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 $)(\mathrm{you})$

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

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

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

reduces to the desired result (7).