), but there is a prospect that it might be 1-1 in the other direction (although nothing we have said guarantees this). The second is that there is a sense in which it is sufficient to effect basic assembly, even if it can't deliver all correct analyses (an empirical matter, which we will be considering one aspect of later). That is, suppose we have a conventional glue analysis G in which all meaning-sides are constants (of appropriate types). All lambdas will then be linear (binding one and only one variable), and we can therefore produce a unificational glue analysis U whose outputs will be good representations of the meaning-terms produced by G. For example, if G has $Everybody^{(e\to t)\to t}$ as the meaning-side of the constructor for *everybody*, then the U will have the following constructor:

$$(23)\quad [Everybody(\lambda X_\alpha.[\ ]_\beta)]_\gamma \ : \ (\uparrow_\alpha \multimap H_\beta) \multimap H_\gamma$$

The linearity of the lambda-abstracts produced by glue assembly means that we're not going to get into a situation where, in effect, a given higher order argument A needs to be applied to more than one argument, which creates difficulties for unification.

[3]As long as 'or course', which copies resources, is absent, but this isn't used in glue.

3 A Monotonicity Constraint

Unificational Glue is thus workable at least at a minimal level, but it can be used to formulate a plausible monotonicity constraint that provides some positive evidence for it. This comes from the properties of certain 'scoping modifiers', such as *confessed* and *self-proclaimed*. Andrews (2003) suggests a glue analyses of these, whereby they are in effect modifiers of properties. In the standard glue treatment of quantified nominals (that is, a quantificational determiner combined with a nominal such as *dog*), the nominal is treated as being of glue type $gv_e \multimap gr_t$, where gv is the value of an attribute VAR, gr of RESTR (both standardly located on a 'semantic projection' which Andrews (2004) argues should be dispensed with, in favor of locating them respectively on the f-structure and a-structure projection of Andrews and Manning 1999).

In the present context, the motivation for using multiple projections is an irrelevant distraction, so we will ignore them and just use VAR and RESTR attributes in the f-structure. With these decisions, conventional Fregean constructors for *dog* and *every* would be:

(24) $$\lambda x.Dog(x) \quad : \quad gv_e \multimap gr_t$$

$$\lambda P.\lambda Q.Every(x, P(x), Q(x)) \quad : \quad (gv_e \multimap gr_t) \multimap (g_e \multimap H_t) \multimap H_t$$

These assemble to give a constructor with form equivalent to that of *everyone*, for the quantified NP *every dog*.

Now we can extend this to deal with a modifier such as *confessed* in complex nominals such as *confessed file-swapper*, by treating these modifiers as being of glue type $(gv_e \multimap gr_t) \multimap gv_e \multimap gr_t$. The idea is that from a predicate meaning P, we derive one meaning $\lambda P.\lambda x.Perf(confessed(x, P(x)))$. This constructor will do the job:

(25) $\lambda P.\lambda x.Perf(confessed(x, P(x))) \quad : \quad (gv_e \multimap gr_t) \multimap gv_e \multimap gr_t$

As shown in Andrews (2003), this will lead to OK results, but it also suffers from a theoretical problem of underconstrainedness.

This problem is that there is nothing to stop us from writing a meaning-constructor for a hypothetical adjective **alleger* such that *an alleger fileswapper* is somebody who makes allegations that other people are fileswappers:

(26) $\lambda P.\lambda x.Allege(x, Some(y, Person(y), P(y)))$

Intuitively, at least in terms of guiding intuitions of Andrews and Manning (1999), this kind of lexical meaning for an attributive adjective ought to be impossible because an NP has a single 'reference index', shared between all its levels, indicating to what entity any properties are to be ascribed. But technically, the result doesn't come through in standard glue, because each nominal *fileswapper, alleged/*er fileswapper* corresponds to a distinct item of semantic type $e \rightarrow t$, with nothing to enforce any kind of identity between the two e's.

But with unificational glue, things are different. The Fregean constructor for the quantificational determiner *every* will be:

(27) $[\mathit{Every}(X_\alpha, [\ ]_\beta, [\ ]_\gamma)]_\delta \ : \ (\alpha_{gv,e} \multimap \beta_{gr,t}) \multimap (\alpha_{g,e} \multimap \gamma_{H,t}) \multimap \delta_{H,t}$

Observe that the treatment of the restriction information is that same as that of the scope, other than that the f-structural locations are different.

Now turning to a possible 'head-embedding' modifier such as *confessed*, in the absence of lambda's, we have to use structure-sharing to express the idea that the confessor is the committer of the crime confessed. The glue term will however be essentially the same as in the standard treatment, except for the tags:

(28) $[\mathit{Perf}([\mathit{Confess}([\ ]_\alpha, [\ ]_\beta)])]_\gamma \ : \ (\alpha_{gv,e} \multimap \beta_{gr,t}) \multimap \alpha_{gv,e} \multimap \gamma_{gr,t}$

How will this fit together with constructors for the quantifier and a nominal? A typical collection would be:

(29) $[\mathit{Every}(X_\alpha, [\ ]_\beta, [\ ]_\gamma)]_\delta \ : \ (\alpha_{gv,e} \multimap \beta_{gr,t}) \multimap (\alpha_{g,e} \multimap \gamma_{H,t}) \multimap \delta_{H,t}$

$[\mathit{Perf}([\mathit{Confess}([\ ]_\epsilon, [\ ]_\zeta)])]_\eta \ : \ (\epsilon_{gv,e} \multimap \zeta_{gr,t}) \multimap \epsilon_{gv,e} \multimap \eta_{gr,t}$

$[\mathit{Fileswapper}([\ ]_\theta)]_\iota \ : \ \theta_{gv,e} \multimap \iota_{gr,t}$

(Our list of Greek letters grows thin.) Now the rules for valid hookups imply that the content of a positive antecedent of a negative conditional must contribute to the content delivered to its negative consequence. The result is that the only well-formed hookup of (29) is:

(30) $[\mathit{Every}(X_\alpha, [\ ]_\beta, [\ ]_\gamma)]_\delta \ : \ (\alpha_{gv,e} \multimap \beta_{gr,t}) \multimap (\alpha_{g,e} \multimap \gamma_{H,t}) \multimap \delta_{H,t}$

$[\mathit{Perf}([\mathit{Confess}([\ ]_\epsilon, [\ ]_\zeta)])]_\eta \ : \ (\epsilon_{gv,e} \multimap \zeta_{gr,t}) \multimap \epsilon_{gv,e} \multimap \eta_{gr,t}$

$[\mathit{Fileswapper}([\ ]_\theta)]_\iota \ : \ \theta_{gv,e} \multimap \iota_{gr,t}$

You should be able to work out that this induces the unifications that we need.

Now consider the prospects for an adjective like our impossible **alleger*. It looks like we could do it with a constructor like this:

(31) $[\mathit{Allege}([\ ]_\alpha, [\mathit{Some}(x_\delta, \mathit{Person}(x), [\ ]_\beta)])]_\gamma \ : \ (\alpha_{gv,e} \multimap \beta_{gr,t}) \multimap \delta_{gv,e} \multimap \gamma_{gr,t}$

However a hookup done this way will have a property we can balk at, namely the assignment of two different contents to the *gv* location, namely, the variables introduced by the quantifier and by **alleger*. In standard glue there's no basis for complaint, since the variables introduced at the two positions will be different anyway, and only ultimately rendered effectively the same via the pattern of lambda-application on the meaning-side (which wouldn't be expected to be relevant to the syntax). But in the unificational version of glue, we can consider imposing a requirement that multiple contents associated with one location bear some relationship to each other. Identity is one possibility, but is too strong, since it would rule out the accounts of quantifier scope and similar phenomena. But a weaker 'monotonicity' constraint (Asudeh p.c.) seems sustainable, which says that if two contents are associated with one f-structure location, one of them must contain the other as a substructure. If quantified variables contain nothing but themselves, as seems reasonable, a requirement that the variables associated with the VAR attribute be unique will follow.

It is perhaps worth showing that the standard glue analysis of intersective adjectives is consistent with the monontonicity constraint. In Dalrymple (2001), these are given two meaning constructors, one of them rather complex. These are (a) a constructor to express the basic meaning of the adjective, as a one-place predicate (or more places, for adjectives that take additinal argments) (b) a constructor to integrate the meaning of the adjective phrase with that of the rest of the NP, shown to be necessary by Kasper (1995) to treat adverbial modification of adjectives.

The latter constructor for an adjective such as *Danish* can be plausibly written as follows (uninstantiated):

(32) $[\mathit{Danish}[\ ]_\alpha]_\beta \ : \ ((\mathrm{ADJUNCT}\uparrow)\,\mathrm{VAR})_{\alpha,e} \multimap \uparrow_{\beta,t}$

The 'compositional' constructor is treated by Dalrymple as introduced by the lexical entries of adjectives, but employing the proposal of Asudeh and

Crouch (2002) to allow PS rules as well as lexical entries to introduce meaning-constructors, we could represent it as follows (associated with the AP in the PS rule for attributive modifiers):[4]

$$(33)\quad [[\ ]_\gamma \textit{ and } [\ ]_\delta]_\epsilon \ : \ ((\uparrow \mathrm{VAR})_{\zeta,e} \multimap \downarrow_{\gamma,t}) \multimap ((\uparrow \mathrm{VAR})_{\zeta,e} \multimap (\uparrow \mathrm{RESTR})_{\delta,t}) \multimap (\uparrow \mathrm{VAR})_{\zeta,e} \multimap (\uparrow \mathrm{RESTR})_{\epsilon,t}$$

The meaning-side is relatively intelligible: γ represents the proposition expressed by the adjective, δ that of the head noun (or as much of the nominal as has been processed so far), ϵ the entire result. The glue-side is more complex, but one can see that it consists of two argument implications yielding as output a final implication (whose parentheses are omitted by the right-associativity convention), all with antededent $(\uparrow \mathrm{VAR})_{\zeta,e}$, expressing the idea that all the predications are to apply to the same entity. The consequents of these implications are co-tagged to the γ, δ and ϵ substructures of the meaning-side in the appropriate manner. Now suppose we combine these constructors with one for a noun, such as *Bachelor*. The constructors are too long to display both the meaning and glue-sides, but we can display suitably instantiated and hooked-up glue-sides as follows:

$$(34)\quad \begin{array}{lll} Dc & : & (\zeta_{gv,e} \multimap \gamma_{h,t}) \multimap (\zeta_{gv,e} \multimap \delta_{gr,t}) \multimap \zeta_{gv,e} \multimap \epsilon_{gr,t} \\ Dl & : & \alpha_{gv,e} \multimap \beta_{gr,t} \\ b & : & \iota_{gv,e} \multimap \kappa_{gr,t} \end{array}$$

Unlike the quantificational constructor, the compositional adjective constructor doesn't specify any substructure of its meaning-side to serve as content for its positive antecedent e-terms; rather this is supplied from the negative antecedent ζ, which has to collect content from elsewhere (a quantificational determiner). This shows that we don't want a requirement that each tag on the glue-side of a meaning-constructor tag a substructure on the meaning-side, although we will later discuss a restriction that each glue-side tag pick out a substructure of the final assembled meaning.

And we see that the *gv* attribute is only every related to one thing as semantic correspondent, the (meaning-language) variable introduced and bound by the quantifier of the NP.

[4]If this were put in the lexical entries of the adjectives, we'd need to replace references to $\uparrow$ with references to $(\mathrm{ADJUNCT}\uparrow)$, and $\downarrow$ with $\uparrow$.

4 Tensors and Anaphora

Although predicates and arguments can be dealt with using the implicational fragment, the treatment of pronouns appears to require more, in particular, the introduction of 'tensors' (symbolized $\otimes$), which effect a sort of 'splitting' of resources. It is a rather intriguing fact that tensors only seem to be required for anaphora, but, unfortunately, we can't plausibly compartmentalize them out of glue, because of the way anaphora interacts with quantifiers. Here I will look at the treatment of anaphora and 'resumptive pronouns' provided Asudeh (2004).[5]

Asudeh's basic syntactic treatment of pronouns gives them an antecedent appearing as value of a grammatical function ANTECEDENT.[6] What the pronoun basically does is make a copy of the antecedent's resource, return one of the instances to that location, and the other to its own location. Asudeh treats ANTECEDENT as an attribute of the semantic projection, which we're not using here, so we will have it as an attribute of the f-structure.

So a sentence such as *John thinks that Mary likes him* will get an f-structure like this:

(35)
$$f:\begin{bmatrix} \text{SUBJ} & g:\begin{bmatrix}\text{PRED} & \text{'John'}\end{bmatrix} \\ \text{PRED} & \text{'Think'} \\ \text{COMP} & h:\begin{bmatrix} \text{SUBJ} & i:\begin{bmatrix}\text{PRED} & \text{'Mary'}\end{bmatrix} \\ \text{PRED} & \text{'Like'} \\ \text{OBJ} & j:\begin{bmatrix}\text{PRED} & \text{'Pro'} \\ \text{ANTECEDENT} & g:[\ ]\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

In the standard treatment, the instantiated glue sides of the constructors for *John* and *him* will be:

[5] There is also an analysis of anaphora in Dalrymple (2001), which is more comprehesive, but employs what appear to be some not very fully explained innovations, such as bracketted list terms.

[6] This raises the question of how to deal with antecedents that are split (*every professor$_i$ told a colleague$_j$ that they$_{i\oplus j}$ would edit a journal*), or present only in the common ground of the discourse rather than a recent utterance (*he's resigned*, spoken on the evening of August 8, 1974). I won't consider these issues here.

(36) g_e

$g_e \multimap (g_e \otimes j_e)$

The '$\otimes$' symbol is linear logic *and*, read 'tensor', and it's effect is to produce or consume a pair of resources. The polarity rules for $\otimes$ are:

(37) a. The components of a positive $\otimes$-combination are positive

b. The components of a negative $\otimes$-combination are negative

The effect is that if we hook up like this:

(38) g_e

$\downarrow$

$g_e \multimap (g_e \otimes j_e)$

we get two unlinked positives, one at the location of the antecedent (g), the other at the location of the pronoun (j). This has the effect of being a 'splitter' for a resource, making it available at two places rather than the expected one.

For the actual semantics, we want the contents at these places to be the same, and the obvious way to acheive this in the unificational approach is to have the meaning side of the pronoun be a single structure coindexed with all of the literals on the glue-side (note that it can carry type information for distinguising different ontological types of pronouns). However it might be rash to presume that this will work for all applications of tensors, so we will use a pair, and get the identification effect by co-tagging the two pair components, indicating that they are one thing linked into the structure in two ways. So the full meaning-constructors will be:

(39) $[John]_\alpha \;:\; g_{\alpha,e}$

$<[\ ]_\beta, [\ ]_\beta> \;:\; g_{\beta,e} \multimap (g_{\beta,e} \otimes j_{\beta,e})$

The effect of the linkage of (38) will then be to identify α and β, making the content *John* available at both the antecedent and both components of the consequent locations.

As a result, here are instantiated meaning-constructors for (35):

(40)
$$\begin{array}{rcl}
[John]_\gamma & : & g_{\gamma,e} \\
[Think([\ ]_\delta, [\ ]_\epsilon)]_\zeta & : & h_{\epsilon,t} \multimap g_{\delta,e} \multimap f_{\zeta,t} \\
<[\ ]_\lambda, [\ ]_\lambda> & : & g_{\lambda,e} \multimap (g_{\lambda,e} \otimes j_{\lambda,e}) \\
[Like([\ ]_\theta, [\ ]_\iota)]_\kappa & : & j_{\iota,e} \multimap i_{\theta,e} \multimap h_{\kappa,t} \\
[Mary]_\eta & : & i_{\eta,e}
\end{array}$$

This hookup will produce the desired structure (unifications left as an exercise):

(41)
$$\begin{array}{rcl}
[John]_\gamma & : & g_{\gamma,e} \\
[Think([\ ]_\delta, [\ ]_\epsilon)]_\zeta & : & h_{\epsilon,t} \multimap g_{\delta,e} \multimap f_{\zeta,t} \\
[\ ]_\lambda & : & g_{\lambda,e} \multimap (g_{\lambda,e} \otimes j_{\lambda,e}) \\
[Like([\ ]_\theta, [\ ]_\iota)]_\kappa & : & j_{\iota,e} \multimap i_{\theta,e} \multimap h_{\kappa,t} \\
[Mary]_\eta & : & i_{\eta,e}
\end{array}$$

The same basic treatment will work for quantifiers; here are the constructors for *nobody thinks that Mary likes them* (using the singular, gender-unspecified sense of *they*):

(42)
$$\begin{array}{rcl}
[No(x_\alpha, Person(x), [\ ]_\beta)]_\gamma & : & (g_{\alpha,e} \multimap H_{\beta,t}) \multimap H_{\beta,t} \\
[Think([\ ]_\delta, [\ ]_\epsilon)]_\zeta & : & h_{\epsilon,t} \multimap g_{\delta,e} \multimap f_{\zeta,t} \\
[\ ]_\lambda & : & g_{\lambda,e} \multimap (g_{\lambda,e} \otimes j_{\lambda,e}) \\
[Like([\ ]_\theta, [\ ]_\iota)]_\kappa & : & j_{\iota,e} \multimap i_{\theta,e} \multimap h_{\kappa,t} \\
[Mary]_\eta & : & i_{\eta,e}
\end{array}$$

The hookup is basically the same as for (41), except that the positive antecedent of the implicational argument of *nobody* is what is connected to the

Thinker argument of *think*, and the output of *think* is sent through the type t literals of *nobody*.

An important property of hookups with tensors is that a semantic content-based correctness criterion won't work, although a more abstract algebraic one will, as detailed in de Groote (1999). Intuitively, there is a requirement that the content-flow lines issuing from an output tensor must recombine before anything interesting is done with them, such as discharging a hypothesis upon which both depend. This is closely related to the way in which the pairs have to be handled in a deductive approach, to which we now turn.

In a deductive approach, $-\circ_{elim}$ with the pronoun and antecedent constructors will produce a pair, to be used in further deductions:

(43) $<[John]_{\beta}, [John]_{\beta}> \quad : \quad g_{\beta,e} \otimes j_{\beta,e}$

Unfortunately, in an intuitionistic linear system, this pair can't be broken up into its individual components, but must function together as a unit. In a Natural Deduction formulation for regular, lambda-based glue, it does so by means a rule of Conjunction Elimination (Asudeh 2004:55), which looks like this:[7]

(44)
$$\dfrac{\begin{array}{cc} & [x:A]^i \quad [y:B]^j \\ \vdots & \vdots \\ <a,b> : A \otimes B & f : C \end{array}}{\textbf{let } <a,b> \textbf{ be } x \times y \textbf{ in } f : C} \otimes_{elim,i,j}$$

This rule applies to two subderivations, one producing the pair $<a,b>$ associated with the tensor glue term, the other producing $f : C$ from a derivation with temporary hypotheses $x : A$ and $y : B$. Then what the mysterious-looking **let** statement in the conclusion's meaning-side does is tell us that we're supposed to derive the result by replacing x with a and y with b in f, thereby discharging the hypotheses.

For a unificational version of (44), we can get the effect of the **let** statement by unifying the left and right components of the pair with the left and right hypotheses from which the 'matrix' ($f : C$) premise is derived (so we need to keep track of the tag-updates in the derivation on the right of the rule).

[7] Slightly altered here to hopefully be a bit easier to follow

Asudeh's analysis of resumption inolves a further use of pairs/tensors, this time negative polarity (input-consuming ones) to implement the 'resource managers' that Asudeh uses to account for resumptive pronouns. The problem posed by resumptives is that they provide an excess pronominal resource, which must be disposed of in order to get a correct assembly. *Wh*-Questions in Irish provide a clear illustration:

(45) Céacu ceann aN bhfuil dúil agat ann
which one REL is liking at you in it
Which one do you like?
(McCloskey 2002:189, (10b), cited Asudeh (2004:106)

Here the preposed *Wh*-phrase provides the content for the propositional 'Likee' object, leaving the resource contributed by the pronoun as excess.

Asudeh refutes a theory that is likely to occur to many LFG practioners, which is that resumptive pronouns are semantically empty, lacking any kind of pronominal meaning-constructor at all; the basic problem with this idea is that it doesn't explain why resumptive pronouns always look like ordinary pronouns, rather than having a special morphological form.

The solution is to have the complementizer of resumptive pronoun constructions introduce a constructor that 'absorbs' the meaning-constructor of the resumptive pronoun, so that the net effect in terms of resources is as if the resumptive pronoun wasn't there (although it can introduce some information into the actual meaning). If a is the location of the antecedent resource, p the location of the pronoun, then the glue-side of the manager will look like this:

(46) $(a \multimap a \otimes p) \multimap a \multimap a$

This is likely to be fairly daunting at first glance, so the first thing to note about it is that its antecedent will match with, and thereby consume, the glue side of the pronoun. The result is something that consumes, and will be made to return as a product unaltered, the resource provided by the antecedent of the pronoun, so that the total effect is as if the pronoun weren't there. Here's a full relevant glue-side hookup:

(47) Antecedent: a

Pronoun: $a \multimap a \otimes p$

Manager: $(a \multimap a \otimes p) \multimap a \multimap a$

A question one might ask is why does the consequent of the whole thing have to have the form $a \multimap a$; the answer is that there has to be something in the consequent, so that the match to the pronominal constructor can be an antecedent and thereby opposite in polarity to the pronominal constructor, and $a \multimap a$ is the simplest thing that will do the job.

So what of the meaning-side? In the conventional approach, the meaning-side is $\lambda P.\lambda x.x$, which basically just throws away the meaning of the pronoun. I find this theoretically problematic, because if we can do this to pronouns, why not to anything? For example why couldn't there be a quantifier which simply asserted that its domain nominal has a nonempty extension, tossing the meaning of the scope:

(48) $\lambda P.\lambda Q.Some(x, P(x))$

Intuitively, the reason that the meaning of the pronoun can be eliminated is that it is merely a piece of plumbing, a stream-splitter, rather than a real contributor of content, but it's not clear how the lambda-based formulation will capture this distinction.

So how will this work out in the unificational formulation? For $\multimap_{elim}$, the meaning-side of the result is the meaning-side of the implicational premise, with some additional unifications, so the meaning-side of the resource-manager will have to be a structure co-tagged with the final output of the glue-side, and it should be clear that all the other terms have to likewise be co-tagged, so we get:

(49) $[\ ]_\alpha \ : \ (\alpha_{a,e} \ \multimap \ \alpha_{a,e} \ \otimes \ \alpha_{p,e}) \ \multimap \ \alpha_{a,e} \ \multimap \ \alpha_{a,e}$

Using this and the other indicated meaning-sides for the assembly of (47) above, we see that the net effect is that the resumptive pronoun could add some semantic features to the meaning of the antecedent (*Kim say's he/she is ready*), but no longer creates a resource problem. And it follows from the account together with the restriction that every literal on the glue-side must index a semantic (sub-)structure that apparent cases of total discard of meaning-constructors will only occur when the constructors make no substantive contribution to the meaning.

In summary, we find that unificational glue provides a workable treatment of anaphora and resource managers, and provides a reasonable basis for a plausible constraint against content being discarded, another beneficial effect.

5 Third Degree Arguments

We saw in the treatment of quantifiers that unificational glue can deal with second degree arguments of type $a \rightarrow b$ where a and b are atomic; it would be good to check that third degree works as well, such as with the analysis of *seek* provided by Dalrymple et al. (1997). To avoid putting lambda-abstraction in the meaning-language, we'll have to lexically decompose *seek* as meaning 'try to find', which might be seen as a bug or a feature, depending on how the project of lexical decomposition works out.

In the lambda-based analysis, *seek* takes as its object argument the qeneralized quantifier expressed by that object, which can therefore be intensional. If we want to do lexical decomposition, we can interpret this quantifier as applying to a predication $\lambda x.Find(s,x)$, where s is the content of the subject. In a lambda-calculus formulation, the whole meaning-constructor would look like this, here h, g, f are the f-structures of object, subject and verb/clause, respectively:

(50) $\lambda P.\lambda x.\mathrm{Try}(x, P(\lambda y.Find(x,y))) \;:\; ((h_e \multimap H_t) \multimap H_t) \multimap g_t \multimap f_t$

Cranking through how this accounts for the narrow-scope reading of *Mary seeks a Dane* is a good refresher exercise in Montague Grammar; on the other hand, understanding how it also deals with the wide scope reading, or a proper name object, as in *Mary seeks Fred*, is a good demonstration of the power of glue, and other logically-based approaches to grammar which in effect contain a rule of implication introduction as well as elimination (Oehrle 1998).

Given a subject and a quantifier, the hookups responsible for the two readings are:

(51) a. $(h_e \downarrow \multimap H_t \uparrow) \multimap H_t \downarrow$

$((h_e \multimap G_t) \multimap G_t) \multimap g_e \uparrow \multimap f_t$

$g_e \uparrow$

b. $(h_e \downarrow \multimap H_t \uparrow) \multimap H_t$

$((h_e \multimap G_t) \multimap G_t) \multimap g_e \uparrow \multimap f_t$

$g_e \uparrow$

The proper name hookup is like (b), except that the name output connects directly into h_e, with the verbs final output delivering the final result. The way in which the content-flow interpretation of proof-nets provides the correct results in conjunction with the proposed meaning-side for *seek* is rather interesting, but I won't it in detail here (the key is that the positive polarity $h_e \multimap G_t$ subexpression is assigned a free variable of semantic type $e{\rightarrow}t$ as content, which automatically gets applied to the content of h_e and then abstracted over, implementing automatic 'type lifting' of the object).

In the unificational account, the meaning-side of *seek* comes up as two distinct pieces, neither containing the other, but both sharing an argument (the syntactic subject):

$$\begin{array}{ll}[\mathit{Try}([\ ]_\alpha, [\ ]_\beta)]_\gamma & : \ ((\delta_{h,e} \multimap \epsilon_{H,t}) \multimap \beta_{H,t}) \multimap \alpha_{g,e} \multimap \gamma_{f,t} \\ [\mathit{Find}([\ ]_\alpha, [\ ]_\delta)]_\epsilon & \end{array} \quad (52)$$

A containment relation is however specified by the structure of the glue-side, which says in effect that the *Find*-predication will function as nuclear scope to a quantification, which will provide the second argument to *Try*.

This gets cashed out in assembly. If we add a constructor for a quantified NP such as *someone*:

$$[\mathit{Some}(x_\iota, \mathit{Person}(x), [\ ]_\kappa)]_\lambda \ : \ (\iota_{h,e} \multimap \kappa_{G,t}) \multimap \lambda_{G,t} \quad (53)$$

$$\begin{array}{ll}[\mathit{Try}([\ ]_\alpha, [\ ]_\beta)]_\gamma & : \ ((\delta_{h,e} \multimap \epsilon_{H,t}) \multimap \beta_{H,t}) \multimap \alpha_{g,e} \multimap \gamma_{f,t} \\ [\mathit{Find}([\ ]_\alpha, [\ ]_\delta)]_\epsilon & \end{array}$$

it can be seen that the two hookup patterns yield the two desired readings (subject still omitted):

(54) a. $[\mathit{Try}([\ ]_\alpha, [\mathit{Someone}(x, [\mathit{Find}([\ ]_\alpha, x])])]$

b. $[\mathit{Someone}(x, \mathit{Try}([\ ]_\alpha, [\mathit{Find}([\ ]_\alpha, x])])]$

To see this, note that $\iota_{h,e}$ and $\delta_{h,e}$ will have to link any any well-formed hookup, which fixes x as the second argument of *Find*. Then we have two choices for $\epsilon_{H,t}$: we can link it to either $\beta_{H,t}$ or to $\kappa_{G,t}$. The first choice identifies ϵ and β, thereby setting the *Find*-predication as the second argument of the *Try*-predication. The ensuing forced linking of $\gamma_{f,t}$ to $\kappa_{G,t}$ then yields the wide-scope reading. The second choice forces the later linking of $\lambda_{G,t}$ to $\beta_{H,t}$, with the net effect that the *Find*-predication is set as the nuclear scope of the

quantification, which is set as the second argument of the *Find*-predication. If the object is a proper name, then the first path is the only viable one, but there is no quantification, so the output of the verb becomes the final output.

I think it is intruiging that the unificational treatment winds up with the same ultimate result as the lambda-based one, via what appears superficially to be a rather different route.

6 A Major Limitation

Before concluding, I'll discuss a major limitation of the unificational glue approach, which could be either a strong point or a fatal flaw, depending on how things work out empirically. This is what would be in the lambda-based version a constraint if there is lexical decomposition on the meaning-side, a higher-order argument can only be used once within a meaning-side. The reason can be seen by looking at an analysis of NP-conjunction that is standard in the computational semantics literature:[8]

(55) a. Vincent and Mia danced

b. Vincent or Mia danced

c. Vincent but not Mia danced

On the lambda-calculus approach, these can be straightforwardly analysed in terms of a generalized quantifier analysis whereby the meanings constructed for the subject NPs would be:

(56) a. $\lambda P.P(\mathbf{v}) \wedge P(\mathbf{m})$

b. $\lambda P.P(\mathbf{v}) \vee P(\mathbf{m})$

c. $\lambda P.P(\mathbf{v}) \wedge \neg P(\mathbf{m})$

and with glue-sides effectively of the form $(g_e \multimap H_t) \multimap H_t$, where g is the f-structure correspondent of the coordinate NP (we'll be silent on the matter of how this is derived).

Such a treatment cannot be carried over smoothly to unificational glue as presented so far, because when we identify the subject of the *danced* predication with either one of Vincent or Mia ($\mathbf{v}$ or $\mathbf{m}$), we preclude identifying it

[8]See for example Blackburn and Bos (1999).

with the other. In principle, one might think of trying to wiggle around this by using subsumption to make copies of structures, but it's not so easy to come up with a clean and systematic way to do this starting with glue-logic notations, especially one that doesn't turn out to just be an implementation of the lambda-calculus. The lambda-based approach doesn't have this issue, since feeding an argument to a function is not construed as altering the function itself, so the same function can be applied to different arguments in different places, without causing trouble.

For the (a) and (b) examples, there is an alternative form of analysis that has already been suggested by Asudeh and Crouch (2002:9), which is to treat the coordinate NP as producing some kind of collectivity by means of non-boolean conjunction, and then interpret the conjunction as a quantification restricted by the constitution of that collectivity. For example (a) might be replaced with something like:

(57) $\lambda P.every(x, x \in \mathbf{v} \oplus \mathbf{m}, P(x))$

Here we use an 'i-sum' operation to construct a plural individual comprising Vincent and Mia, and then do universal quantification over that, thereby avoiding copying/two applications of one function to different arguments). It should be clear how to render (57) in unificational terms, and how to extend the treatment to (56b). But the (c) example seems to resist such treatment, and similar problems arise elsewhere, such as with the presuppositions associated with *again* (Kamp et al. to appear). So it may be the case that unificational glue is just wrong.

Then again, maybe not. Glue assembly is not necessarily the correct place to handle presuppositions, and there are other potential analyses for structures such as (55c). A difference between (c) on the one hand and (a, b) on the other is that (c) is considerably more natural with comma-pauses after the conjuncts:

(58) a. ?Vincent, and Mia, danced

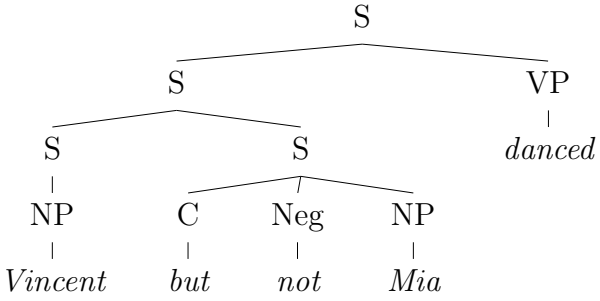
b. ?Vincent, or Mia, danced

c. Vincent, but not Mia, danced

In (58a) it is in particular clear that there is no group reading for the subject; Vincent and Mia are being said to have danced at some event, but not being said to have danced with each other.

Amongst the conceivable analyses for these examples, which would be consistent with the comma-pauses, is that they are actually right node raising constructions where a VP is extracted from coordinate Ss, which dominate the NPs:

(59)



7 Conclusion

Using unification instead of lambda-calculus in glue logic appears to provide viable analyses for an interesting range of the constructions that have so far been analysed in LFG with glue, with some prospects of supporting empirically useful constraints. I'll close by suggesting that a major factor in determining whether one should be interested in this program is what kind of attitude one takes towards the level of 'logical form' (LF; or 'semantic representation', or 'conceptual representation'). People who are interested in the idea that such a level might exist think of it as something significantly more abstract than conventional 'syntactic representation' (comprising structures to support grammatical relations, inflectional features, etc.), but which is nonetheless a genuine level of linguistic representation, that is, a kind of syntax, as a logician would see things. It has been clear since the work of Montague that there is no conceptual requirement for such a level to exist (we don't actually need it to produce formal theories of entailment), but it is possible to argue that it nonetheless does (for example Jackendoff 2002).

If we're interested in the idea that both f-structure and LF exist as levels of linguistic representation, we ought to be interested in proposals that make the relationship between f-structure and LF similar in nature to the relationships between other linguistic levels, in particular, we should want to be able to see it as involving parallel correspondence relationships with a requirement consistency of information playing an important role. This can be done at best rather awkardly in the lambda-calculus-based approach, but

is quite natural under the unificational one, because we can the coindexing relation implemented by the tags induces a many-to-many correspondence between f-structure and the meaning-structures (LF). To get maximal symmetry with the c-structure and f-structure relationship, we should perhaps see the linked glue terms as playing the same role as the c-structure tree, relating to f-structures in one direction and semantic structures in the other, as the c-structure tree does for f-structure and phonology. It remain to be seen whether this view will lead to good empirical results.

Bibliography

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

Andrews, A. D. 2004. Glue logic vs. spreading architecture in LFG. In C. Mostovsky (Ed.), *Proceedings of the 2003 Conference of the Australian Linguistics Society*. URL: `http://www.newcastle.edu.au/school/lang-media/news/als2003/proceedings.html`.

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

Asudeh, A. 2004. *Resumption as Resource Management*. PhD thesis, Stanford University, Stanford CA. URL: `http://www.stanford.edu/~asudeh`.

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`.

Blackburn, P., and J. Bos. 1999. *A First Course in Computational Semantics*. Vol. I. available Oct 2004 at `http://www.cogsci.ed.ac.uk/~jbos/comsem/book1.html`.

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

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

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

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

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

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

Kamp, H., J. van Genabith, and U. Reyle. to appear. Discource representation theory. In *Handbook of Philosophical Logic*. 2nd edition. Draft at URL: `http://www.ims.uni-stuttgart.de/~hans/`.

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).

Oehrle, R. T. 1998. Multi-modal type logical grammar. To appear in Borsley and Böjars (eds), *Non-Transformational Syntax*, Blackwell.

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`.