Propositional Glue Semantics is Fully Variable-Free

Avery Andrews
ANU, Dec 2016
early draft (v1); please do not criticize in print

Asudeh (2006) argued that LFG with its 'glue semantics (Dalrymple 2001, Asudeh 2012) was variable-free in the sense of Jacobson (1999), but there was at least an apparent gap in the argument, which is that glue derivations are normally presented with 'meaning terms' that involve variables in their presentation. In this paper I show that if we use the 'propositional' glue[1] proposed in Andrews (2010b), a simple application of 'rudimentary' linear logic, the system is definitively variable-free, and, furthermore, bears a certain resemblance to Chomsky's external and internal Merge, for what that may be worth (possibly very litte).

Unfortunately, the discussion requires basic category theory, being a trivial application of Lambek and Scott (1986:41-81), henceforth LS. Trying to include an introduction to category theory in a linguistics paper is not realistic, but LS is in principle self-contained, and a more leisurely introduction to the required background material can be found in Barr and Wells (1999a), chs 1-6.[2] We will be reviewing some aspects of Cartesian Closed Categories (CCCs), but will assume more basic material without comment. The degree to which this material is elementary cannot be overestimated; indeed, I am puzzled as to why it seemed necessary to me to write this now, rather than having had it available to read at least fifteen years ago.

Glue semantics, without monads[3] and similar elaborations, is essentially Montague Semantics without his NL syntax, which latter is replaced by a technique for connecting the 'semantic derivation' to LFG's f-structures. To have Montague Semantics without the syntax, we need to use something like Montague's intermediate language of lambda-terms, which we here present as arrows in a free CCC. These are then mapped by a functor into whatever CCC the model theoretic semantics lives in, which by default should I think be taken to be **Sets** (the category of sets), just as in Montague's work.

In the first section, I'll discuss these two CCCs and the relationship between them, and in the second, I will show how the free CCC is closely related to another kind of category, a free 'symmetric monoidal closed category' (SMCC) which is just another way of presenting rudimentary linear logic. I will also point out that if we omit the information about the connection to NL syntax, we can use the standard linear logic rules of 'implication elimination' and 'implication introduction' as approximate equivalents to external and internal Merge, and with them generate the set of meanings of a given type that can be assembled from a multiset of lexical meanings, that is, a numeration.

[1] The first version of glue, 'old glue', is presented in many of the papers in Dalrymple (1999), including Dalrymple et al. (1999), which describes the transition to 'new glue', in its 'System F' version, which is based on a limited form of higher order linear logic, and the most widely used. Kokkonidis (2008) argued that System F could be replaced by first order linear logic, while Andrews (2010b) argued that propositional 'rudimentary' linear logic will suffice.

[2] With some obvious possible omissions, such as factorization systems and sketches.

[3] As discussed by Giorgolo and Asudeh (2012 and 2014).

I suspect that the significance of this is that Merge is not really a new idea, but an imprecise way of putting things that have been known for a long time. Then in the third section, I briefly discuss ways of connecting the semantic generation system to LFG syntax in order to get glue semantics for LFG.

1 Montague Semantics via CCCs

Richard Montague (1970a, 1970b) founded contemporary formal semantics by connecting linguistic expressions to a system of semantic types built up recursively by function-space formation from basic types such as 'entity' and 'truth value'. His technique involved considerable use of highly repetitive definitions involving variables and assignments of values to the variables (Bennett (1974) is probably the most extensive example, Partee (1976) the classic anthology). Given that Montague's work was to a considerable extent based on Lambek (1958), it is perhaps not surprising that Lambek's later work would also apply usefully to it.

1.1 Free CCCs

A CCC is a category with three things:

(1) a. A terminal object $\mathbf{1}$, such that for each object A there is a unique arrow $\mathbf{1}_A : A \to \mathbf{1}$.

b. A product, consisting of, for each pair of objects A, B, a product object $A \times B$ and two arrows (projections) $\pi^1_{A,B} : A \times B \to A$ and $\pi^2_{A,B} : A \times B \to B$ such that for any arrows $f : C \to A$ and $g : C \to B$, there is a unique arrow $<f, g> : C \to A \times B$ such that $\pi^1_{A,B} \circ <f, g> = f$ and $\pi^2_{A,B} \circ <f, g> = g$.

c. An exponential, consisting of, for all objects A, B, an exponential object $A \multimap B$[4] and an evaluation arrow ϵ^A_B such that for each arrow $h : C \times A \to B$, theres is a 'curried' arrow $\mathbf{cur}_{A,B,C}(h)$ which is the unique arrow h' from C to $A \multimap B$ such that $\epsilon^A_B \circ (h' \times \mathbf{Id}_A) = h$.

For **Sets**, any set with only one element will do as $\mathbf{1}$, the regular cartesian product with its projections and pairing operations can, and for Montague Semantics, must, be take to serve as the product, and function space formation will therefore be determined as the exponential.[5]

Lambek defines free CCCs by first setting up an appropriate kind of 'deductive system', which is a kind of graph where formulas are objects and deductions are arrows, and then using equations to group the deductive system arrows into equivalence classes, which

[4]The 'lollipop' is the linear implication symbol, an unusual choice for the exponential, but all the choices for this have some problem, and the lollipop is widely used in glue semantics, our destination.

[5]Via the adjunction relation between product and exponential in CCCs, which we won't discuss here. In more general terms, the CCC functor must be 'strict' in certain respects, but can be 'lax' in others.

obey the category and CCC axioms. The objects can also be thought of as types, and we form them by starting with some basic objects/types, and applying constructors recursively to get more. In the linguistic applications, the basic types are the basic semantics types such as e(ntity) and p(proposition),[6] and any others we wish to have. For a CCC, the type-constructors are will be binary operation symbols for 'product' and 'exponential'. We also pick an otherwise unused symbol to be **1**.

The deductive system arrows will furthermore be symbolic objects as specified in (1), with identity arrows added, and with syntactic concatenation. This is then converted to a CCC by grouping deductions (graph arrows) into equivalence classes to satisfy the axioms for categories in general and CCCs in particular. The category axioms are for the identities and associativity of concatenation, while the CCC axoms are given below.

For arrows to **1**:

(2) If $f : A \to \mathbf{1}$, then $f = \mathbf{1}_A$.

The interpretation of this is that any arrow that happens to go from A to **1** is grouped into the same equivalence class as $\mathbf{1}_A$.

For products:

(3) For all $f : C \to A, g : C \to B, h : C \to A \times B$:

a. $\pi^1_{A,B} < f, g > = f$

b. $\pi^2_{A,B} < f, g > = g$

c. $< \pi^1_{A,B} h, \pi^2_{A,B} h > = h$

One can show that these equations give the same results as the standard formulation in (1b).

And finally the exponential. The full **cur** symbol with its three subscripts is too cumbersome for convenient use, espcially when any of the types are long, so we will normally write it as a * superscript (the types can be read off from that of what it applies to). So the equations are:

(4) a. $\epsilon^A_B < h^* \pi^1_{C,A}, \pi^2_{C,A} > = h$

b. $(\epsilon^A_B < k \pi^1_{C,A}, \pi^2_{C,A} >)^* = k$

This again gives the same result as the formulation in (1b).

What is important about the free CCC on a given set of basic objects is not the particular technique that is used to construct it, but rather the Universal Mapping Property of what is constructed:

[6] t is more commonly used for things that have truth-values, but p is less suggestive of extensionality

(5) Any mapping of of the basic objects of the free CCC into the objects (basic or not) of any other CCC $\mathscr{C}$ can be extended in a unique way to a strict CCC functor from the free CCC to $\mathscr{C}$.

The reason for this is that the free CCC is constructed so as to obey no laws other than the CCC laws and that it contain the basic objects, so it contains nothing that is contradictory with the principles of any other CCC that can provide images for the basic objects. Universal Mapping Properties also have the consequence that any two systems that satisfy them are isomorphic. Therefore we can talk about *the* free CCC on a set of basic objects, and forget about the construction method, once we have satisfied ourselves that it works.

What we mean by a 'strict CCC functor' here is a functor that maps all of the CCC apparatus of the source CCC onto that of the target CCC exactly ('on the nose'). So if F is a functor from $\mathscr{C}$ to $\mathscr{D}$, then:

(6) a. $F(\mathbf{1}_{\mathscr{C}}) = \mathbf{1}_{\mathscr{D}}$ (note that the category symbol subscript here is taken to indicate the category that the **1** is the terminal object of, rather than an arrow to the terminal object).

b. $F(\mathbf{1}_A) = \mathbf{1}_{F(A)}$ (here our subscripts indicate the object that the arrow to the terminal object is from).

c. $F(A \times B) = F(A) \times F(B)$ (we could have subscripted $\times$ with category symbols here, but didn't).

d. $F(\pi^i_{A,B}) = \pi^i_{F(A),F(B)}$ for $i = 1, 2$, and $F(<f, g>) = <F(f), F(g)>$.

e. $F(A \multimap B) = F(A) \multimap F(B)$, $F(\epsilon^A_B) = \epsilon^{F(A)}_{F(B)}$,
and $F(\mathbf{cur}_{A,B,C}(h)) = \mathbf{cur}_{F(A),F(B),F(C)}(F(h))$

By contrast, a 'lax' functor maps the CCC apparatus of the source category onto things that are natually isomorphic to the specified CCC apparatus of the target, which is often good enough for the purpose at hand.

For Montague Grammar, we need a functor taking the free CCC into **Sets**; for things to work, we need to take the product apparatus strictly onto the cartesian product apparatus of **Sets** (I think), but the other aspects of the funcor can be lax: any one element set will do as value of **1**, and any right adjoint of $\times$ in **Sets** will do as the value of the exponential apparatus. The free CCC on the basic types and the CCC functor into **Sets** therefore give us a variable-free counterpart to Montague's lambda-calculus intermediate language. But there are some details in setting up interpretations that are worth looking at.

1.2 Montague Interpretation of a free CCC

If we have decided on some basic types, such as e and p, we are ready to set up an 'interpretation functor' Σ from the free CCC on those basic types to **Sets**. But, because

we're using category theory, we'll designate the types as E and P, using uppercase, since this is the convention for objects in category theory.

So our first decision is to chose Σ-images for the basic types. E is easy, any old set will do, since entailment is typically defined in terms of an arbitrary 'universe of discourse', meaning, entities that one might want to say something about. P is harder, because its image is a fixed choice embodying a great deal about what kind of semantics we are doing. For a simple extensional semantics, it can be any two-element set with one element representing 'true', the other 'false', for Montague's intensional semantics, the powerset of some set of 'possible worlds' (which would also vary with the interpretation). These are not the only possibilities.

The next thing we need to think about is interpretations for lexical items, which sends us back to fiddling with the free CCC. In **Sets**, these will be elements of the members of images of the types (basic and composite); for example the interpretation of *Rover* will be an element of $\Sigma(E)$, the intepretation of *Barks* an element of $\Sigma(E \multimap P) = \Sigma(P)^{\Sigma(E)}$. What will correspond to these in the free CCC? Happily LS:57-58 provide an answer, indeed they provide two, although only the second one seems suitable for us, because we want to add arrows from $\mathbf{1}$. In particular, to the deduction system underlying the free CCC, we add some 'assumptions', being arrows from $\mathbf{1}$, and then form a 'free CCC with indeterminates' by imposing appropriate equations on arrows. On pg. 58 an appropriate universal property is formulated and proved.

Σ will now map the assumptions onto arrows in **Sets** from $\mathbf{1}$ in that category, and, as such, they will pick out elements of the appropriate interpreted semantic types. Furthermore, if we construct a product of the 'assumptions', which we will call 'lexical indeterminates', and construct an arrow that goes from this product to P, we have something that Σ will map onto a model-theoretic interpretation. So for example if we have the lexical indeterminates of (a), below, we can apply ϵ^E_P to their product to get an arrow that Σ will map onto the truth-value/proposition expressed by *Rover barks* under the interpretation implemented by Σ:

(7) a. $Rover\colon \mathbf{1} \to E$, $Barks\colon \mathbf{1} \to E \multimap P$

b. $\epsilon^E_P(Rover \times Barks)$[7]

With this technique, we get a variable-free implementation of model theoretic semantics that furthermore lacks the extensive and repetitive boilerplate formulations of early Formal Semantics that is I think perhaps not completely gone even today. That it is truly variable-free might not be entirely clear at this point, because what will we do about wide scope readings of quantifiers? We will see how these work in the next section, when we start looking at glue. But first I want to make a few more remarks about aspects of using CCCs for formal semantics of Natural Language.

[7]In this case, we could pair the lexical arrows rather than make a product of them, but the minor convenience attained this way dissipates in more complex situations.

1.3 Aspects of Formal Semantics of NL in CCCs

The first point is that to do formal semantics of natural language in a more or less traditional way, we require two kinds of lexical indeterminates, 'fixed', and 'variable'. Montague's program for formal semantics comprised two things, the first, defining truth, the second, defining entailment. The first is I think rather dubious for, NL, for reasons such as those adduced in LePore (1983). The second is however much more straightforward (at least by comparison to the first). The idea is that a set of sentences S entails a single sentence s iff every 'interpretation' that assigns 'true' all the sentences of S also does this for s.[8] An interpretation is an assignment of model-theoretic values to items which have them, which in Montague's framework would not include the logical words, whose 'meanings' are fixed by the structure of the interpretation function. So model-theoretic semantics provides us with a mathematically precise account of entailment and other meaning-based properties and relations, which, although itself nonconstructive, can be used to provide a basis for constructive accounts via completeness theorems, and has other uses as well.[9]

But to adapt such an account to this new setting, we need to specify the interpretations of the 'logical words' as fixed, and leave the other ones to vary so as to provide a set of 'admissable interpretations' to define entailment:

(8) A set of (declarative) sentences S entails a sentence s if every admissable interpretation (choice of Σ) that assigns 'true' to the sentences of S true also assigns 'true' to s.

There are more meaning-based properties of and relations between sentences than entailment, but entailment is clearly the most important one.

Returning to lexical indeterminates, a sample fixed one would be the interpretation of *every* in accord with the now standard account of natural language quantifiers as generalized quantifiers (Barwise and Cooper 1981). Its type is $(E \multimap P) \multimap (E \multimap P) \multimap P$, and if we assume that the first '$(E \multimap P)$' applies to the nominal (the 'restriction'), and the second to the scope, the interpretation in an extensional semantics must be the function that maps the two one-place predicate arguments to 'true' if the first denotes a subset of the second, to 'false' otherwise. Sample variable arrows would be *Rover* and *Bark*, as above.

Another point is that some arrows can have their interpretation fixed as an inherent property of admissable interpretation functors, while others can be constrained by relationships that they must satisfy with respect to others, as in algebraic semantics as discussed by Link (1998).

The free CCC gives us a variable-free intermediate language whose structure is very close to that of model-theoretic interpretations, that can be mapped into those in-

[8] Note that assigning a model-theoretic 'true' value to a sentence doesn't imply making any claims about what actually makes that sentence true.

[9] Some discussion of the issues from a viewpoint largely independent of traditional formal semantics and logic can be found in Andrews (2016).

tepretations so as to define entailment (by contrast, some other variable-free systems, such as combinatory logic, look very different from **Sets**; we do not assume that any particular system is 'best', but that the more different ones are known and understood, the better). The direct categorical representations are rather forbidding, but lambda-calculus can be used as a more user-friendly notation, with the complexities of variables and the rather complicated systems for managing them and calculating alpha equivalence relegated to the status of occasionally annoying downsides of a notation that is often useful, rather than as actually constitutive of anything.

A final observation is that although this approach does restrict us to CCCs for semantic interpretation, it does not restrict us to Montague-style set-theoretic interpretations: anything that can be presented as a CCC and provides a reasonable counterpart to concepts such as 'true', 'every', etc. will suffice. The possibilities would include Pollard's (2008) 'intensional semantics', and perhaps some other systems as well, such as maybe some version of Jackendoff's 'conceptual semantics'.

On the other hand, there are significant challenges, such as what to do about dynamic phenomena such as those studied in DRT,[10] and worked out versions of formal semantics that are not obviosly instances of a CCC, such as Muskens (2007). But we will leave these topics here, and push onto the next one, which is converting free CCCs into a technique for doing free semantic composition of multisets of words, that is, 'numerations' in the sense of recent works by Chomsky.

2 Free Semantic Composition

It is obvious that the idea that the meaning of a collection of words is heavily constrained, sometimes even fully determined, by just the meanings of those words and nothing more. A collection of words such as $\{bit, Rover, John\}$ probably means *Rover bit John*, and definitely does not mean *John was rude to Susan*. But the set of words is not quite enough, the multiplicity of each word counts. For example, we cannot convert *I did not eat the last chocolate* from a denial to an 'understood confession' either by ignoring the *not*, or by interpreting it twice to produce a double negative.

What can be done with a 'multiset' of words, where each must be used exactly as many times as it appears, is part of the idea of a numeration in the Minimalist Program, but also a fundamental idea of linear logic as developed by Girard and others in the 1980s, providing the mathematical basis of glue semantics. The early versions of glue semantics look very distant from CCCs, but the distance is significantly reduced in the 'propositional glue' approach of Andrews (2010b), but is still substantial, because the 'rudimentary linear logic' (RLL; multiplicative intuitionistic linear logic without exponentials, less derisively called MILL$^-$) used in propositional glue is based on Symmetric Monoidal Closed Categories, whose definition is considerably more complex than that of CCCs.

[10] Monads as discussed in Giorgolo and Asudeh (2012 and 2014) seem relevant, but do not yet constitute a full solution.

2.1 Free SMCCs

However, I think it may have been overlooked, or at least not pointed out with sufficient clarity and emphasis for the needs of linguists, that for glue semantics, we can bypass many of the complexities of SMCCs, since all we need for glue is *free* SMCCs on a set of basic types, which are considerably simpler than general SMCCs, and closely related to the free CCCs that are useful for semantics. What we will do is form the subcategory of a free CCC that results from only recognizing some of the CCC arrows, namely, certain combinations of the projections and pairings, and their compositions. Typographically, we also replace the '$\times$' symbol with the '$\otimes$' ('tensor') symbol, since among the arrows discarded are some that are responsible for some of the essential properties of CCC products (the possibilities for 'copying' and 'erasure') and we relabel the terminal object $\mathbf{1}$ as the CCC as I, the unit of the SMCC, since it is not a terminal object, because we discard all of the arrows to it. We regard these symbolic substitutions as mere typographical cues as to what rules we are working with at the moment; the actual constitutive elements remain the same.

What we do next is define the free SMCC on the basic types as a subcategory of the free CCC on those types, having the same objects but not all the arrows. The arrows that it has are those of the smallest subcategory that contains first, these:

(9) a. For every object A, its identity arrow $\mathbf{Id}_A$.

b. For any two objects A, B, $\sigma_{A,B}: A \otimes B \to B \otimes A = <\pi^1_{A,B}, \pi^2_{A,B}>$ (the 'symmetry').

c. For any three objects A, B, C:
$\alpha_{A,B,C}: A \otimes (B \otimes C) \to (A \otimes B) \otimes C = <<\pi^1_{A,B\times C}, \pi^1_{B,C}\pi^2_{A,B\times C}>, \pi^2_{B,C}\pi^2_{A,B\times C}>$
and $\alpha^{-1}_{A,B,C}: A \otimes (B \otimes C) \to (A \otimes B) \otimes C$ (the 'associator' and its inverse; you can construct the latter explicitly from the CCC materials).

d. For any object A, $\rho_A: A \to A \otimes I = <\mathbf{Id}_A, \mathbf{1}_A>$ (the 'right unitor' for A; for glue semantics we don't actually need the unitors, but they are required for a proper SMCC; below we'll discuss a consequence of omitting them).

And then those derived from other arrows by these steps, inductively:

(10) a. For any two arrows $f: A \to B$ and $g: B \to C$, their composition $gf: A \to C$.

b. For two arrows $f: A \to B$ and $g: C \to D$, the 'tensored arrow'
$f \otimes g: A \otimes C \to B \otimes D = <f\pi^1_{A,B}, g\pi^2_{A,B}>$.

c. For any arrow $h: A \otimes B \to C$, its curry $\mathbf{cur}_{A,B,C}(h)$.

Note especially that we cannot use the projections or the arrow-pairing construction on their own anymore, but only in the combinations specified in (9), since free use would allow meanings to be ignored or used multiple times.

By defining the free SMCC as a subcategory of the free CCC on the same objects, we avoid the need to contemplate the rather complex collection of arrow equations normally used to define SMCCs, as for example in Troelstra (1992:81-86). The necessary behavior follows from the CCC principles, with some behaviors that we don't want, such as copying and deletion, excluded.

There are many things we can't do with free SMCCs, such as quantum physics, but they are all we need for compositional semantics. But there is a problem with the lexical arrows. Naively, one might think that they can simply be adjoined as arrows from I, similarly to what works for CCCs, but this turns out to destroy the resource sensitivity that we want: if we have an arrow $f : I \to A$, we can construct an arrow from I to A, $A \otimes A$, etc. This is because of the unitors. There is a construction for adjoining arrows from I to an SMCC (Hermida and Tennent 2009), but it doesn't have the properties we want.

So we need to do something a bit awkward, but not horribly so (and this is not the only way to do it). We start by taking a product of the lexical arrows in the CCC, including multiple occurrences of anything we are going to use more than once, one occurrence for each use. We can call this the 'lexical product arrow', or even 'the numeration'. Then we shift to the SMCC to find an arrow in the SMCC from the target of the lexical product arrow to P or whatever other kind of meaning we are looking for. We can call this the '(semantic) composition arrow'. Composing the image of this arrow under the embedding functor from the SMCC with the lexical composition arrow in the free CCC gives us an arrow from a product of $\mathbf{1}$'s, which is not significantly different from an arrow from $\mathbf{1}$, to an element of P or whatever we were looking for. The usual labelled deduction format can be viewed as an incremental way of representing arrow-building, whereby the deduction steps represent steps in building the SMCC arrow, the formulas the target type of the arrow constructed thus far, and the meaning terms the CCC arrow constructed thus far. The crucial step is to start with a product of arrows, rather than mere objects.

It is furthermore the case, most easily seen by the use of proof nets (Andrews (2010b) and briefly below), that if we are looking for a specific type such as P, E or $E \multimap P$, that there are a only a finite number of different such arrows, since each arrow is represented by a pairing of atomic symbols in formulas subject to the proof-net constraints, and there are only a finite number of such pairs (but if we don't have a finite list of types of result we will accept, there are no longer a finite number of assemblies, as we discuss below).

So we now have something that functions in a way analogous Chomsky' numeration with Merge, except that the semantics is inherently part of it, and effectively does the work that features do to control the application of Merge. A practical use of the system (which could probably be done with Merge as well) is to find what sense can be made of a multiset of words on the basis of their semantic types alone, ignoring syntax. It is then a cleaner (and more powerful) version of the functional realization mechnanism of Klein and Sag (1985). It could be helpful, and perhaps something like it even used by humans, in language-learning situations where the syntax is not (very

well) known, but the meanings are known well enough, and context provides a filter on possible interpretations.

2.2 Arrows from Tree-style ND

It has been known for a long time that free SMCCs are equivalent to various ways of doing rudimentary linear logic deductions (showing this is presented as a 'straightforward exercise' in Troelstra (1992:86)), so we can in fact continue to do them with our favorite method without worrying about categories, but nevertheless, I think it would be good to see how to construct the SMCC arrows we want by doing conventional glue proofs, since these do provide a relatively easy-to-use construction method. We will look at tree-style natural deduction, which is the system most often used in the literature, e.g. Dalrymple (2001) and Asudeh (2005a, 2012), for the most part restricted to the implication rules.

Tree-style ND proofs are normally presented as 'labelled deductions', where the label appears to the left of a colon in the formulas being proved, and, in our present framework, represents the CCC interpretation of the arrow constructed so far, and to the right of the colon appears a type, also constituting the proposition being proved from the assumptions (types of the lexical items in the enumeration).

The two basic rules are:

(11)
$$\frac{f : A \multimap B \qquad a : A}{f(a) : B} \multimap \text{elim} \qquad \frac{\begin{array}{c}[x : A]^i \\ \vdots \\ m : B\end{array}}{\lambda x.m : A \multimap B} \multimap \text{intr}^i$$

In spite of unusual-looking lollipop, these are the normal rules of implication elimination (Modus Ponens) and implication introduction (Hypothetical Deduction); what is different is that the environment they are functioning in, linear logic, which is very well suited to the tree-style presentation, since the requirement that each premise be used once and once only can be naturally implemented by requiring that the premise instances appear as leaves of a tree, with the nodes and their branchings determined by the deduction rules. Here is an example, where it is assumed that transitive verbs are curried, and the arguments applied in order of increasing agency:

(12)
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathit{Loves} : E \multimap\ E \multimap\ P \qquad [x : E]^1}{\mathit{Loves}(x) : E \multimap\ P} \multimap \text{elim} \qquad \mathit{Rover} : E}{\mathit{Loves}(x)(\mathit{Rover}) : P} \multimap \text{elim}}{\lambda x.\mathit{Loves}(x)(\mathit{Rover}) : E \multimap\ P} \multimap \text{intr}^1 \qquad \mathit{Everybody} : (E \multimap\ P) \multimap\ P}{\mathit{Everybody}(\lambda x.\mathit{Loves}(x)(\mathit{Rover})) : P} \multimap \text{elim}$$

The index i in the rule is to be instantiated with a different number or similar uniquely

identifying tag for each use of the rule in the deduction tree. Note the opposite conventions about types versus function application: for the types, we omit rightmost parentheses including outermost, for function application, leftmost, so that, for a 2 argument function, the function plus the argument that appears first is construed as a function that applies to the argument that appears second, both arguments in parentheses of their own (the original Montague convention).

This format of derivation pushes the categorical machinery under the rug/behind the drywall; our purpose here is to pull it out and look at it (very likely, only once!). To work through it, we need evaluation, currying and the symmetry and associators (unitors appear to play no useful role, and their availability perhaps can even cause trouble, as we discuss below).

To begin with, it is relatively easy to see what the $-\!\circ$ elim rule does: it applies evaluation. So if we start with $Barks \times Rover$ in the free CCC, this gives us a very simple derivation that we could represent like this:

(13) $$\frac{Barks : E -\!\circ P \qquad Rover : E}{\epsilon^E_P(Barks \times Rover) : P}$$

We don't need the justifications anymore, because the existence of the arrows is shown by their actual appearance in the conclusions of the steps, to the left of the semicolon. Note that since all the SMCC arrows are available in the CCC, we can code the SMCC/CCC steps into the left sides, while leaving the right only designating the target of the arrow constructed so far.

A more complicated derivation would be:

(14) $$\frac{\dfrac{Loves : E -\!\circ E -\!\circ P \qquad Susan : E}{\epsilon^E_{E-\circ P}(Loves \times Susan) : E -\!\circ P} \qquad Rover}{\epsilon^E_P(\epsilon^E_{E-\circ P}(Loves \times Susan) \times Rover) : P}$$

An initial question is: what if the assumptions are organized wrong? The arrow produced above will work on assumptions organized as in (a) but not (b) below:

(15) a. $(Loves \times Susan) \times Rover$

b. $(Rover \times Susan) \times Loves$

The answer is that the symmetry, associators and identities (the 'rearrangement arrows') can always be used to get the premises into the right organization. There is in fact a celebrated theorem, MacLane's Coherence theorem, that says that this can always be done in a unique way. It would be a reasonable 'finger exercise' to write out the combination of associator, symmetry and identity arrows to get the target of the (b) arrow in (15) to match the source of the arrow in (14) (identities are needed to operate on objects that are embedded inside tensors, for example $\mathbf{Id}_A \otimes \sigma_{B,C}$ goes from $A \otimes (B \otimes C)$ to $A \otimes (C \otimes B)$).

The rearrangement arrows also need to be used inside the glue derivation, when we use $-\!\circ$ intr to get the effects of bound variables, without actually having the latter. This works by introducing and then ultimately in effect removing an identity arrow:

(16)
$$\dfrac{\begin{array}{c}\ldots[\mathbf{Id}_A : A]^i\ldots \\ \vdots \\ \phi : B\end{array}}{(\phi \circ \xi)^* : A -\!\circ B}\, i$$
where ξ is rearrangement determined as discussed below

The '...'s flanking the introduced assumption represent the other premises used to derive the arrow ϕ with target B. But to use this rule, we will usually need to use the rearrangement facilities to in effect get the A input to the outer right edge of the source of ϕ.[11]

For a simple example, consider the composition of the meaning of *Rover loves everybody*, assuming that *everybody* is of semantic type $(E-\!\circ P)-\!\circ P$. We start by composing an identity arrow and the $Rover : E$ with $Love : E-\!\circ E-\!\circ P$:

(17)
$$\dfrac{\dfrac{Love : E-\!\circ E-\!\circ P \qquad [\mathbf{Id}_E : E]^i}{\epsilon^E_{E-\!\circ P}(Love \times \mathbf{Id}_E) : E-\!\circ P} \qquad Rover : E}{\epsilon^E_P(\epsilon^E_{E-\!\circ P}(Love \times \mathbf{Id}_E) \times Rover) : P}$$

This is an arrow from $((E-\!\circ E-\!\circ P) \otimes E) \otimes E$ to P, but to get apply the curry step as we want to we need to get the inner E to the outer right edge, which first applying $\alpha_{E-\!\circ\, E-\!\circ\, P,E,E}\sigma_{E,E}\alpha^{-1}_{E-\!\circ\, E-\!\circ\, P,E,E}$, which will be ξ in (16) above. So the curry step is:

(18)
$$\dfrac{\dfrac{\dfrac{Love : E-\!\circ E-\!\circ P \qquad [\mathbf{Id}_E : E]^i}{\epsilon^E_{E-\!\circ P}(Love \times \mathbf{Id}_E) : E-\!\circ P} \qquad Rover : E}{\epsilon^E_P(\epsilon^E_{E-\!\circ P}(Love \times \mathbf{Id}_E) \times Rover) : P}}{(\epsilon^E_P(\epsilon^E_{E-\!\circ P}(Love \times \mathbf{Id}_E) \times Rover)\alpha_{E-\!\circ\, E-\!\circ\, P,E,E}\sigma_{E,E}\alpha^{-1}_{E-\!\circ\, E-\!\circ\, P,E,E})^* : E-\!\circ P}\, i$$

Now we have something that *Every* can apply to yield the desired meaning. One could derive a precise algorithm to read off ξ from the position of the discharged premise amongst the other premises that lead to the discharge step, but this is not necessary, since the system here is not constitutive, but only aspires to be helpful. Suites of operations to assist with concisely specifying rearrangements of premises can be found in Mackie et al. (1993) and Biermann (1994).

There are two additional rules in the RLL system, the introduction and elimination rules for tensors. Both are little used, and the introduction rule can be avoided entirely if we assume that all arguments are curried. It looks like this as an arrow-building rule:

[11] But in another formulation of Natural Deduction, sequent-style ND, this is done more explicitly.

(19) $$\frac{f : A \qquad g : B}{f \otimes g : A \otimes B}$$

We can see that if we decide that the type of transitive verbs should be $(E \otimes E) \in P$, then we can use (19) to develop the two arguments independently and then combine them together into something to apply the verb to.

The elimination rule is more complex, and looks like this as a regular ND rule:

(20) $$\frac{\begin{array}{cc}[a, b] : A \otimes B & \begin{array}{c}[x : A]^i \; [y : B]^j \\ \vdots \\ z : C\end{array}\end{array}}{\textbf{let } a, b \textbf{ be } x, y \textbf{ in } z : C} \otimes\text{elim}^{i,j}$$

To see how this rule is used and what the mysterious-looking **let**-statement is supposed to be doing, consider the assembly below for *John shaves himself*. The idea is that the pronoun makes a copy of its antecedent meaning and puts one in its own position, the other in that of its antecedent (details and motivation in Asudeh (2012)):

(21) $$\frac{\dfrac{\lambda x.[x, x] : e \multimap (e \otimes e) \qquad \mathit{John} : e}{[\mathit{John}, \mathit{John}] : e \otimes e} \qquad \dfrac{\dfrac{\mathit{Shave} : e \multimap e \multimap p \qquad [x : e]^1}{\mathit{Shave}(x) : e \multimap p} \qquad [y : e]^2}{\mathit{Shave}(x)(y) : p}}{\mathit{Shave}(\mathit{John})(\mathit{John})} \otimes\text{elim}^{1,2}$$

So we can see is that what the **let**-statement expresses is simultaneous replacements in a formula, an inherently apparently complex operation.

The categorical version of this rule allows us to replace this complex operation with inherently simpler-looking steps, rearrangements with the symmetry and associators. Given the form of the left hand portion of the rule, it would seem easiest if we assumed that the right side is rearranged to the form $(A \otimes B) \otimes C$, where A ane B are the two items we are going to substitute for, C the remaining premise. Mac Lane's theorem tells us that we can always rearrange the premises into this organization; $\alpha_{E \multimap E \multimap P, E, E} \sigma_{E \multimap E \multimap P, E \otimes E}$ will do the job in the present case. In general, there will always be a unique rearrangement (arrow composed only of α's and σ's) $\tau^{i,j}$ and a type S^{-i-j} whose source type is $(A \otimes B) \otimes S^{-i-j}$ and whose target type is the source type of f. Therefore we can reformulate (20) as:

(22) $$\frac{\begin{array}{cc}g : A \otimes B & \begin{array}{c}\ldots [x : A]^i \; [y : B]^j \ldots \\ \vdots \\ f : C\end{array}\end{array}}{f\tau^{i,j}(g \otimes \mathbf{Id}_{S^{-i-j}}) : C}$$

The multi-substitutions are thus replaced by mere rearrangements.

We can therefore build the arrows we need for glue semantics using the familar method of ND tree derivation (as well as various others), and it is also straightforward to show that this method as well as various others can build all of them. So what we have done is demonstrate that the appearance of variables in the glue terms is a mere convenience.

2.3 Notes on Proof-nets, Units, and Incomplete Assembly

Before going on to glue semantics itself, there are some related topics that people might or might not want to look into.

One of the techniques for doing RLL proofs as a rather interesting one called 'proof-nets'. To discuss proof-nets we first need to introduce the concept of 'sequent': a sequent is a statement that a collection of premises leads to one or more conclusions; the premises are written to the left of a symbol such as $\vdash$, the conclusion(s), to the right. Fortunately, semantic assembly requires only single-conclusion ('intuitionistic') sequents, so we don't have to worry about the multiple-conclusion ones. So a sequent for a very simple assembly would be:

(23) $\mathit{Barks} : E \multimap P, \mathit{Rover} : E \vdash P$

Several proof methods, such as sequent-style ND, and 'Gentzen sequent proof', use direct manipulation of sequents, but proof-nets are yet another method, due to Girard, where you can represent a proof merely by connecting the atomic formula letters (often called 'literals') of a sequent in pairs, according to certain rules. The proof-net proof of (23) for example is:

(24) $\mathit{Barks} : E \overset{\frown}{\multimap} P, \mathit{Rover} : E \vdash P$

The rules for proof-nets are discussed at considerable length in Andrews (2010b), and, less technically, in Andrews (2010a), and won't be discussed here, but what I want to point out is that they provide at least two things:

(25) a. A clear demonstration that there are only a finite number of essentially different RLL proofs of a given sequent (including zero, if the sequent is invalid).

b. A very efficient way to represent partial assemblies: all we need to do is put in some but not all of the links, and do not even need to include the conclusion.

Point (b) becomes relevant when we consider the how meanings might function in online speech processing: words come in, we don't necessarily have all of the syntax, but we want to put things together as best we can, without wasting any available information, of which the semantic types of the words provide a certain amount. This acquires a bit more bite when we move onto the next topic, the SMCC units.

The proof-net technique works well, and is simple and efficient unless any of the formulas (premises or conclusion), contains a unit, whereupon things get very difficult:

proof-nets with units have been the topic of more than one PhD thesis, and there still doesn't seem to be any fully satisfactory solution, although there are workable ones. Fortunately for us, for NL semantics we don't seem to need them: there is no clear use for constructors of the form $I \multimap A$, $A \multimap I$, etc. But they do play a rule in certain intermediate steps of deductions, which can be problematic.

Suppose for example that we had a semantic type of the form $(A \multimap A) \multimap B$. Given this, we can prove either of the two sequents below:

(26) a. $(A \multimap A) \multimap B \vdash B$

b. $A \multimap A, (A \multimap A) \multimap B \vdash B$

(b) not disturbing: it resembles assemblies/proofs involving quantifiers, except that there is only one basic type involved instead of two.[12] (a) is another matter. We can prove it because we can assume A, and then discharge A immediately to derive $A \multimap A$, and then use that to satisfy the implicational argument position of $(A \multimap A) \multimap B$. Mathematicians and logicians want $A \multimap A$ to be a theorem, but in linguistics, it seems to make a bit of a mess. And having it be a choice to include $A \multimap A$ among the premises or not is even worse.

In categorical terms, this unwanted proof involves the unitor: because we have the arrow $\rho_A^{-1} : A \otimes I \to A$, we also have the arrow $(\rho_A^{-1})^* : I \to A \multimap A$. If we did not include the unitors in the category formed by dropping arrows from the free CCC, this possibility would be blocked, along with various other useless ones, such as deductions formed by tensoring $B \multimap B$ onto the conclusion of anything, for any B:

(27) a. $\vdash B \multimap B$

b. $A \vdash A \otimes (B \multimap B)$

c. $E, E \multimap P \vdash P \otimes (E \multimap E)$

Dropping the unitor and its inverse is essentially the same thing as forbidding deduction from zero premises. Such a system seems to work for semantic assembly, but does not seem desirable to logicans and mathematicians, who want theorems such as (a) above. Perhaps the system without unitors could be called 'Excessively Rudimentary Linear Logic'.

It would be a strong point in favor of ERLL if it blocked all useless possible conclusions from a collection of premises, but it doesn't. Here is another kind of structure that can be produced in the conclusion of a sequent without any specific warrant from the premises:

(28) $A \vdash B \multimap (B \otimes A)$

[12] And so might arise in practice if we had only one basic type (Partee 2006).

The trick used here can be repeated to produce conclusions of arbitrary complexity from a single premise A. This particular one could be blocked by forbidding the use of tensors taking inputs (the ones not used in current glue), but there would still be no proof that further unwanted forms of conclusion don't exist. So ERLL on its own doesn't constitute a full solution to this problem; perhaps something more general can be worked out and proved sufficient.

3 Glue Semantics

We now have a system for producing semantic assemblies of multisets of lexical items; to turn this into glue semantics, we need to connect this to syntactic structure, so that grammar can constrain assembly. The technique can be regarded as subdividing the type system, so that syntactic locations are also part of it. The essential idea is to 'enhance' lexical arrows to become 'meaning-constructors', which contain not only the semantic and type information, but also information about the relationship to syntax.

However there are a number of variations in how this can be accomplished. There seem to be at least two major ones:

(29) a. To use or not to use a semantic projection.

b. To use 'co-description', or 'description by analysis' (DBA).

Most glue work, including Dalrymple (2001) and Asudeh (2012), has used a semantic projection and co-description. The semantic projection is a system of feature-structures proceeding off f-structures, typically with its own attributes, while co-description is the tactic of including the meaning-constructors in full (inflected) lexical entries, using LFG's ↑-arrows and instantiation to connect them to the f-structure.

On such an account, the full lexical entries, including the meaning-constructors for *Alfie chuckled* are given as (30) below, roughly following Asudeh (2012:82) but including tense:

(30) a. $\mathit{alfie} : (\uparrow\text{PRED}) = \text{'Alfie'}, \mathit{Alfie} : \uparrow_{\sigma_e}$

b. $\mathit{chuckled} : (\uparrow\text{PRED}) = \text{'Chuckle(SUBJ)'}, (\uparrow\text{TENSE}) = \text{PAST},$
$\mathit{Chuckle} : (\uparrow\text{SUBJ})_{\sigma_e} \multimap \uparrow_{\sigma_t}, \mathit{Past} : \uparrow_{\sigma_t} \multimap \uparrow_{\sigma_t}$

The full lexical entries for these words produce the following f-structure for this sentence with its semantic projection (left out of Asudeh's presentation), whose (disconnected) substructures I have tagged as s_g and s_f:

(31)

$$\begin{bmatrix} \text{SUBJ} & \left[\text{PRED} \quad \text{'Alfie'}\right] \\ \text{PRED} & \text{'Chuckle'} \\ \text{TENSE} & \text{PAST} \end{bmatrix} \quad \begin{matrix} [\quad]{:}s_g \\ [\quad]{:}s_f \\ \\ \end{matrix}$$

The three constructors therefore become instantiated as follows:

(32) a. $\mathit{Alfie} : s_{g_e}$

b. $\mathit{Chuckle} : s_{g_e} \multimap s_{f_t}$

c. $\mathit{Past} : s_{f_t} \multimap s_{f_t}$

We can think of this as expanding the original basic semantic types to pairs, one member of which is a basic semantic type, the other a location in the semantic projection:

(33) a. $\mathit{alfie} : <s_g, e>$

b. $\mathit{chuckle} : <s_g, e> \multimap <s_f, t>$

c. $\mathit{Past} : <s_f, t> \multimap <s_f, t>$

The full pair-types are what is used for the construction of glue arrows, and the structural component is discarded in the mapping to the free semantics CCC.

The main issue with the semantic projection is lack of motivation. Most of the glue literature contains none whatsoever; Asudeh (2005b) offered an argument that was countered by Andrews (2010b), who proposes instead that meaning-constructors link directly to f-structures. There is clearly no problem in principle with this, and it also would appear to be the default hypothesis from the point of view of simplicity. But more recently, Asudeh et al. (2014) have presented an account of linking theory wherein an enriched version of the semantic projection plays the role of argument structure (or is it perhaps the other way around?). If this wins out over any competitors, or no comparably worked out competitors are developed, then we can regard the semantic projection as empirically motivated.

The issue of co-description versus DBA glue is trickier. The idea of DBA[13] is that meaning-constructors are introduced/licensed on the basis of f-structure configuations, without reference to conventional lexical items. Andrews (2007, 2008) proposes DBA glue on primarily architectural and esthetic grounds, and the glue implementation that existed in the XLE[14] for a while also used DBA, but this appears to have been a technologically rather than theoretically motivated decision (Richard Crouch p.c., and folklore).

Andrews' 2008 version of DBA glue works by having a 'Semantic Lexicon', whose entries (SLEs) specify both meaning-constructors and f-descriptions. Here the f-descriptions are represented graphically, and the connections by dotted lines, going directly to f-structure, with no semantic projection:

(34) $\left[\text{PRED} \quad \text{'Watch'}\right] \cdots\Leftrightarrow\cdots e \rightarrow t : \lambda x.\mathit{Watch}(x)$

[13] See Dalrymple et al. (1995:275-276, 283-284) for some general discussion of description by analysis.
[14] Xerox Linguistic Environment; a powerful implementation of most of LFG theory.

If this SLE is used, the PRED-feature value 'Watch' in the f-structure is marked as 'used', or 'checked off', so that no SLE cannot be introduced again on the basis of this occurrence of the feature-value. This version of DBA was thought of in conventional LFG terms as 'generating' the c-structure and computing the more abstract levels from that.

The proposal of Andrews (2007), intended for use in an Optimality Theoretic version of LFG, uses the same format of SLE, but the conceptualization of what they do is somewhat different: we might call it 'co-generative'. In that approach, you use the SLE's and glue assembly to produce a pair of f-structure and meaning-assembly, and then you use regular OT-LFG to produce a matching f-structure. Instead of checking off, there is a stipulation that in this process, features don't unify (at least by default; stipulative variation in both mechanisms are easily envisioned). This idea can be adapted to non-OT LFG, by simply using regular LFG to produce c-structure f-structure pairs, with the f-structures produced by the two 'generative' processes required to match (matching involves some issues, such as the fact that certain features, such as syntactic case, would appear to have to be ignored for matching).

It seems to me that it will be hard to find a clear empirical difference between co-descriptional and DBA glue (alfter almost ten years, there is still nothing that looks very solid), and that the co-generative version of DBA will be even harder to distinguish from the other one, and is probably just a matter of preferred conceptualization. But the co-generative conception does lead to an architecture that is not so different from the Minimalist Program, in that there is a semantic level (glue assembly) connected to an overt performance level (c-structure) by meaning of an interface mechanism. The details are very different, but perhaps a greater comparability of architecture can make it easier to identify empirically significant differences.

4 Conclusion

So this story takes as as far as 1970s Montague Grammar, but of course now there is more, staring with the phenomena addressed in the Discourse Representation Theory and its variants and competitors that appeared at the end of the 1970s, which may or may not fit in to a CCC-based approach. Monads as discussed for example in Giorgolo and Asudeh (2012 and 2014) are helpful, but are not yet a complete solution. I also have a suspicion that another kind of mathematical device called 'coalgebras' might be needed, since these are good for thinking about things, such as input-output streams and conversations, for which termination is not an intrinsic aspect of their nature (whereas, a sentence, in order to function as part of a conversation, needs to be thought of as inherently finite in nature).

I close with two final observations. The first is that this general approach resembles that of Chomsky in that linear order is essentially irrelevent. By contrast, Lambek has proposed two somewhat similar approaches, categorial grammar, and pregroup grammar, which are fundamentally based on the idea that syntactic structures are linearly ordered. This is a possibly profound difference into which we may some day get some real empirical evidence. The other is that pregroup grammar is associated with the

use of compact closed categories, which are a kind of 'opposite' to cartesian closed categories. A significant attraction of the compact closed categories is the possibility of doing a kind of 'compositional distributional semantics' based on (linear algebraic) tensor algebra, and it seems plausible that this could be done off the glue SMCC as a kind of independent branch of the interpretation of the semantic structure. Unfortunately, it is not all clear to me, from the kind of literature I have seen, exactly what compositional distributional semantics is supposed to accomplish, but the field is moving fast, so perhaps something helpful will appear before too long—it often takes time for intuitions that people are pursuing to get presented in more widely intelligible forms.

References

Andrews, A. 2016. Reconciling NSM and formal semantics. *Australian Journal of Linguistics* 36:79–111.

Andrews, A. D. 2007. Generating the input in OT-LFG. In J. Grimshaw, J. Maling, C. Manning, and A. Zaenen (Eds.), *Architectures, Rules, and Preferences: A Festschrift for Joan Bresnan*, 319–340. Stanford CA: CSLI Publications. URL: `http://AveryAndrews.net/Papers` (Recent Publications (Refereed)).

Andrews, A. D. 2008. The role of PRED in LFG+glue. In M. Butt and T. H. King (Eds.), *The Proceedings of the LFG '08 Conference*, 46–76. Stanford CA: CSLI Publications. `http://cslipublications.stanford.edu/LFG/13/lfg08.html` (accessed 19 Feb 2010).

Andrews, A. D. 2010a. 'Grammatical' vs. 'lexical' meaning constructors for glue semantics. In Y. Treis and R. D. Busser (Eds.), *Selected Papers from the 2009 Conference of the Australian Linguistic Society*. The Australian Linguistic Society. URL: `http://www.als.asn.au/proceedings/als2009.html`, also `http://AveryAndrews.net/Papers` (Recent Publications (Refereed)).

Andrews, A. D. 2010b. Propositional glue and the correspondence architecture of LFG. *Linguistics and Philosophy* 33:141–170.

Asudeh, A. 2005a. Control and resource sensitivity. *Journal of Linguistics* 41:465–511.

Asudeh, A. 2005b. Relational nouns, pronouns and resumption. *Linguistics and Philosophy* 28:375–446.

Asudeh, A. 2006. Direct compositionality and the architecture of LFG. In M. Butt, M. Dalrymple, and T. H. King (Eds.), *Intelligent Linguistic Architectures: Variations on Themes by Ronald M. Kaplan*, 363–387. Stanford, CA.

Asudeh, A. 2012. *The Logic of Pronominal Resumption*. Oxford University Press.

Asudeh, A., G. Giorgolo, and I. Toivonen. 2014. Meaning and valency. In *Proceedings of the LFG14 Conference*, 68–88. CSLI Publications.

Barr, M., and C. Wells. 1999a. *Category Theory for Computing Science*. Montréal: Centre de Recherches Mathématiques. 3rd edition. URL: `http://www.math.mcgill.ca/triples/Barr-Wells-ctcs.pdf`. The most essential content for linguists, but no exercises, can be found in Barr and Wells (1999b).

Barr, M., and C. Wells. 1999b. Category theory lecture notes for ESSLLI. URL: `http://www.let.uu.nl/esslli/Courses/barr-wells.html`. a condensed version, without exercises, of Barr and Wells (1999a). It is impressively well targeted on material that seems likely to be useful to linguists.

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

Bennett, M. R. 1974. *Some Extensions of Montague Fragment of English*. PhD thesis, University of Los Angeles, Los Angeles CA.

Biermann, G. M. 1994. *On Intuitionistic Linear Logic*. PhD thesis, Wolfson College, Cambridge University, Cambridge.

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., R. M. Kaplan, J. T. Maxwell, and A. Zaenen (Eds.). 1995. *Formal Issues in Lexical-Functional Grammar*. Stanford, CA: CSLI Publications. Retrieved Nov 15, 2010, from `http://standish.stanford.edu/bin/detail?fileID=457314864`.

Giorgolo, G., and A. Asudeh. 2012. (M,η,$\star$) monads for conversational implicatures. In A. A. Guevara, A. Chernilovskaya, and R. Nouwen (Eds.), *Proceedings of Sinn und Bedeutung 16, Volume 1. MIT Working Papers in Linguistics*, 265–278.

Giorgolo, G., and A. Asudeh. 2014. Monads as a solution for generalized opacity. In *Proceedings of the EACL 2014 Workshop on Type Theory and Natural Language Semantics (TTNLS)*, 19–27.

Hermida, C., and R. D. Tennent. 2009. Monoidal indeterminates and categories of possible worlds. *Theoretical Computer Science* 249:39–60.

Jacobson, P. 1999. Towards a variable-free semantics. *Linguistics and Philosophy* 22:117–184.

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

Kokkonidis, M. 2008. First order glue. *Journal of Logic, Language and Information* 17:43–68. First distributed 2006; URL: `http://citeseer.ist.psu.edu/kokkonidis06firstorder.html`.

Lambek, J. 1958. The mathematics of sentence structure. *American Mathematical Monthly* 65:154–170.

Lambek, J., and P. J. Scott. 1986. *Introduction to Higher Order Categorical Logic*. Cambridge University Press.

LePore, E. 1983. What model theoretic semantics cannot do. *Synthese* 54:167–187.

Link, G. 1998. *Algebraic Semantics in Language and Philosophy*. Stanford CA: CSLI Publications.

Mackie, I., L. Román, and S. Abramsky. 1993. An internal language for autonomous categories. *Applied Categorical Structures* 1:311–343.

Montague, R. 1970a. English as a formal language. In B. Visentini (Ed.), *Linguaggi nella Società e nella Tecnica*, 189–224. Milan: Edizioni di Communità. reprinted in Thomason (1974), pp. 188-221.

Montague, R. 1970b. Universal grammar. *Theoria* 36:363–398. reprinted in Montague (1974).

Montague, R. 1974. *Formal Philosophy: Selected Papers by Richard Montague*. New Haven CN: Yale University Press. edited by Richmond Thomason.

Muskens, R. 2007. Intensional models for the theory of types. *Journal of Symbolic Logic* 72:98–118.

Partee, B. (Ed.). 1976. *Montague Grammar*. New York NY: Academic Press.

Partee, B. H. 2006. Do we need two basic types. URL: `http://people.umass.edu/partee/docs/`.

Pollard, C. 2008. Hyperintensions. *Journal of Logic and Computation* 18:257–282. URL: `http://www.ling.ohio-state.edu/~pollard/cvg/hyper.pdf`27 Jan 2015.

Thomason, R. 1974. *Formal Philosophy: Selected Papers by Richard Montague*. New Haven CN: Yale University Press.

Troelstra, A. S. 1992. *Lectures on Linear Logic*. Stanford CA: CSLI Publications.