Semantic Composition for NSM, Using LFG+Glue[*]

AVERY D. ANDREWS
School of Language Studies
The Australian National University
Avery.Andrews@anu.edu.au

Abstract
The Natural Semantic Metalanguage (NSM) program of Anna Wierzbicka and her colleagues has a lot to say about the meanings of individual words, but virtually no work has been done on the problem of how to assemble these meanings to produce meanings for utterances, which is the problem of semantic composition that is the major focus of formal semantics. In this paper I begin to fill this gap by making some definite proposals for doing semantic composition in NSM using the 'glue logic' that has been proposed as a method of semantic assembly for the syntactic theory of LFG.

Although many different generative syntactic theories could provide a basis for semantic composition in NSM, LFG is a reasonable choice, because it combines to a relatively high degree the properties of being formally explicit, easy to learn, and applicable to a typologically diverse range of languages, and the architecture of LFG+Glue provides a clean separation between issues of semantic composition on the one hand, and syntactic realization on the other.

I will examine some issues that arise in composing explications for some of the valence options of the verbs *warn* and *go*, showing that naive substitution is insufficient, but that the typed lambda calculus can deal with the problems adduced. We will also see that the problem of composing explications should not be deferred indefinitely, since attempting to compose explications can expose deficiencies which aren't evident when the explications are viewed in isolation. I will conclude with a brief discussion of some of the problems afforded by phenomena of quantifier scope.

Keywords
NSM, Natural Semantic Metalanguage, Semantic Composition, Typed Lambda Calculus, Glue Semantics, Glue Logic, LFG, Lexical Functional Grammar

[*] I am indebted to Anna Wierzbicka, Cliff Goddard, and two anonymous ALS reviewers for useful comments and discussion of previous versions of this paper, as well as the audience of a closely related presentation at the 2005 ALAS conference at Macquarie University.

1 Introduction

The Natural Semantic Metalanguage (NSM) program of Wierzbicka and her colleagues[1] has been extensively developed as a theory of lexical semantics, but has no account at all of semantic composition, the way in which word and construction meanings are combined to give meanings for novel utterances. This is a serious deficiency, because without such an account, NSM has nothing to say about how people can understand an in-principle infinite range of utterances that they haven't encountered previously. And it is a deficiency which it would be good to address sooner rather than later, since attempting to compose explications can reveal problems: explications which seem reasonably good in isolation often produce bad results when combined. So one can't just do lexical semantics indefinitely and assume that the results will continue to hold up when attention is turned to semantic composition.

In this paper I will show how to make a start on doing semantic composition for NSM by using Lexical-Functional Grammar (LFG) and 'Glue Logic'.[2] Although this is certainly not the only possible approach—many contemporary syntactic theories could do the job—it is a convenient combination, because LFG normalizes a great deal of cross-linguistic grammatical variation in terms of case-marking, agreement, word-order and other features of overt structure into a far more uniform system of 'f-structures', which is furthermore close to traditional conceptions of grammatical relations and inflectional features such as tense, number and case. Then Glue Logic (so named because it is based on a kind of logic, and is used to 'glue' individual contributions to meaning into an integrated result) provides a technique whereby the f-structures (with further input from overt constituent structure, if necessary) can constrain semantic composition. The net result is that LFG+Glue allows one to think about problems of semantic composition in one language without having to get enmeshed in the details of overt syntax, but with reasonable confidence that the results will carry over to many other languages.

2 Variables and Substitution

Although NSM lacks an explicit account of semantic composition, there have been from the beginning some implicit hints as to what might be involved, in the form of upper case 'variables' in explications that appear to be intended as targets for substitution. An example from Goddard (1998:205) that we will be discussing from various points of view is this explication of the three-argument valence frame of the verb *go*:

[1]See for example Goddard (1998), Wierzbicka and Goddard (2002), and the NSM homepage at `http://www.une.edu.au/arts/LCL/disciplines/linguistics/nsmpage4.htm`.

[2]For a thorough introduction to LFG, see Falk (2001); for recent discussions of Glue Logic at introductory and more advanced levels, see Andrews (2004, to appear), Asudeh (2004), Lev (2005), and Dalrymple (1999, 2001).

(1) X went from A to B (yesterday) =
before this X was in place-A
X wanted to be somewhere else
because of this, X moved for some time (yesterday)
because of this, after this, X wasn't in place-A anymore
X was in place-B

There are several features of this explication that require comment. One is the parenthesized 'yesterday', which I will take as an allusion to the fact that something must be done about tense and time reference, a problem that I will entirely ignore. A substantive semantic point is that it attributes volitionality to the subject. This might be challenged on the basis that sentences such as *John's suitcase went from New York to Karachi* are fine, but observe that *John went from New York to Karachi* implies that John at least intended to go somewhere, although not necessarily to Karachi (he might have wanted to go to, say, Reykjavík, but got on the wrong plane). And this sentence would not cover a situation where the CIA just shipped John off to Karachi with no active involvement on his part. Therefore, the explications for *go* with human versus inanimate subjects would appear to be different (higher animals such as dogs can be treated either way, it seems to me). And finally, there are the instances of 'place-' prefixed to the variables A and B, which need to disappear under substitutions, and which I will take as indications of the need for some sort of type system, something which we will be discussing later.

This use of variables in NSM is found as far back as Wierzbicka (1972), and so may be considered as an original feature of the framework. And it is reasonable to suppose that if we want to compose the meanings of the words in a sentence such as:

(2) I/You went from Albury to Mildura

what we are supposed to do is substitute things based on the explications of the NPs in the sentence for the variables, in some manner prescribed by the overt syntax. In (1), for the variable X of the explication of *go*, the obvious thing to substitute is the semantic primitive representing the subject itself (*I* or *you*), while for the source and goal proper names, I would suggest that we simply substitute the names themselves. This might be regarded as an evasion of rather than a solution to the problem of proper names in NSM (Goddard, pc), but it has the virtue of being simple (and consider that all languages seem to have proper names, and use them in pretty much the same way).

So one could envision a 'naive subsitution-based' approach to semantic composition in NSM, whereby some device uses explications as standardly presented together with the grammatical structure to assemble word-explications (and maybe construction-explications) into utterance explications. This is in fact not too far off what we will be actually suggest, except that there turns out to be some pre-exisiting mathematics (typed lambda-calculus) that is relevant for organizing the substitutions. There are however at least three problems that might be adduced against the general approach, the first one trivial, which will be dealt with here, and the other two requiring some more serious development, in following sections.

The trivial problem is that NSM is not supposed to be making use of abstract symbols, which is what the variables appear to be. The answer is that these symbols can be thought of as 'assembly symbols', which are used only to orchestrate semantic assembly, and disappear from the final results, where only NSM primitives ('content symbols') are allowed. This clarification of the policy is worth establishing now, since when we get to typed lambda-calculus, the apparatus of assembly symbols becomes substantially more imposing, and we will have a use for a mathematical result to the effect that we can always tell when it will disappear as required.

3 Accidence

A more substantial problem is that when we substitute different primitives for X in (1), it is sometimes necessary to make certain flow-on adjustments to the form of the explication, as required by phenomena of case-marking and agreement, traditionally called 'accidence'. For example if the subject is 'I', we want the last line of (1) to be 'I was in Mildura', but if it is 'you', 'you were in Mildura'. The problem will be more extensive for NSM explications in languages where the role of case-marking and agreement is greater.

There are in principle at least two possible solutions to this problem. The first would be to construct a system of 'morphological fixup functions', which would take as inputs combinations of primitives and perhaps other symbols such as *be+Past+I*, and would produce as outputs actual words such as *was* and *were*. The facilities for the mid-90s activity of MOO-programming could be adapted for this purpose,[3] but I think a better approach would be one that addressed some other issues at the same time, such as formalizing NSM syntax, especially the valence properties of the primitives.

Although the use of grammatical constructions in NSM was originally rather free-wheeling, working out a definite syntax has become a priority, explored in many of the papers in Wierzbicka and Goddard (2002), although not on the basis of formal syntactic proposals. A rather salient problem is how to state the valence (combinatorial properties) of the primitives in a uniform way, while the overt syntax of the different NL instantiations of the NSM metalanguage show a wide range of variation in how syntactic relations are expressed in terms of surface word-order, case-marking, agreement, etc. A reasonable way to approach this problem would be use some selection from the current techniques of generative grammar (construed widely to include all mathematically-based approaches to linguistic structure, not just the GB/Minimalist tradition) to formalize NSM, by setting up first a universal system of what we'll call 'NSM terms', and then language-particular 'rendering schemes' to convert NSM terms into ordinary expressions (typically, monologic discourses) in the various specific NL instantiations of NSM. So the NSM term for *this is good* might be 'GOOD(THIS)', with the rendering component for English NSM responsible for providing the copula, and setting up the agreement and morphologically present tense in the English version of the explication.

The NSM terms will then lack phenomena of accidence, so that we can do substitu-

[3]MOOs were user-programmable text-based virtual environments, in which for example you might enter a room and be told that you saw a lizard, and type something like 'pick up the lizard'. Then you would see on your screen 'you try to pick up the lizard, but it hisses and scurries away', while other people in the room would see 'Maal Dweb tries to pick up the lizard, but it hisses and scurries away'. A description of these facilities can be found at `http://www.nwe.ufl.edu/writing/help/moo/jhc/builder_help.shtml`).

tions into them without worrying about this problem, and deal with it later in the rendering component. The resulting setup would have a significant resemblance to formal semantics, with overt NL expressions connected to abstract formulas, but there is a very important difference in the flow of explanation for meanings: in formal semantics, the abstract structures are supposed to explain the meanings by virtue of some sort of mathematical definition[4] of entailment and other meaning-based relations between utterances, while in this view of formalized NSM, the abstract structures are part of an account of the syntax of NSM, and the meanings reside in the overt utterances.

Working out this kind of syntax for NSM is not a trivial project, and could be approached in various ways, but one thing which we are likely to want and will in any event need for this paper is something called a 'type system', which can be thought of as a set of categories into which expressions of a formalized language can fall. For example a phrase-structure grammar has a simple, finite type system with no interesting structure, consisting of its node-labels (phrase-types and parts of speech).

However most work on type systems focusses on infinitary ones with interesting structure, and some kind of ontological significance for the types is typically assumed. For example there might be a type e for 'entities', and a type t for 'propositions'.[5] An infinite system of types is then produced by means of 'type constructors' which create new types from combinations of old ones, of which the most essential is the 'exponential' type constructor, which combines a type a and a type b to produce the exponential type $a{\rightarrow}b$. This is understood to be a type of expressions which combines with expressions of type a to produce expressions of type b. For example if 'GOOD' is of type $e{\rightarrow}t$ and 'THIS' is of type 'e', then 'GOOD(THIS)' will be of type t.

This example illustrates that to go along with an exponential type constructor, we need a syntactic construction which we will call 'application', whereby a 'predicate' of an exponential type is applied to an 'argument' of the input type of the exponential, to produce an expression of the output type of the exponential.[6] There are various notations in circulation for application; the one most commonly encountered in linguistics is to put the predicate first, followed by the argument in parentheses.

What about predicates with two or more arguments? Since Montague, the standard way of dealing with these in formal semantics has been to treat them as predicates which apply to an argument to produce another predicate; for example a two place predicate such as SEE would be of type $e{\rightarrow}(e{\rightarrow}t)$, so that 'I see this' would have the NSM term representation 'SEE(THIS)(I)', if we assume the widely followed convention, motivated by Marantz (1984), of applying the 'least active' argument first. Notice now, in the NSM term, that the predicate

[4]Usually model-theoretic, but interest in deductivex accounts seems to be increasing, as discussed recently by Szabolcsi 2005.

[5]There is an issue as to whether e and t shouldn't rather be 'sorts' (following Partee and Borschev (2004), elements of the naive ontology of language), rather than types; we'll see below that we definitely want there to be types, and it is reasonable although not perhaps strictly necessary to include e and t among these.

[6]People with some background in formal semantics might expect to hear about functions here, but for the purposes of formalizing NSM, we only want the syntactic aspects of type-theory, not the model-theoretic ones, at least in the first instance (it would be interesting to try to do model-theory on NSM, but not esssential). Therefore I use the more grammatically-oriented term 'predicate'.

'SEE' applies to the argument immediately after it to produce the predicate 'SEE(THIS)' of type $e \rightarrow t$, which then applies to the next argument, so that there is implicit leftward grouping of application expressions. This technique, called 'currying' (after the mathematician H.B. Curry, who used it extensively, although it appears to have been invented earlier), is a bit hard for beginners to get used to, but allows us to do a great deal using only the exponential type constructor.

Before moving on to the next problem, I'll point out that there are some fairly difficult issues involved with setting up a type system for NSM. For example, do we want to take a rather syntactic view, and put 'someone' and 'something' in the same type, or a more ontological/conceptual one, and have distinct person/thing types? In the latter case, what do we do about 'semi human' favored animals, such as dogs and cats, for which both 'someone' and 'something' don't seem to be very appropriate (*some animal/*someone/*something has gotten into the garbage*). I won't advocate any particular position here, but would tend to assume that 'someone' and 'something' are of the same type, but different sorts. This question also arises for the 'place-' qualifiers in the explication (1).

Since we lack a worked out formalization of NSM, we will continue to write out explications informally in English NSM, although a proper formal term system or equivalent is ultimately wanted.

4 The Failure of Naive Substitution

Now we come to our third problem, which is that naive substitution is not enough. This can be seen by considering an explication such as this one for the verb 'warn' (Wierzbicka 2005), and how we need it to behave under composition:

(3) X said something like this to Y:
    If you do not do this:
        Z
    something bad can happen

Suppose we want to combine this with other explications to get a meaning for a sentence such as:

(4) You warned me to go

For reasons that we will consider later, I suggest using the following explication for monovalent *go* instead of Goddard's (1998:204) original one:

(5) X does something because X doesn't want to be in some place anymore
    because of this, after this, X is in another place

If we just do our substitutions naively in the obvious way, we would presumably end up with something like this:

(6) you said something like this to me:

  If you do not do this:

    X does something because X doesn't want to be in some place anymore
    because of this, after this, X is in another place

  something bad can happen

But this is clearly wrong, since (a) we have some uneliminated assembly symbols in the final result (b) this result concomitantly doesn't intuitively mean anything, let along the right thing.

The effect we would like to achieve is to have 'you' substitute for X in the 'go' explication, yielding this as final result:

(7) you say something like this to me:

  If you do not do this:

    you do something because you don't want to be in some place anymore
    because of this, after this, you are in another place

  something bad can happen

But naive substitution cannot achieve this, at least without losing some of its naivete.

But fortunately, there is an established mathematical technique that can achieve the effect we want, which is flexible enough to have some plausibility as a general solution, and has appropriate mathematical properties to be combined with (a formalized version of) NSM. This is the typed lambda calculus, to which we turn in the next section.

5 Typed Lambda Calculus

One way of thinking about the problem posed by the bad assembly (6) is that to eliminate it, we need to get 'you' to substitute for X in the explication of the complement verb. Observe that syntactic control won't help here, since the syntactic controller is *me*, associated with the primitive 'I' rather than the desired 'YOU'. But if we could trigger some further substitutions inside of the explication of 'warn', things might work out. In particular, keeping in mind the above discussion of the application construction for exponential types, it might occur to us to write down something along the lines of:

(8) X said something like this to Y:

  If you do not do this:

    Z(you)

  something bad can happen

which expresses the hope that Z can be construed as a predicate of type $e{\rightarrow}t$, which will somehow apply to 'you' so as to produce our desired result.

Typed lambda calculus (TLC) provides some rather deeply understood facilities for doing this kind of thing. For our current purposes, a hopefully straightforward way of understanding it is as a technique for extending any kind of formalized language that has a type system, especially if that language already has an application construction. The ingredients are as follows:

(9) a. In the type system, an exponential type-constructor and corresponding application construction, both of which may already be present.

b. For each type, an infinite number of variables of that type. These can be formally represented with some symbol superscripted with the type and subscripted by a positive integer (ξ_1^e, $\xi_{146}^{e\rightarrow t}$, etc.), although people tend to informally use various letters without subscripts, and omit the type superscripts when they're clear from context (x^e, $P^{e\rightarrow t}$, etc). The variables might already be present in the language, although probably not in the case of formalized NSM.

c. The 'lambda (λ-) abstraction' construction: given a term E of type b and a variable X of type a, the lambda-abstract of E by X is $(\lambda X.E)$, of type $a\rightarrow b$ (meaning roughly, something which, if fed something of type a, produces something of type b). Syntactically, this is a technique for making new predicates out of pre-existing materials. Parentheses not needed for disambiguation are usually omitted.

We want to characterize this as an 'extension' of a formal language, because the significance of the lambda-abstraction is given by the rule of 'β-reduction', along with some additional principles, which will eliminate the lambda-abstractions from certain formulas, reducing them to expressions in the original, unextended, language. This is clearly essential for any NSM application, since we certainly don't want the lambda-apparatus in particular to appear in final explications for utterances.

We now take a brief informal look at how β-reduction works—a careful and thorough account of the technicalities can be found in Hindley and Seldin (1986), and many introductions to formal semantics (although these tend to dwell on the model-theoretic interpretation of TLC, which we don't need for our present purposes). A concise on-line presentation is also provided by Pollard (2004).[7]

Since the type of $(\lambda X.E)$ will be $a\rightarrow b$ if X is of type a and E is of type b, we will be able to put this in front of an expression A of type a, so as to get the following result of type b:

(10) $(\lambda X.E)(A)$

But what is (10) supposed to mean? The β-reduction rule says, for a first approximation, that what it means is what you get by taking E, and replacing in it every 'free' occurrence of X with A. A free occurrence of a variable X in an expression E is one that does not

[7]Unfortunately, Hindley and Seldin (1986) is currently out of print, and any begins with the untyped lambda calculus, so that a beginner would not find it easy to deal only with TLC. A relaxed but rigorous free-standing introduction to the syntactic aspects of the TLC would be quite useful.

occur inside any lambda abstraction within E whose variable is X itself (these can be seen as ‘protected’ by their binding λ). In particular, if our application expression is:

(11) $(\lambda X^e.$ X does something because X doesn’t want to be
in some place anymore
because of this, after this, X is in another place $)(\text{you})$

(think of the box as an indication that the contents are an informal NL rendition of an NSM term, and note the omission of type superscripts on bound occurrences of variables), we then want the following to be the result of β-reduction applying to (11):

(12) you do something because you don’t want to be
in some place anymore
because of this, after this, you are in another place

It is hopefully clear how this will work, and that it will fit with the tentative explication (8).

But we should also say something about why we described this as only a ‘first approximation’ to β-reduction. A problem arises if A contains any free variable that becomes bound upon substitution into E (X itself doesn’t count, since original occurrences free in E disappear). If this were allowed to happen, the order in which we did β-reductions in complex expressions would matter, which is something we don’t want to happen. To prevent it, if A contains any such free variables, we replace the variables in E that would capture them with others that won’t. See Hindley and Seldin (1986:7-10) for a careful discussion of how this is done (α-conversion), and pg. 72-73 for the final rule of the system, η-reduction.

To move on to a complete assembly for our example, we need to observe that lambda-abstracts can be nested within each other. So for example if we have expression E of type t, and variables X, Y of type e, then we have expression $(\lambda X.(\lambda Y.E))$ of type $e{\rightarrow}(e{\rightarrow}t)$, which, following (Hindley and Seldin 1986:3), we can conventiently abbreviate as $(\lambda XY.E)$. Now if we write $(\lambda XY.E)(B)(A)$, this will be β-reduced in two stages, by first replacing the free X’s in E with B, and then the free Y’s with A (making any substitutions needed to avoid wrongful capture of variables).

So now we can revise the explication of *warn* to:

(13) $\lambda Z^{e\rightarrow t}Y^eX^e.$ X said something like this to Y:
If you do not do this:
$Z(\text{you})$
something bad can happen

If we abbreviate (13) as *Warn*, and (11) as *Go*, the final result is that

(14) *Warn*$(Go)(\text{you})(\text{I})$

reduces to the desired result (7).

TLC can thus manage at least the kinds of substitutions that seem to be required to do semantic assembly for *warn*, and so seems like a reasonable technique to start with, especially because of the depth of understanding which has been achieved for it. Most importantly, it is possible to mechanically tell whether β-reduction and the other rules of TLC can convert a given expression to a 'normal form' in which there are no lambda-abstracts. This is essential for NSM, due to the requirement that these be absent from the final form of an assembled explication. [8] It is perhaps worth pointing out that in the syntactically simpler but mathematically much deeper 'untyped lambda calculus', there isn't any mechanical way to tell whether the lambda-abstracts can be removed from a given expression, making this not so suitable for use in conjunction with NSM.

6 Explications Under Composition

In addition to simply being able to manage semantic composition (and thereby *inter alia*, making some progress towards allowing a meaningful comparison of NSM with formal semantics), I said earlier that trying to compose explications can reveal inadequacies, which would better be revealed sooner than later.

An example of this is provided by Goddard's (1998:204) explication for monovalent *go*, somewhat adapted by the removal of the time adverbial, and the addition of a lambda-abstraction:

(15) $\lambda X.$

Before this, X was somewhere
X wanted to be somewhere else
because of this, X moved for some time
because of this, after this, X wasn't in this place anymore
X was somewhere else

Using this, what we will get for *you warned me to go* will be an assembled explication like this:

(16) you say something like this to me:

  If you do not do this:

    before this you are somewhere
    you want to be somewhere else
    because of this, you move for some time
    because of this, after this, you aren't in this place anymore
      you are somewhere else

The problem of course is with the 'before this you are somewhere' component. This is wrong and makes no sense, because it is not part of what the addresee of *warn* is being told to do, but behaves like a presupposition of the warning.

[8] It is generally thought that lambda abstractions are needed in the expression of certain kinds of sentence meanings (Pollard 2001:2), but if current NSM isn't too far off the mark, this must be incorrect.

This problem is avoided by the explication given in (11), where in effect the presupposition is built into the 'anymore' qualification in the explication. Another feature of this explication, not strictly relevant to the topic here, but worth mentioning briefly, is the replacement of the 'X moved for some time' component with 'X did something'. The justification for this is a scenario such as the following.

Bob and Alice, characters in a science fiction story, get bored with their present location and decide to go to the Tau Ceti Mega Mall. They hop onto a teleportation platform, then Alice uses her implants to set the destination as TCM2, and activate the machine. At this point they instantly 'go to the Tau Ceti Mega Mall', without anybody 'moving for some time', or indeed, moving at all. But Alice's use of her implants to activate and direct a machine does seem to count as 'doing something', so that she can go somewhere by doing this (and Bob can count as going by proposing the destination, or assenting to Alice's proposal).

The semantic behavior of presuppositions under syntactic embedding is notoriously complex, and it remains to be seen if NSM can deal with all cases as easily as it seems to handle monovalent 'go' with the aid of the 'ANYMORE' primitive. For example *again* seems to afford some tough challenges, which make for a good comparison with formal semantics due to the extensive treatment of *again* in Kamp et al. (to appear) within the framework of current Discourse Representation Theory.

7 Controlling Composition

The final thing we will do is to give a fairly brief indication of how LFG+Glue can organize the assembly of the lambda-terms so as to produce the results that we have been wanting to get. I'll for the most part assume basic LFG, although giving review hints at various points. There are in fact a number different-looking but mathematically equivalent ways in which this can be done. One attempt at an explanation for beginners in terms of things called 'proof-nets' is provided by Andrews (2004); here I'll try a different approach based on the use of 'Natural Deduction' as used by Asudeh and Crouch (2002) and Asudeh (2004), but with a slight change in notation that might make it a bit more accessible to syntacticians.

To set the stage, consider what's involved in assembling a meaning for the sentence *John likes Mary*. Assume we start with semantic contributions for the three words, with types as follows:

(17) $John^{e}, Mary^{e}, Like^{e\to e\to t}$

As far as the types alone are concerned, there are two possible assemblies:

(18) a. $Like(John)(Mary)$

b. $Like(Mary)(John)$

of which the grammar of English allows only (b) (assuming the least-active argument first convention).

One of several possible ideas for implementing such restrictions is to let the syntactic structure enrich the type information in some manner that would deliver the required constraints. In LFG, the sentence will have as one of its syntactic representation an ‘f-structure’ that looks pretty much like this:

(19)
$$f:\left[\begin{array}{ll} \text{SUBJ} & g{:}\left[\text{PRED}\quad \text{‘John’}\right] \\ \text{TENSE} & \text{PRES} \\ \text{PRED} & \text{‘Likes(SUBJ, OBJ)’} \\ \text{OBJ} & h{:}\left[\text{PRED}\quad \text{‘Mary’}\right] \end{array}\right]$$

For people whose LFG is rusty, an f-structure is a representation of the kind of information about a sentence provided by traditional grammar, but with less in the way of parochial assumptions, and more in the way of a worked-out mathematical framework. The components of the f-structure, which are the things surrounded by square brackets, represent grammatically significant units in the structure of the sentence, which may be represented by discontinuous sequences of words, or no words at all (in the case of ‘omitted arguments’), and the upper-case grammatical function labels such as SUBJ and OBJ to their left indicate the grammatical relations that the components bear to each other and the whole. These components are themselves f-structures, so the whole f-structure is therefore a composite made of smaller f-structures, composed as specified by the labels. Grammatical feature labels such as TENSE also appear to the left of grammatical feature values such as PRES, which represent grammatical properties of the grammatical units. In this diagram, each f-(sub)structure has an italic ‘tag’ in front of it, to facilitate external reference to the structure and its components (these lack theoretical significance in themselves, but can be used to help specify theoretically interesting things, such as, for us, now, the way in which the syntactic structure constrains the semantic assembly.

Intuitively, one can think of the words *John* and *Mary* as ‘delivering content’ of type e to the subject and object f-structure positions, which are the g and h substructures, respectively, and the word *likes* as collecting content from these positions and delivering content of type t to the f-structure of the whole sentence.[9] Now suppose that, instead of being restricted to being merely the types of our meaning-language (a formalized NSM), the types relevant for assembly also included an f-structure tag (intuitively interpretable as a location). We could do this by taking the types to be pairs composed on an f-structure tag and a semantic type. The superscript notation for types becomes rather unreadable if we do this, so we’ll change the notation to put the type to the right of a colon, so that the types of (17) are replaced by:

(20)
$$\begin{array}{rcl} \textit{John} & : & <g, e> \\ \textit{Mary} & : & <h, e> \\ \textit{Like} & : & <h, e>\rightarrow<g, e>\rightarrow<f, t> \end{array}$$

These types clearly constrain the semantic assembly in the desired way, for this simple ex-

[9]The intuitions about content delivery and collection can be made precise in the proof-net-based formulation of Glue, as discussed for beginners in Andrews (2004).

ample.[10]

And, happily, although our account of the kind of assembly we want has so far been intuitive, there is a pre-existing system (albeit much younger than TLC), that does what we want. This is the 'linear logic' of Girard (1987), which is a variant of standard logic in which premises by default disappear when used, and so can only be used once. This tends to strike people as a rather bizarre feature to put into a logic, so to motivate it a bit, consider that a person interested in the structure of valid arguments might be want to count how often a premise is actually used in order to derive a given conclusion. For example we might have two arguments to the effect that the world is doomed, one of them making no use at all of the premise that George Bush is a imbecile, and another using this premise three times (the end of the world following from three bad decisions whose inevitability each follows from the GBi premise). Once you start counting premises, you've basically started doing linear logic. And whatever its motivations and antecedents in logic are, it immediately delivers an essential feature of the semantic interpretation of NLs, that word (and perhaps construction) meanings are normally used once, and once only, something which is easy to guarantee when the syntactic structure is a tree, but not when it might contain multiattachments and even cycles (since a given semantic contribution will be encountered more than once on a scan of the structure).[11]

The technique whereby linear logic deductions are used to control semantic assembly is a standard tool of categorial grammar, the notion of a 'labelled deduction', in which the formulas of some logical system are paired with objects of some other kind, and deduction rules whereby formulas produce formulas are paired with operation rule whereby the objects paired with the premises produce an object paired with the conclusion. We can in fact interpret the formulas of (20) in this way: to the left of the colon, on the 'meaning side', appears an expression in our meaning language, and to the right, on the 'glue side', a formula in linear logic, where the '$\rightarrow$' arrow is conventionally replaced by the symbol '$\multimap$', often glossed 'lollipop', which is standardly used to represent implication in linear logic. If we add the semantic types to the meaning sides, (20) becomes:

(21)
$$\begin{array}{rcl} \textit{John}^{e} & : & <g,e> \\ \textit{Mary}^{e} & : & <h,e> \\ \textit{Like}^{e\rightarrow e\rightarrow t} & : & <h,e> \multimap <g,e> \multimap <f,t> \end{array}$$

The evident redundancy here will be discussed soon.

In the Natural Deduction formulation of linear logic, the rules dealing with the $\multimap$ connective are $\multimap$-elimination and $\multimap$-introduction (basically the same things as traditional Modus Ponens and Hypothetical Deduction, respectively). The assemblies needed for our analysis of *warn* can all be done with $\multimap$-elimination alone, which is traditionally formulated in various ways, of which the 'tree format' is most suited for our present purposes:

[10]This can be seen as a slight extension of Klein and Sag's (1985) notion of 'bounded closure', which can be seen as a sort of antecedent to the basic ideas of Glue Semantics (Asudeh 2004:88-89).

[11]Linear logic is one of a number of 'substructural logics', to which a comprehensive introduction can be found in Restall (2000). An incomplete but extremely useful introduction to linear logic for linguists is Crouch and van Genabith (2000).

(22) $$\frac{A \multimap B \qquad A}{B}\ \multimap\text{-elim}$$

Here the premises appear above the line, the conclusion below, and the name of the rule used to justify the conclusion to the right of the line. Derivations of arbitrary size can be made by combining these structures together in trees (ordinary, rightside-up ones rather than the upside-down ones favored by linguists); the linear logic restriction that every premise be used once and once only amounts to the requirement that the tree have no 'upwardly combining' branches (re-entrancies), and that it be connected.

For semantic assembly, we need the labelled version of ⊸-elim, where the label of an implication is a lambda-abstraction of 'corresponding type' (to be defined more rigorously below), and what ⊸-elim does to the labels is apply the label of the implication to the label of the other premise:

(23) $$\frac{f : A \multimap B \qquad a : A}{f(a) : B}\ \multimap\text{-elim}$$

This will only make sense if there is a systematic relationship between the structures of the formulas on the right and the types of the labels on the left, and what we want for glue is in essence the requirement that we can derive the semantic type from the formula by forgetting the f-structural information, and replacing the ⊸ connective with →.

More explicitly, the type that a meaning expression must have in order to be paired with a glue expression can be derived by the following conversion function $\mathcal{C}$:

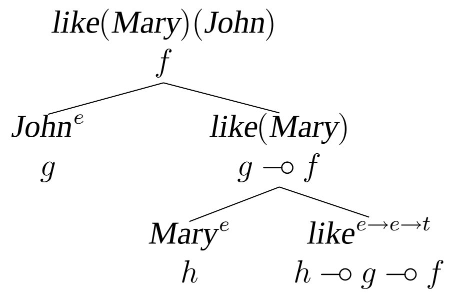
(24) a. If a formula is a 'literal' of the form $<f, a>$, where f is an f-structure tag and a is a type (of our TLC), then $\mathcal{C}(<f, a>) = a$.

b. If a formula is of the form $A \multimap B$, then $\mathcal{C}(A \multimap B) = \mathcal{C}(A) \rightarrow \mathcal{C}(B)$.

Thanks to this relationship, we can omit the semantic type information from the glue side, since it is predictable from that of the meaning side.[12]

A pairing of a meaning expression with a glue expression of compatible type is called a 'meaning constructor', and the meaning constructors of (21) can be assembled in only one way, which can be made more familiar-looking to syntacticians by hanging the tree upside-down, and made more horizontally compact by stacking the meaning and glue expressions vertically rather than separating them by colons:

[12]There is however a tradition of writing constructors in a format such as $\lambda XY.\textit{Like}(X, Y) : g \multimap f$, with content-free lambda-abstractions being used as partial indications of the semantic type on the meaning side, and no information about the semantic type on the glue side. I see no point in doing this. In situations where the comma-separated argument notation is useful, one can use Montague's convention of treating $f(a_1, \ldots, a_n)$ as syntactic sugaring for $f(a_n)\ldots(a_1)$ (the order-reversal is to respect the traditions used when using the comma-separated notation, where the most rather than the least active argument tends to come first). We are also omitting the 'semantic projection' from the meaning constructors, for reasons discussed in Andrews (2004).

(25)


Here we specify the semantic type information only on the meaning sides of the leaves of the tree, since it is predictable elsewhere; note also that the linear order of sisters isn't significant.

Therefore, enriching the semantic type information with f-structural 'locational' information allows the syntactic structure to constrain the assembly of the individual semantic contributions made by words and perhaps constructions. There are however a few more points to develop or expand upon before proceeding to apply this approach to our actual examples.

In the first place, exactly where do the meaning constructors come from? The answer is that they come from the lexicon (and perhaps the PS rules), via the standard LFG process of 'instantiation'. In the lexicon, meaning-constructors are listed in 'uninstantiated' form, with functional designators such as '$\uparrow$' and '($\uparrow$ SUBJ)' in place of the f-structural tags, so that the uninstantated form of the constructors of (21) will be:

(26)

$$\begin{array}{rcl} John^e & : & \uparrow \\ Mary^e & : & \uparrow \\ Like^{e \to e \to t} & : & (\uparrow \text{OBJ}) \multimap (\uparrow \text{SUBJ}) \multimap \uparrow \end{array}$$

Instantiation is discussed in the basic LFG literature, and briefly in Andrews (2004), so needn't be considered further here.

So the production of the syntactic structure also produces a collection of uninstantiated meaning-constructors, whose $\uparrow$ (and perhaps $\downarrow$) arrows point to various c-structure nodes, which are then instantiated, and functional designators such as (f SUBJ) resolved in the usual way. This collection of instiantiated constructors must then be assembled into a tree in accordance with the rules of linear logic, of which we have so far only seen $\multimap$-elim, whose operations bear a (non-coincidental, I think) similarity to External Merge in the Minimalist Program. Linearity means that each constructor can be used only once, and there is a further constraint that all must be used, to produce a single tree whose root locates content of type t (at least for declarative sentences) at the f-structure of the whole utterance.

We are now ready to take on the *warn* example. This is a plausible proposal for its f-structure, where the dotted arc indicates that the two structures it connects are actually the same structure, linked into the the whole structure at two places:

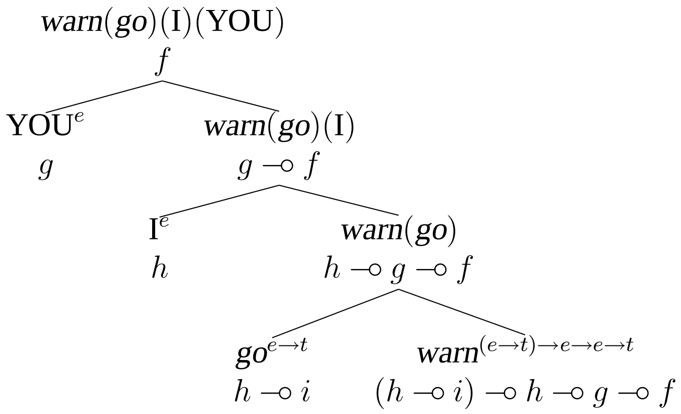
(27)
$$f{:}\left[\begin{array}{ll}
\text{SUBJ} & g{:}\left[\begin{array}{ll}\text{PRED} & \text{‘Pro’}\\ \text{PERS} & \text{II}\\ \text{NUM} & \text{SG}\\ \text{CASE} & \text{NOM}\end{array}\right]\\
\text{TENSE} & \text{PAST}\\
\text{PRED} & \text{‘Warn(SUBJ, OBJ, XCOMP)’}\\
\text{OBJ} & h{:}\left[\begin{array}{ll}\text{PRED} & \text{‘Pro’}\\ \text{PERS} & \text{I}\\ \text{NUM} & \text{SG}\\ \text{CASE} & \text{ACC}\end{array}\right]\\
\text{XCOMP} & i{:}\left[\begin{array}{ll}\text{SUBJ} & h{:}[\quad]\\ \text{PRED} & \text{‘Go(SUBJ);}\end{array}\right]
\end{array}\right]$$

and these can be its instantiated meaning-constructors, using the analysis of functional control by arguments proposed by Asudeh (2002), and using convenient abbreviations for the explications of the verbs:

(28)
$$\begin{array}{rcl}
\text{I} & : & g\\
\text{you} & : & h\\
\textit{Warn} & : & (h \multimap i) \multimap h \multimap g \multimap f\\
\textit{Leave} & : & h \multimap i
\end{array}$$

(we continue to ignore tense).

Now the full assembly will look like this in our modified tree format:

(29)


warn(*go*)(I)(YOU)
f
YOU^e
g
warn(*go*)(I)
$g \multimap f$
I^e
h
warn(*go*)
$h \multimap g \multimap f$
go$^{e\to t}$
$h \multimap i$
warn$^{(e\to t)\to e\to e\to t}$
$(h \multimap i) \multimap h \multimap g \multimap f$

As discussed by Asudeh, the double appearance of the f-structure tag h in the meaning constructor for *warn* as both antecedent of an embedded implication argument, and as an argument, allows this f-structure to in effect play two semantic roles in the structure, ‘understood subject’ of the infinitival complement, and ‘object/Warnee’ of the verb *warn.* and the NSM explications compose as required.

At this point we could say that the basic mission is accomplished: we've showed how to use LFG+Glue to compose explications for the words in the example we have been considering (and a reasonable range of similar ones). But we've so far only used the rule of $\multimap$-elim, and if this was all there was to semantic composition, simpler mechanisms than glue would be sufficient. Much of the interest in Glue is due to its ability to deal with more complex assembly issues such as quantifier scope, in which we have to relate a single NP in the syntax to two positions in the semantics, one the position appropriate to its semantic role, the other appropriate to its scope. A very simple example is afforded by the ambiguity of sentences like (30), which can have either the sense of (a) or (b):

(30) everybody didn't go
  a. it is not the case that everybody went
  b. nobody went

Under current LFG, the f-structure for this example (both readings) would be something like:

(31)
$$f{:}\left[\begin{array}{ll} \text{SUBJ} & g{:}\left[\begin{array}{ll}\text{QUANT} & \text{`Every'}\\ \text{PRED} & \text{`Pro'}\\ \text{HUMAN} & +\end{array}\right] \\ \text{POL} & \text{NEG}\\ \text{TENSE} & \text{PAST}\\ \text{PRED} & \text{`Go(SUBJ)'}\end{array}\right]$$

Like the overt phrase-structure, this f-structure doesn't represent the scope facts.

With Glue Semantics, the representation of the scope ambiguities is accomplished by the meaning-constructors associated with quantifiers, together with how the linear logic deductions work. In currently standard treatment, NL quantifiers are treated as generalized quantifiers in the sense of Barwise and Cooper (1981) (there are various alternatives under development). This means that they are viewed as predicates over intransitive verb meanings, so that their meaning type is $(e{\rightarrow}t){\rightarrow}t$. In an extensional truth-theoretic account, for example, *everybody went* is true if and only if the set of 'goers' is included in the set 'everybody', which is the set of predicates that are true of all people. More generally, one can say that they are 'properties of properties': the property *everybody* may or may not be true of the property *went*, which may or may not be true of any particular person.

The underlying assumptions of NSM are so different from those of the truth-conditional approach from which this treatment of quantifiers emerges that analyses are unlikely to transport from one to the other without a great deal of adjustment, if at all, but I think it's worth showing how Glue manages the ambiguity in order both to demonstrate some features of Glue, and also to pose some basic issues that NSM will have to deal with in order to succeed in this area. Standard constructors for the negative and quantifier would be as follows, with formal rather than substantive meaning sides:

(32)
$$\begin{array}{rcl} \textit{Neg}^{t\rightarrow t} & : & \uparrow \multimap \uparrow \\ \textit{Everybody}^{(e\rightarrow t)\rightarrow t} & : & (\uparrow \multimap H) \multimap H \end{array}$$

The *Neg* constructor applies to content located at the f-structure where the negative marker introduces the negative polarity feature, so that for example if the f-structure were:

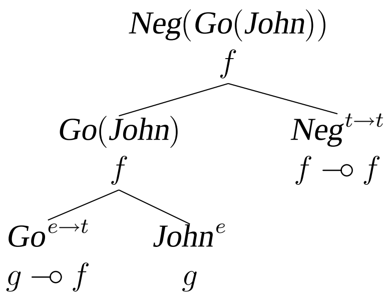
(33)
$$f{:}\begin{bmatrix} \text{SUBJ} & g{:}\begin{bmatrix}\text{PRED} & \text{`John'}\end{bmatrix} \\ \text{POL} & \text{NEG} \\ \text{PRED} & \text{`Go(SUBJ)'}\end{bmatrix}$$

then the instantiated constructors would be:

(34)
$$\begin{array}{rcl} \mathit{John}^{e} & : & g \\ \mathit{Neg}^{t\to t} & : & f \multimap f \\ \mathit{Go}^{e\to t} & : & g \multimap f \end{array}$$

and the assembly would be:

(35)



This illustrates how Glue assembly can ‘expand’ a relatively flat f-structure into a more hierarchical pattern of applications of predicates to arguments, an important resource whereby Glue can deal with some of the problems created by the relative ‘flatness’ of f-structure as opposed to overt phrase structure. The linear logic property that premises disappear when used is also useful here, for without it, we might expect the negative to apply multiple times to whatever the meaning associated with the f-structure f happened to be.

The quantifier constructor has a novel feature, the appearance of the variable H where we expect an $\uparrow$ or $\downarrow$ arrow. This variable is standardly interpreted as a universal ‘any’ quantifier in the linear logic; it would also be possible to interpret it as a freely instantiable f-structure variable, that can be identified with any other f-structure produced by the solution algorithm, or even novel one that isn’t. Remarkably, as shown in Dalrymple et al. (1997), the principles of the logic will prevent this apparently quite underconstrained scheme from producing excess readings in classical examples sucn as *every representative of a company demonstrated a product*, where naive ‘quantifier extraction’ techniques tend to produce six readings, but there are actually only five.

Returning to example (30), the only viable instantiations/(linear) universal quantifier bindings of the H variable is to f of (31), so that the instantiated meaning-constructors are effectively:

(36)
$$\begin{array}{rcl} \mathit{Everybody}^{(e\to t)\to t} & : & (g \multimap f) \multimap f \\ \mathit{Neg}^{t\to t} & : & f \multimap f \\ \mathit{Go}^{e\to t} & : & g \multimap f \end{array}$$

These have two valid glue assemblies, the first using only $\multimap$-elim, which you should be able to work out at this point, the other using $\multimap$-introduction rule (traditional Hypothetical Deduction), which we haven't discussed yet, but turn to now.

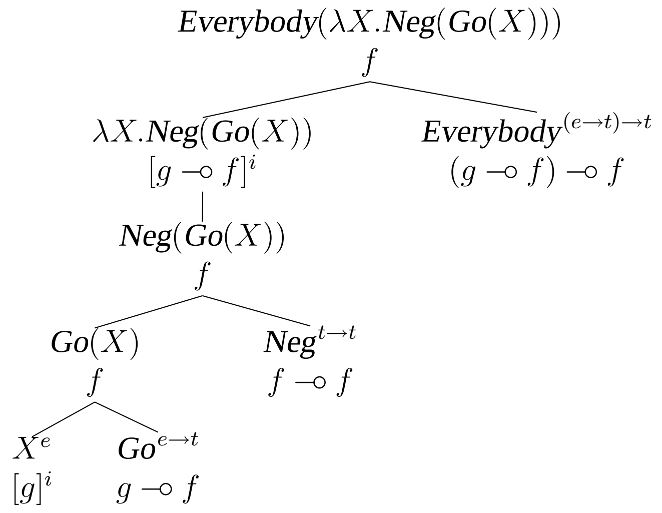
The purely logical version of this rule lets us derive a conclusion B from a 'provisional assumption' A, and then 'discharge' the assumption by deriving the conclusion $A \multimap B$, which then no longer depends on the premise A. The traditional tree formulation of this rule is:

(37)
$$\dfrac{\begin{array}{c}[A]^i \\ \vdots \\ B\end{array}}{A \multimap B}\ \multimap\text{-intr}^i$$

where the brackets represent the provisional character of the assumption A, and the superscripts the relationship between the assumption and the deduction step whereby it is discharged.

When labels are added, the provisional premise is labelled with a variable not used in any other premise, of type compatible with its formula, and in the $\multimap$-elim step, the label of the derived formula B (which will contain a free occurrence of X) is lambda-bound by X in the conclusion. Adapting this notation to the upside-down tree format in an obvious way, the derivation of the wide-scope sense of (30) can be depicted like this:

(38)


Everybody$(\lambda X.$*Neg*$($*Go*$(X)))$
f
$\lambda X.$*Neg*$($*Go*$(X))$
$[g \multimap f]^i$
Everybody$^{(e\rightarrow t)\rightarrow t}$
$(g \multimap f) \multimap f$
Neg$($*Go*$(X))$
f
Go(X)
f
Neg$^{t\rightarrow t}$
$f \multimap f$
X^e
$[g]^i$
Go$^{e\rightarrow t}$
$g \multimap f$

The last (topmost) two steps of this derivation are exactly equivalent to 'base generating' the interpreted output of quantificational NP-raising as described by Kratzer and Heim (1998), making them somewhat reminiscent of 'Internal Merge' in the Minimalist Program.

$\multimap$-elim and $\multimap$-intr constitute the 'implicational fragment' of linear logic, which is sufficient for many interesting glue analyses, although Asudeh (2004) argues for and makes use of an additional rule of 'tensor elimination' which has the effect of binding on two variables at once. Lev (2005) argues for an alternative analysis of some of these phenomena which does not require this rule and can be done within the implicational fragment, but the issues are still open.

We have now seen how Glue can represent the scope ambiguity, but what about the actual meanings? Here is an initial proposal. As frequently mentioned, *everybody* doesn't normally mean everybody in the universe, but all of the people in some restricted domain, which the speaker can be taken as thinking of. This suggests something like this:

(39) $\lambda P^{e \to t}.$ $\boxed{\text{if somebody is one of the people I am thinking about, then } P(\text{this person})}$

But it is then not so clear how we should deal with the negation. NOT is one of the currently standard primitives, but to get decent results for negative sentences under composition it often seems to be necessary to render it as something like *this is not true: Z*, yielding something like this for the wide-scope reading:

(40) if somebody is one of the people I am thinking about, then
    this is not true:
        this person did something because this person
          wanted to be somewhere else
        . . .

And similarly for narrow scope:

(41) this is not true:
    if somebody is one of the people I am thinking about, then
        this person did something because this person
          wanted to be somewhere else
        . . .

It is an awkward feature of this approach that we don't seem to be able to render English verbal negation with the NOT primitive alone, but have to drag TRUE into the explications as well. This could be taken to indicate a problem with this approach to explicating these constructions, but at least it serves to present some of the issues that NSM must deal with here.

8 Conclusion

Therefore, we see that the f-structures provide a sort of framework whereby the meanings encoded in meaning constructors for words can be assembled into meanings for complete utterances; this technique can also be extended to constructions, by associating meaning constructors with certain phrase-structure rules as well as with lexical items (Asudeh and Crouch 2002). However, although we can do this for some constructions, it is clear that many challenges remain.

As I mentioned at the beginning LFG+Glue is clearly not the only formal theory of syntax that can be used for this purpose, but the rather traditional nature of the f-structure representation means that it is potentially relatively accessible to descriptivists, while the considerable degree of abstraction from the details of overt sentence form that the f-structures provides means that the results of investigation of composition can be expected to be reasonably transportable between different syntactic frameworks as well as different languages.

Bibliography

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

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

Asudeh, A. 2004. *Resumption as Resource Management*. PhD thesis, Stanford University, Stanford CA. `http://www.ling.canterbury.ac.nz/people/asudeh.shtml` (viewed Jan 2, 2005).

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

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

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

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

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

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

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

Girard, J.-Y. 1987. Linear logic. *Theoretical Computer Science* 50:1–102.

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

Hindley, J. R., and J. P. Seldin. 1986. *Introduction to Combinators and λ-Calculus*. Cambridge University Press.

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

Klein, E., and I. Sag. 1985. Type-driven translation. *Linguistics and Philosophy* 8:163–201.

Kratzer, A., and I. Heim. 1998. *Semantics in Generative Grammar*. Oxford University Press.

Lev, I. 2005. A gentle introduction to glue (with some new results). URL: `http://www.stanford.edu/~iddolev/`.

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

Partee, B. H., and V. Borschev. 2004. Genitives, types, and sorts. In J. yung Kim, Y. A. Lander, and B. H. Partee (Eds.), *Possessives and Beyond: Semantics and Syntax*, 29–43. Amherst MA: UMass GSLA. URL: `http://people.umass.edu/partee/Research.htm`.

Pollard, C. 2001. Cleaning the HPSG garage: Some problems and some proposals. URL: `http://www.ling.hf.ntnu.no/HPSG2001/summer_school/courses.html`.

Pollard, C. 2004. Higher Order Grammar, a NASSLI 2004 course. URL: `http://www.ling.ohio-state.edu/~hana/hog/`, seen Dec 21,2004.

Restall, G. 2000. *An Introduction to Substructural Logics*. Routledge.

Szabolcsi, A. 2005. Questions about model theory, proof theory, and semantically flavored syntactic features. Presented at Fall 2005 semantics workshop at Rutgers University. URL: `http://homepages.nyu.edu/~as109/szabolcsi_model_proof_rutgers.pdf`.

Wierzbicka, A. 1972. *Semantic Primitives*. Frankfurt: Athenäum Verlag.

Wierzbicka, A. 2005. Colour and shape in language and thought and the nature of conceptual analysis. Conference on Concepts and Conceptual Analysis, Center for Consciousness, ANU Jan 20-21, 2005.

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