MULTIDIMENSIONAL SEMANTICS WITH UNIDIMENSIONAL GLUE LOGIC

Gianluca Giorgolo and Ash Asudeh
Carleton University Carleton University &
University of Oxford

Proceedings of the LFG11 Conference

Miriam Butt and Tracy Holloway King (Editors)

2011

CSLI Publications

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

Abstract

We present an implementation of multidimensional semantics in Glue Semantics. Our approach differs from the proposal of Arnold and Sadler (2010) by restricting multidimensionality to the meaning language and therefore avoiding the introduction of tensors in the compositional glue logic. We use a construction from category theory — monads — to create structured mappings from the algebra of unidimensional semantics to the multidimensional case. Monads have already been successfully employed in theoretical computer science to provide a denotational semantics of side effects. Here we follow the suggestion of Shan (2001) to use monads to model semantic phenomena and show how monads can be used to capture the analysis of natural language expressions like appositives and expressives. We argue that monads allow us to keep the simplicity of unidimensional composition while also allowing the ability to track multiple meaning dimensions and to control information flow between these different dimensions appropriately.

1 Introduction

Recently much attention has been paid to the semantic contribution conveyed by a diverse group of expressions that includes *expressives*, *appositives*, *epithets* and *non-restrictive relative clauses*. The contribution to meaning of this type of expressions regularly escapes the scope of logical operators such as negation and question forming elements. Consider for instance (1):

(1) Most fucking neighbourhood dogs pee on a damn hydrant on this street.

This sentence conveys the information that the majority of the dogs living in the neighbourhood urinate on a hydrant in the contextually defined street. However, the sentence also conveys a generally negative attitude towards dogs and/or their urinating on the aforementioned hydrant. This effect is obtained in (1) by the use of the two expressives *fucking* and *damn* (compare (1) with the more neutral *Most neighbourhood dogs pee on a hydrant on this street*).

Nevertheless, the resulting interpretation is not just the conjunction of the two contributions. If an interlocutor replies to (1) with *No, that's not true*, the interpretation commonly associated with this reply can be paraphrased as "No, the neighbourhood dogs don't pee on a hydrant on this street". The reply does not negate the semantic contribution of the expressive. The same reply to something along the lines of *Most neighbourhood dogs pee on a hydrant on this street and I hate dogs and their urinary habits* or *Most neighbourhood dogs pee on a hydrant on this street and dogs and their urinary habits are detestable* would instead take scope over both conjuncts, thus also potentially targeting the negative attitude towards dogs.

The mini dialogue in (2) exemplifies the same behavior.

[†]This research is supported by an Early Researcher Award from the Ontario Ministry of Research and Innovation and NSERC Discovery Grant #371969. The authors thank Doug Arnold, Mary Dalrymple, Dag Haug, Louisa Sadler and the audience at LFG11 for their comments and questions.

(2) A: John Lee Hooker, the bluesman from Tennessee, appeared in *The Blues Brothers*.
B: *No, that's not true.*
$\Rightarrow$ No, John Lee Hooker did not appear in *The Blues Brothers*.
$\not\Rightarrow$ No, John Lee Hooker was not from Tennessee.

B's reply does not target the information about the birth place of John Lee Hooker, conveyed by A with the appositive *the bluesman from Tennessee*. The only strategy available to B to correct A's utterance is the one illustrated in (3):[1]

(3) B: True, but actually John Lee Hooker was born in Mississipi

In (3) the information conveyed by the main clause is first acknowledged and only then the appositive contribution is amended.

Potts (2005, 2007) introduces a unified analysis of these diverse expressions in terms of two parallel semantic levels, usually called 'dimensions'. According to this view, there are two different dimensions of meaning to which expressions can contribute, the 'at-issue' dimension and the 'side-issue' dimension, also called the 'conventional implicatures dimension' or 'CI dimension'.

Contributions to the *at-issue* dimension represent that part of meaning that speakers present as 'under discussion'. At-issue contributions are sensitive to logical operators and, in a communicative setting, they enter the common ground only after being (possibly silently) acknowledged by the other communicative agents. In (1) the at-issue contribution corresponds to the information about the urinary habits of the neighbourhood dogs.

Expressions contributing to the CI dimension mainly convey information that the speaker presents as *uncontroversial*. Moreover the information is presented as *peripheral* and not *under discussion*. Very often, as in the case of expressives, meaning contributed to the CI dimension is speaker oriented and implicitly expresses the mental state of the speaker. In this sense it enters the common ground in a different way, as the speaker's choice of words indicates that she is the relevant source of truth regarding the propositions contributed (as is always the case for speaker-oriented material). As illustrated above, CI material regularly escapes the scope of logical operators such as negation and question forming operators.

Potts formalizes these intuitions in a type logic based on the addition of a second kind of propositions, *CI propositions*. The logic is set up in such a way that expressions contributing to the CI dimension denote a pair of values, the first one of the type usually associated with the syntactic category they belong to (e.g. a function from sets to sets in the case of a noun-modifying expressive), and the second one, which is always of the CI propositional type. The logic is also structured in such a way that the information can only flow from the at-issue dimension to the CI dimension, in the sense that at-issue meaning components contribute to the CI dimension, while CI material cannot contribute to the at-issue dimension during the compositional phase.

[1] John Lee Hooker was in fact born in Mississippi.

Arnold and Sadler (2010) expand on Potts's proposal and give an implementation of the analysis in LFG (Kaplan and Bresnan, 1982; Bresnan, 2001; Dalrymple, 2001) with Glue Semantics (Glue; Dalrymple et al., 1993; Dalrymple, 1999, 2001). Their starting point is the intuition that, from a resource-logical point of view, expressions contributing to the CI dimension create a pair of resources, one at the at-issue level and one at the side-issue one. To do so, Arnold and Sadler use a tensor pair as the resulting resource produced by a CI-contributing expression. The paired resources that are produced by the compositional process are then split and the at-issue material is used subsequently in the proof, while the side-issue material is only collected at the end of the compositional process.

In this paper we present an alternative treatment of Potts's analysis in LFG that starts from the proposal of Arnold and Sadler (2010) but strives for more generality and for a simpler treatment of multidimensional semantics. At the same time our approach allows for finer control of the flow of information between the two dimensions. In fact, although Arnold and Sadler (2010)'s implementation correctly restricts the flow of information only from the at-issue dimension to the CI-dimension during the compositional phase, at the same it lacks enough structure to model a class of interdimensional meaning interactions that Potts (2005, 2007) and AnderBois et al. (2010) discuss.

Our proposal is based on the suggestion made by Shan (2001) of using *monads* to uniformly model a large number of semantic phenomena. In general, a monad allows us to reproduce the structure of a certain compositional algebra in a more complex but related setting. We will show how the monadic approach allows us to retain a compositional interface based only on functional application and functional abstraction, even in the case of a multidimensional semantics. At the same time, the monadic machinery allows us to combine different semantic phenomena and to control how they interact. This will also allow us to shed light on the restrictions that can be observed in the kind of inter-dimensional meaning interactions that are not directly related to the compositional process.

The paper is structured as follows. In Section 2, we discuss the proposal of Arnold and Sadler (2010) and motivate our version on the basis of the aforementioned data that illustrate more complex patterns of interactions between dimensions. In Section 3, we introduce the concept of monads and discuss how we can use them to give structure to linguistic meanings. Section 4 provides a fully worked out example of how the monadic approach can be used in Glue Semantics to characterize multidimensional meanings and how monads can help us to model the interaction between the dimensions in cases involving non fully compositional phenomena. Section 5 concludes and presents directions for future research.

2 Arnold and Sadler (2010)

In this section we briefly review the proposal of Arnold and Sadler (2010), placing it in the context of Potts's analysis and identify the points of departure of our ap-

proach with respect to theirs. Subsequently we review a number of circumstances, initially discussed by Potts (2005, 52ff.), and more recently by AnderBois et al. (2010), in which at-issue content seems to require access to side-issue content, which would be precluded by Potts's type theory. We claim that this form of interaction also calls for an analysis based on monads.

Arnold and Sadler (2010) start their analysis from the assumption that expressions like appositives, non-restrictive relative clauses and expressives are fully integrated in the constituent and functional structures of the sentences to which they contribute, a view we fully agree with. All these constructions contribute to meaning as adjuncts and compose with the rest of the linguistic material following the standard projection architecture of LFG. The focus of their proposal is on the compositional rules that govern the interactions between dimensions.

The implementation they discuss is based on the analysis of Potts (2005) and, in particular, on a suggestion made by Potts (2005, 85ff.) for a resource-sensitive implementation of the theory. As discussed above, the idea of Potts is that each linguistic expression denotes a pair of objects: the at-issue contribution, of the type usually associated with the expression, and a possibly empty side-issue contribution, always of type t^c, a distinct CI propositional type. The meaning of linguistic expressions is composed according to the structure of a semantic tree derived from a syntactic tree using two different modes of composition. The first one involves only the at-issue dimension and corresponds to functional application:

(4)
$$\begin{array}{ccc} \langle \alpha(\beta), \_ \rangle & & \langle \alpha(\beta), \_ \rangle \\ \langle \beta, \_ \rangle \quad \langle \alpha, \_ \rangle & & \langle \alpha, \_ \rangle \quad \langle \beta, \_ \rangle \end{array}$$

In these rules, the at-issue meaning, the first component of the pairs, is composed via functional application. The CI dimension is left untouched and Arnold and Sadler particularly stress that it is not percolated up the tree. There is then another form of composition, specific for CI meaning:

(5)
$$\begin{array}{ccc} \langle \beta, \alpha(\beta) \rangle & & \langle \beta, \alpha(\beta) \rangle \\ \langle \beta, \_ \rangle \quad \langle \alpha, \_ \rangle & & \langle \alpha, \_ \rangle \quad \langle \beta, \_ \rangle \end{array}$$

Here the at-issue material is combined and transferred to the CI dimension (the second component of the pairs). At the same time the β component is duplicated and copied in the at-issue part of the meaning. These types of rules regulate the flow of information between the dimensions. In particular the type theory is set up so that values can only travel from the at-issue dimension to the side-issue one. The interpretation process is then completed by an additional step that collects all the CI propositions and conjoins them with the propositional content of the at-issue dimension. These rules break the resource sensitivity assumption of Glue Semantics (Asudeh, 2004, 2012), which is grounded in the use of linear logic (Girard, 1987), a resource logic, for semantic composition. In both cases the argument value β is used two times: it is first copied to the at-issue dimension but it is then reused as the argument of the function α in the side-issue dimension.

Arnold and Sadler follow and extend the suggestion of Potts of considering CI-contributing lexical items as objects producing a *pair* of resources, one for the at-issue dimension and one for the side-issue dimension. So, in general, the glue term describing the compositional behavior of these expressions will have a return type composed by two resources, one of the CI propositional type, combined with the tensor $\otimes$. This is reflected in the meaning terms for these expressions, as they will result, after all their arguments have been saturated, in a pair of objects.

However only the at-issue resource is going to be further used in the Glue proof, as the CI material must be inaccessible once it has been created. Therefore Arnold and Sadler introduce a rule to split the tensored pair into two resources. They call this rule *at-issue-ci-split*; we repeat it here in (6)

(6) $$\frac{\langle m, m' \rangle : r \otimes r_{t^c}}{m : r \quad m' : r_{t^c}}$$

Any CI resource obtained by the split rule is not used in the inferential process, as there are no linguistic expressions that consume a resource of that type. Nevertheless, to obtain the final interpretation, the CI propositions must be collected. To this end, Arnold and Sadler use the 'of course' operator, '!', which allows the term that it takes scope over to be used any number of times, thus relaxing resource sensitivity. This operator is then used for the meaning constructor of a silent linguistic operator applied to the root of the derivation. The silent operator simply scans the derivation, collects the CI propositions and conjoins them with the at-issue proposition. The silent operator would need to be different in case of other types of utterances, for instance in the case of questions, as in these cases the CI contribution cannot just be conjoined with the at-issue content of the speech act.

Arnold and Sadler also propose an alternative implementation that departs from Potts's proposal by adding a new projection to the LFG architecture. The idea is to avoid having to introduce a special propositional type for CI material distinct from the other standard type t, and, at the same time, to avoid having to introduce the of course operator. This new structure — *CI-structure* — is projected from the functional structure and parallel to the semantic structure. In this way, Arnold and Sadler can keep the resources separated, as they are instantiated from different structures, without the need to introduce an *ad hoc* type for CI propositions.

Our approach has points of contact with the implementation of Arnold and Sadler but there are also some fundamental differences. We will present here our account in an informal way but stressing the differences with Arnold and Sadler's system. In the next section we will give the formal details that justify our claims.

We start from the same assumption that linguistic constructions contributing to the CI dimension are fully integrated in the syntactic and functional structure of the sentence they appear in. We also share Potts's intuition that the denotation of linguistic expressions corresponds to a pair (or more generally, a tuple) of values. However, in our case the CI dimension is represented by a collection of propositions (the CI contributions made so far) rather than a single propositional value. This also means that in our analysis the CI content is percolated through

the semantic tree/proof. However the process of combining and percolating the CI component is built into the compositional process and is not accessible to lexical resources. This means that the CI material remains accesible only locally and only to those lexical items that operate on it by adding new propositional content.

Given that the compositional process is in a sense "aware" of the paired nature of meanings, we can uniformly treat them as atomic resources. Therefore we do not replicate the asymmetry present in Arnold and Sadler's system in which all expressions denote a pair, but where, at the level of the resource logic, some of them are represented as atomic resources while others are represented as pairs of resources. This also means that we do not need a special rule to split the resources and then recombine them, nor do we need to postulate a special type for CI propositions. In our derivations, the CI component is implicitly kept separated from the at-issue material and threaded through the compositional process in the background. As a result the proofs we obtain easily satisfy a strict version of resource-sensitive composition, as we do not need an operator like of course, !, which, with its possibility of repeated application, breaks down resource sensitivity.

By exploiting the paired nature of denotations we can also avoid introducing a special propositional type for the CI dimension, while simultaneously avoiding the introduction of a new kind of semantic structure. The two dimensions are in fact identified completely by the position they occupy in the pair. In this way we distinguish not between two different types of proposition but rather between two different modalities with which propositions are introduced in the common ground, a very similar distinction to the one proposed by Arnold and Sadler with the projection to the CI-structure.

Finally the monadic approach can be seen as more parsimonious from a theoretical point of view. Shan (2001) in fact shows how the organizing principles of monads can be used to model a wide range of semantic phenomena. He lists among the possible applications of monads phenomena such as focus, question formation, and intensionality. Giorgolo and Unger (2009) show how monads can also model dynamic phenomena like anaphora. This is particularly relevant for parenthetical constructions as they seem to interact with these type of phenomena in non-trivial ways. In particular, dynamic effects have raised some interest as they seem to create contexts in which the flow of information between dimensions is less constrained than we would predict on the basis of Potts's theory. Potts (2005, 52ff.) himself and AnderBois et al. (2010) present a number of cases in which information flows from the CI dimension to the at-issue dimension. All these cases have in common the fact that they involve some form of *uncertainty* in the meaning they denote. We list here some examples of "unruly" contexts:

1. Presupposition

   (7) Mary, a good drummer, is a good singer <u>too</u>.

2. Anaphora

(8) Jake$_1$, who almost killed a woman$_2$ with his$_1$ car, visited her$_2$ in the hospital.

3. VP ellipsis

(9) Lucy, who doesn't help her sister, told Jane to.

4. Nominal ellipsis

(10) Melinda, who won three games of tennis, lost because Betty won six.

What we see is that, in order to resolve the uncertainty introduced in the at-issue dimension by these constructions, we need to take into consideration the CI dimension. For instance in the case of (7), the presupposition triggered by *too* is resolved by the information that Mary has some additional musical talent beside singing, a fact we are informed of by the appositive *a good drummer.* Similarly in the case of anaphora and ellipsis, the unstated referent/property in the matrix clause is introduced in the CI contribution. The possibility of modeling multidimensionality and dynamic phenomena using the same theoretical apparatus seems to us a good reason to propose an alternative approach to Arnold and Sadler's.

3 Monads

The concept of a monad arises in category theory (Barr and Wells, 1984). It has found many applications in theoretical computer science as a model for the denotational semantics of side-effecting computation (Moggi, 1989, 1990) and in functional programming as a way to structure and compose side-effects (Wadler, 1992a,b). Here we try to provide the main intuitions behind monads and how they can be used in the context of natural language semantics.

The first intuition behind monads is that they are a way to reproduce a certain algebraic structure, in our case the algebra of meaning composition, in a richer setting that carries *more information*. The idea is that if we have a collection of functions and values that represent our meanings we can map them to new objects that contain the original information plus some additional meaning material. The monadic mapping allows us to maintain in the new richer setting the same compositional configurations we started with. The additional information varies in different types of monadic mappings, but in all cases we are able to reconstruct the original compositional configurations.

The characteristic of a monadic mapping is that the original meanings are associated with some kind of default information. In this way we obtain an object of the correct enriched type without committing to any particular enriched information. For example, in the case of multidimensionality, the meanings of linguistic expressions that contribute only to the at-issue dimension are mapped from the traditional unidimensional collection of meanings to a collection of paired meanings,

and the objects they are mapped to consist of the original meaning and a vacuous CI contribution.

The principles behind this intuitive view of monads continue to apply when we consider monads as models of computations. According to this perspective, a monad is a computation that yields a value while at the same time producing some side effects, like modifying some global environment or communicating with the "real world". Also in this case we can assume that we start from pure — i.e. side-effect free — functions and values and map them to computations with possible side effects that yield the starting object as the result of running. A monadic mapping will map a pure value/function to a computation that has no side effect and that only returns the original value when run. The computational perspective also exposes another important property of monads that makes them an interesting model of natural language meaning. The notion of side effects is intimately connected to the idea of sequentiality. For instance, the order in which we access a file by reading from and writing to it is fundamental in determining whether the computation fails or not. Monads can be composed to create larger computations from more elementary ones and the monadic approach requires the specification of a fixed order of evaluation. This property is particularly relevant for the non-compositional phenomena that we discussed above, in which we need to keep track of the linear order of appearance of the various expressions in order to predict the licit patterns of anaphora, presuppositions and ellipsis.

We formalize these intuitions by defining a monad as a triple: $\langle M, \eta, \star \rangle$.[2] M can be understood as the mapping that tells us to which type of enriched collections of values/functions we are lifting our unidimensional meanings. M can also be interpreted as a name for this specific collection. We will use the notation $M\ \alpha$ to indicate the type of objects that result from the application of the mapping M to objects of type α. η (pronounced 'unit') is the operation that brings us from the original, information-poor collection of meanings to the information-rich collection. It does so by encapsulating each object in the source collection in a "container" that also stores default, vacuous information. $\star$ (pronounced 'bind') is a binary operation that performs both the role of creating new monads from simpler ones and imposing an evaluation order for their computation. $\star$ takes a monad and a function from the type of the result yielded by the monad to another monad of the same kind. The operation runs the first monad, passes the result to the function and creates a new monadic computation. In the background, the side-effects/enriched information from the first monad and the second one are run sequentially/accumulated. In this way, the resulting monad creates a new value using the value produced by the first one and combines the side-effects/enriched information of the two monads. In order to obtain the properties we ascribed to monadic mappings above, the two

[2] We use here the definition normally found in the computer science literature (Moggi, 1989; Wadler, 1992b). This particular definition allows us a more natural description of the meaning of the expressions contributing to the CI dimension. The categorical definition is normally given in terms of a different triple (Barr and Wells, 1984); this is in any case completely equivalent to the one used here.

operations must satisfy the following laws for all x, f, g and m:

$$\eta(x) \star f = f\ x \tag{11}$$

$$m \star \eta = m \tag{12}$$

$$(m \star f) \star g = m \star (\lambda x.f\ x \star g) \tag{13}$$

Laws (11) and (12) characterize η as the left and right identity with respect to the composition of monads. This is a way to require η to couple the lifted value with vacuous information/no side-effect. Law (13) states that $\star$ behaves as an associative operator. This is relevant for us because it guarantees us that the ordering of the side effects is independent of the order of composition.

The specific monad we are going to use to model multidimensional semantics is known in the functional programming tradition as the *Writer* monad. The *Writer* monad maps values and functions to a pair composed by the value/pair and an element of a *monoid*. A monoid is an algebra with a single binary associative operation and an element that is the left and right identity of the operation. In our case the underlying set of the monoid is a set of sets of proposition (i.e. the possible collections of CI contributions), the binary operation is set union and the identity element is the empty set.[3] In the *Writer* monad the identity element corresponds to the vacuous information and the binary operation describes the way in which information is accumulated.

In our case the mapping *Writer* sends an object of type α to an object of type $\langle \alpha, p \rightarrow t \rangle$, a pair of an object of type α and a set of propositions. Type p is a quite conservative extension to the standard type theory based on e and t. p in fact represents the set of names of propositions. In this sense t can be seen as a subtype of p, namely the domain containing only the names $\{\top, \bot\}$.

Having defined *Writer* in this way we have, for example, that the interpretation of an intransitive verb, an object of type $e \rightarrow t$, will be mapped to an object of type $\langle e \rightarrow t, p \rightarrow t \rangle$, or more compactly *Writer* $(e \rightarrow t)$. η pairs every object with the empty set:

$$\eta(x) = \langle x, \{\ \} \rangle \tag{14}$$

$\star$ is instead defined as follows:

$$\langle x, P \rangle \star f = \langle \pi_1(f\ x), P \cup \pi_2(f\ x) \rangle \tag{15}$$

where π_1 and π_2 are respectively the first and second projection of a pair. In words, $\star$ is a binary function that takes 1) an input pair of a value and a collection of propositions and 2) a function f that produces a computation using the first value of the input pair. $\star$ produces a new computation whose value is the value of the computation produced by f and a new collection of propositions that is the union

[3] By using set union we make our monoid commutative. In the applications described in this paper this is not of particular relevance, but in certain cases it may be necessary to use a non-commutative operation to keep track of the order in which the propositions are combined.

of the input collection of propositions with the collection of propositions produced by f. The step involving the union of the collections of propositions is the one that allows us to use the *Writer* monad as a kind of logging system.

Notice that we have not added to the term language anything besides pairs and projections. The monoid structure in the second component of our monads is already expressible in the simply typed lambda calculus that we use for our meaning constructors. The identity element corresponds in fact to the function $\lambda t.\top$ and union can be expressed in terms of disjunction: $\lambda s.\lambda r.\lambda t.s\ t \vee r\ t$.

We still need to see how we can integrate the monadic approach with the LFG framework. Our solution, again inspired by Shan (2001), is to give a new Curry-Howard isomorphism interpretation of the elimination and introduction rules for the glue implication $\multimap$. We will however also need to introduce a new kind of implication in order to give an interpretation to the expressions contributing to the CI dimension.

Our goal is to be able to reproduce the unidimensional compositional configuration at the monadic level. This means that, starting from at-issue only lexical items and lifting them to the monadic level via η, we want to able to saturate a predicate of type *Writer* $(\alpha \rightarrow \beta)$ with an argument of type *Writer* α. Notice how we cannot simply use functional application because we are dealing here with two pairs. The solution proposed by Shan (2001) is to use the $\star$ operator to define a general notion of functional application for monadic meanings. The definition of this new form of functional application, which we call A following Shan (2001), is given in (16):

$$A(f)(x) =_{def} f \star \lambda g.x \star \lambda y.\eta\ (g\ y) : M\ (\alpha \rightarrow \beta) \rightarrow M\ \alpha \rightarrow M\ \beta \quad (16)$$

The monad encapsulating the function is run via $\star$ and its result (the function) is bound to the variable g. Similarly the argument monad is run and the result is bound to y. As a final step a new monad is created that returns the result of applying the function g to the argument y, without adding any additional information/side effect. In the background, the $\star$ operator takes care of threading the additional information (in our case the collection of CI propositions).

To obtain an isomorphism between the proofs in Glue Semantics and the monadic meaning terms we need to define a notion of *functional abstraction*. The definition of abstraction for monads is less mathematically pleasant and depends more heavily on its use in Glue proofs than the definition of monadic functional application. A is in fact just a function operating on values. The corresponding abstraction cannot be defined in the same way but makes use of the specific shape of the meaning language we use to decorate our proofs. The definition is given in (17).

$$\eta(x) \triangleleft m =_{def} m \star \lambda b.\eta\ (\lambda x.b) : M\ \alpha \rightarrow M\ \beta \rightarrow M\ (\alpha \rightarrow \beta) \quad (17)$$

The term $\eta(x) \triangleleft m$ indicates the monadic abstraction of a value x in the computation represented by the monad m. The interpretation of the term is close to that of

a classical abstraction $\lambda x.t$, in the sense that $\triangleleft$ signals a hole in m called x in the same way that λ signals a hole in t called x. The precise definition is however a bit more involved. In (17), $\eta(x)$ is assumed to be a hypothesis introduced in the proof. x must be a fresh variable. The hypothesis allows us to deduce the computation corresponding to m from which we then discard the hypothesis $\eta(x)$ via this form of abstraction. We extract the value yielded by m, bind it to b and return a new computation that returns the abstraction x over b.

In (18) we give the elimination and introduction rules for the glue logic implication, $\multimap$, using the newly defined monadic functional application and functional abstraction.

$$\frac{x : A \qquad f : A \multimap B}{A(f)(x) : B} \multimap E \qquad\qquad \frac{\begin{array}{c}[\eta(x) : A]_i \\ \vdots \\ m : B\end{array}}{\eta(x) \triangleleft m : A \multimap B} \multimap I_i \tag{18}$$

The mode of composition just outlined is not powerful enough to describe how certain expressions move information from the at-issue dimension to the CI one. In fact the class of objects composable with $\multimap$ is restricted to those that operate on the two components independently. To understand why this is so, consider the case of an expressive, like *fucking* in (1). This expressive takes an argument, a noun, and contributes to the at-issue dimension by returning its argument untouched, which means that it encapsulates the identity function as its at-issue meaning, and to the CI dimension by applying the predicate $\frown$ to its argument.[4] Given its at-issue contribution the type we would assign to its denotation is *Writer* $((e \to t) \to (e \to t))$. However an object of this type would not be able to apply the predicate $\frown$ to its argument. If we take a look at the definition of monadic application we can see that the functional value is actually applied to its argument outside of the original monad. The monad is in fact run and its return value collected, but only at the end is it combined with its argument. This means that the denotation for the expressive would not be able to access its argument to generate a CI contribution.

In order to properly generate the CI contribution, we need to assign to expressions like *fucking* a denotation that corresponds to a function that is "aware" of the monadic context in which it is evaluated. The idea is to have these types of expressions take monads as arguments, in our case pairs of at-issue and side-issue meaning material. The type we will assign to an expression like *fucking* is therefore *Writer* $(e \to t) \to$ *Writer* $(e \to t)$. This is the type of function that takes a monadic object encapsulating a predicate and returns another monadic object also encapsulating a predicate. In the case of the expressive, the function will return a monad containing the same predicate but paired with the CI proposition expressing a negative judgment about it.

To keep track of which type of composition it is necessary to introduce in the glue logic a new implication, $\multimap_*$. This new implication behaves exactly like the

[4]The frown symbol, $\frown$, is meant to evoke the idea of a negative judgement.

original one and comes equipped with its own notion of functional application and functional abstraction. By a slight twist of logic, application and abstraction for this new connective corresponds to standard application and abstraction, as we use them in traditional Glue Semantics, as shown by the term to the left of the colon in the following:

$$A_*(f)(x) =_{def} f\ x : (M\ \alpha \rightarrow M\ \beta) \rightarrow M\ \alpha \rightarrow M\ \beta \tag{19}$$

$$x \triangleleft_* m =_{def} \lambda x.m\ x : M\ \alpha \rightarrow M\ \beta \rightarrow (M\ \alpha \rightarrow M\ \beta) \tag{20}$$

In (21) we give the Curry-Howard isomorphism for respectively elimination and introduction of $\multimap_*$ and this additional type of monadic functional application and abstraction.

$$\frac{x : A \qquad f : A \multimap_* B}{A_*(f)(x) : B} \multimap_* E \qquad \frac{\begin{array}{c}[x : A]_i \\ \vdots \\ m : B\end{array}}{x \triangleleft_* m : A \multimap_* B} \multimap_* I_i \tag{21}$$

In the next section we show how the formal machinery introduced here can be used to analyse expressions involving side issue contributions.

4 Monads in action

In this section we present the details of our proposal. We start by working out in some detail the analysis of the contribution to the CI dimension of an expressive. We then move to another example illustrating the interaction between dimensions in the case of presupposition. This example allows us to see how the monadic approach controls the information flow in the desired manner.

Consider the sentence in (22).

(22) John loves goddamn Marilyn Manson.

We assume a standard constituent structure and associated functional structure. In particular we take it that *goddamn* works as a regular modifier that contributes to the ADJUNCT feature of *Marilyn Manson*. In Table 1 we present the meaning constructors that form our lexicon.

Lexical entries that contribute only to the at-issue dimension are assigned a meaning term very similar to the standard one. The one difference is in the 'lifting' of their meaning term to the monadic level by means of the η mapping. The expressive *goddamn* is instead given an interpretation that makes full use of the monadic setting. First of all the glue term associated with it denotes the fact that the expressive composes with the surrounding lexical material in a way that produces a contribution to the CI dimension. The expressive takes the NP *Marilyn Manson* as

WORD	MEANING TERM + GLUE TERM
John	$\eta(j) : j$
loves	$\eta(love) : m \multimap j \multimap l$
Marilyn Manson	$\eta(m) : m$
goddamn	$\lambda x.x \star \lambda y.\texttt{write}(\frown(y)) \star \lambda\_.\eta(y) : m \multimap_* m$

Table 1: Lexicon for *John loves goddamn Marilyn Manson.*

$$\dfrac{\dfrac{[\![\textit{John}]\!] : j \qquad \dfrac{[\![\textit{loves}]\!] : m \multimap j \multimap l \qquad \dfrac{[\![\textit{goddamn}]\!] : m \multimap_* m \qquad [\![\textit{Manson}]\!] : m}{A_*([\![\textit{goddamn}]\!])([\![\textit{Manson}]\!]) : m}}{A([\![\textit{loves}]\!])(A_*([\![\textit{goddamn}]\!])([\![\textit{Manson}]\!])) : j \multimap l}}{A(A([\![\textit{loves}]\!])(A_*([\![\textit{goddamn}]\!])([\![\textit{Manson}]\!])))([\![\textit{John}]\!]) : l}$$

Figure 1: Glue Semantics proof for *John loves goddamn Marilyn Manson.*

its argument via the special implication $\multimap_*$. In this way it can control the evaluation of the meaning term corresponding to the NP and extract from it the necessary information. The meaning term associated with *goddamn* illustrates how this is done: the expressive takes the NP as its first x argument, extracts from it its value (the referent of the NP) and, via the $\star$ operator, passes in the background the side-issue material that may have been computed by its argument (in this case none). The referent is bound to the variable y and using an auxiliary operation `write` the application of the predicate $\frown$ to y is logged to the CI dimension. `write` is a simple function, taking a proposition as its argument and returning a pair of a vacuous value and a collection of propositions containing only the argument. We can therefore assign to `write` the type $t \to Writer\ \perp$, where $\perp$ is a domain with a single inhabitant also named $\perp$. `write` is defined as follows:

$$\texttt{write} = \lambda p.\langle \perp, \{p\} \rangle \tag{23}$$

The final step performed by the denotation of *goddamn* is to return the interpretation of its argument without any additional change to the collection of CI propositions. Notice that the value returned by the `write` operation is not used anywhere in the lambda term and, following a common practice in programming languages, we indicate this by binding it with an underscore.[5]

The Glue proof is shown in Figure 1. The proof makes one use of the new rule for the elimination of $\multimap_*$, which is reflected in the proof term by the use of the special application A_*.

The resulting proof term encapsulates the instructions for computing the denotation of the utterance. The final result will be a pair whose first projection and second projection represent respectively the at-issue and side-issue dimensions. As discussed above, *goddamn* will take the denotation of *Marilyn Manson* to create a

[5]We could have of course used any variable different from y.

new computation whose result is the denotation of *Marilyn Manson* but that contributes a proposition to the CI dimension. The other applications are all instances of standard functional application lifted to the monadic level: the monads corresponding to the function and the argument are 'run', their values applied and, in the background, the CI contributions are collected. To see how this happens we will show the full expansion of the term. We will use the symbol $\rightsquigarrow$ to indicate a reduction and decorate it with subscripts indicating which steps are taken: we will use lex-def for the use of lexical postulates, η-def and $\star$-def for the definition of η and $\star$, A-def and A_*-def for the definition of A and A_*, `write`-def for the definition of `write`, and β for beta reduction (including the reduction of projections and unions).

The term we are reducing is repeated in (24).

$$A(A(\llbracket \textit{loves} \rrbracket)(A_*(\llbracket \textit{goddamn} \rrbracket)(\llbracket \textit{Manson} \rrbracket)))(\llbracket \textit{John} \rrbracket) \tag{24}$$

We start by reducing the subterm $A_*(\llbracket \textit{goddamn} \rrbracket)(\llbracket \textit{Manson} \rrbracket)$.

$$\begin{aligned}
&A_*(\llbracket \textit{goddamn} \rrbracket)(\llbracket \textit{Manson} \rrbracket) && \rightsquigarrow_{\text{lex-def}+A_*\text{-def}}\\
&(\lambda x.x \star \lambda y.\texttt{write}(\frown(y)) \star \lambda_.\eta(y))\ \langle m, \{\ \}\rangle && \rightsquigarrow_{\beta}\\
&\langle m, \{\ \}\rangle \star \lambda y.\texttt{write}(\frown(y)) \star \lambda_.\eta(y) && \rightsquigarrow_{\star\text{-def}}\\
&\langle \pi_1((\lambda y.\texttt{write}(\frown(y)) \star \lambda_.\eta(y))\ m),\\
&\{\ \} \cup \pi_2((\lambda y.\texttt{write}(\frown(y)) \star \lambda_.\eta(y))\ m)\rangle
\end{aligned} \tag{25}$$

The term $(\lambda y.\texttt{write}(\frown(y)) \star \lambda\_.\eta(y))\ m$ appears two times in (25); we show here its reduction and plug the result directly in (25) below.

$$\begin{aligned}
&(\lambda y.\texttt{write}(\frown(y)) \star \lambda_.\eta(y))\ m && \rightsquigarrow_{\beta}\\
&\texttt{write}(\frown(m)) \star \lambda_.\eta(m) && \rightsquigarrow_{\texttt{write}\text{-def}}\\
&\langle \bot, \{\frown(m)\}\rangle \star \lambda_.\eta(m) && \rightsquigarrow_{\star\text{-def}}\\
&\langle \pi_1((\lambda_.\eta(m))\ \ \bot), \{\frown(m)\} \cup \pi_2((\lambda_.\eta(m))\ \ \bot)\rangle && \rightsquigarrow_{\beta+\eta\text{-def}}\\
&\langle \pi_1(\langle m, \{\ \}\rangle), \{\frown(m)\} \cup \pi_2(\langle m, \{\ \}\rangle)\rangle && \rightsquigarrow_{\beta}\\
&\langle m, \{\frown(m)\}\rangle
\end{aligned} \tag{26}$$

Substituting (26) for $(\lambda y.\texttt{write}(\frown(y)) \star \lambda\_.\eta(y))\ m$ in (25) we obtain

$$\langle \pi_1(\langle m, \{\frown(m)\}\rangle), \{\ \} \cup \pi_2(\langle m, \{\frown(m)\}\rangle)\rangle \tag{27}$$

which, after computing the projections and the union, reduces to

$$\langle m, \{\frown(m)\}\rangle \tag{28}$$

In words, the denotation of the NP *goddamn Marilyn Manson* is a pair whose first projection is the individual Marilyn Manson and whose second projection is the proposition stating a negative judgement about that individual.

We continue the reduction by plugging (28) in (24) and expanding the inner application of *loves* to *goddamn Marilyn Manson*.

$$A(A(\llbracket \textit{loves} \rrbracket)(\langle m, \{\frown(m)\}\rangle))(\llbracket \textit{John} \rrbracket) \qquad \rightsquigarrow_{\text{lex-def}+A\text{-def}}$$

$$A(\langle love, \{\,\}\rangle \star \lambda f.\langle m, \{\frown(m)\}\rangle \star \lambda x.\eta(f\ x))(\llbracket \textit{John} \rrbracket) \tag{29}$$

As was the case before, the expansion of the $\star$ operator requires us to compute the same term $(\lambda f.\langle m, \{\frown(m)\}\rangle \star \lambda x.\eta(f\ x))\ love)$ twice. We reduce here independently and plug it in (29) below.

$$(\lambda f.\langle m, \{\frown(m)\}\rangle \star \lambda x.\eta(f\ x))\ love \qquad \rightsquigarrow_{\beta}$$

$$\langle m, \{\frown(m)\}\rangle \star \lambda x.\eta(love\ x) \qquad \rightsquigarrow_{\star\text{-def}}$$

$$\langle \pi_1((\lambda x.\eta(love\ x))\ m), \{\frown(m)\} \cup \pi_2((\lambda x.\eta(love\ x))\ m)\rangle \qquad \rightsquigarrow_{\beta+\eta\text{-def}}$$

$$\langle \pi_1(\langle love\ m, \{\,\}\rangle), \{\frown(m)\} \cup \pi_2(\langle love\ m, \{\,\}\rangle)\rangle \qquad \rightsquigarrow_{\beta}$$

$$\langle love\ m, \{\frown(m)\}\rangle \tag{30}$$

The first and the second projection of (30) are needed in the expansion of (29) as show in the following reduction steps:

$$A(\langle love, \{\,\}\rangle \star \lambda f.\langle m, \{\frown(m)\}\rangle \star \lambda x.\eta(f\ x))(\llbracket \textit{John} \rrbracket) \qquad \rightsquigarrow_{\star\text{-def}}$$

$$A(\langle \pi_1(\langle love\ m, \{\frown(m)\}\rangle),\ \{\,\} \cup \pi_2(\langle love\ m, \{\frown(m)\}\rangle))(\llbracket \textit{John} \rrbracket) \qquad \rightsquigarrow_{\beta}$$

$$A(\langle love\ m, \{\frown(m)\}\rangle)(\llbracket \textit{John} \rrbracket) \qquad \rightsquigarrow_{\text{lex-def}+A\text{-def}}$$

$$\langle love\ m, \{\frown(m)\}\rangle \star \lambda f.\langle j, \{\,\}\rangle \star \lambda x.\eta(f\ x) \tag{31}$$

Also in this case we can avoid clutter in the derivation by reducing only once the

term $(\lambda f.\langle j, \{\,\}\rangle \star \lambda x.\eta(f\ x))\,(love\, m)$ needed for the expansion of the $\star$ operator:

$$(\lambda f.\langle j, \{\,\}\rangle \star \lambda x.\eta(f\ x))\ (love\ m) \qquad \rightsquigarrow_{\beta}$$

$$\langle j, \{\,\}\rangle \star \lambda x.\eta(love\ m\ x) \qquad \rightsquigarrow_{\star\text{-def}}$$

$$\langle \pi_1(\lambda x.\eta(love\ m\ x)\ j), \{\,\} \cup \pi_2(\lambda x.\eta(love\ m\ x))\rangle \qquad \rightsquigarrow_{\beta+\eta\text{-def}}$$

$$\langle \pi_1(\langle love\ m\ j, \{\,\}\rangle), \{\,\} \cup \pi_2(\langle love\ m\ j, \{\,\}\rangle)\rangle \qquad \rightsquigarrow_{\beta}$$

$$\langle love\ m\ j, \{\,\}\rangle \tag{32}$$

We proceed by plugging (32) in (31):

$$\langle love\ m, \{\frown(m)\}\rangle \star \lambda f.\langle j, \{\,\}\rangle \star \lambda x.\eta(f\ x) \qquad \rightsquigarrow_{\star\text{-def}}$$

$$\langle \pi_1(\langle love\ m\ j, \{\,\}\rangle), \{\frown(m)\} \cup \pi_2(\langle love\ m\ j, \{\,\}\rangle)\rangle \qquad \rightsquigarrow_{\beta}$$

$$\langle love\ m\ j, \{\frown(m)\}\rangle \tag{33}$$

The first projection is the proposition that John loves Marilyn Manson and the second projection is the proposition stating a negative judgement about Marilyn Manson.

We mentioned in Section 2 that monads can also be used to model other types of semantic phenomena. We discuss here an example involving a non-restrictive relative clause and a presupposition trigger. Consider the following sentence:

(34) John, who likes cats, likes dogs also.

The sentence contributes two propositions to the common ground: 1) the fact that John likes cats and 2) the fact that John likes dogs. However the presupposition trigger *also* additionally imposes a test on the structure of the common ground. The speaker expresses with this item that in the common ground we must already have some information corresponding to the fact that John likes something besides cats. The information is indeed already present, as the non-restrictive relative clause informs us that John likes dogs and it does so *before* the position in which we are required to apply the test to the common ground. Our analysis will capitalize on the fact that monads can be layered to create new types of monads that combine their ability to enrich meaning. We will therefore create a monad that jointly deals with multidimensionality and presupposition by composing the monad we have described for multidimensional meaning with a monad to keep track of presuppositions. Fortunately, we can also use the *Writer* monad for the treatment of presupposition.

To combine two monads we actually need to consider an additional construct: *monad morphisms*. For our purposes it will be sufficient to understand monad morphisms as monad "factories". The idea is that a monad morphism can be instantiated as a monad by specifying the monads we want it to combine with. The monad

WORD	MEANING TERM + GLUE TERM
comma	$\lambda j \lambda l.j \star \lambda x.l \star \lambda f.\texttt{write}(f\ x) \star \lambda\_ .\eta(x) : j \multimap_* (j \multimap l) \multimap_* j$
also	$\lambda v.\lambda o.\lambda s.s \star \lambda x.v \star \lambda f.o \star \lambda y.\texttt{lift}(\texttt{check}(\exists z.f\ z\ x \wedge z \neq y)) \star$ $\lambda\_ .\eta(f\ y\ x) : (d \multimap j \multimap l) \multimap_* d \multimap_* j \multimap_* l$
John	$\eta(j) : j$
who	$\eta(\lambda P.P) : (j \multimap l) \multimap (j \multimap l)$
likes	$\eta(like) : c \multimap j \multimap l$
cats	$\eta(\iota x.cat^*(x)) : c$
likes	$\eta(like) : d \multimap j \multimap l$
dogs	$\eta(\iota x.dog^*(x)) : d$

Table 2: Lexicon for *John, who likes cats, likes dogs also.*

morphisms will provide some additional enrichment to the information stored in the monad we wrap it around. In our case we simply take the monad morphisms to add an additional component to our meanings. This means that the meanings we will end up with will be formed by a pair whose first component is another pair. To be able to use the functions defined for the internal monad we also need a way to lift them to the level of the more complex monad. This can be done in our case using the following function `lift`:

$$\texttt{lift}(m) = m \star \lambda x.\eta(\langle x, \{\ \}\rangle) : \textit{Writer}\,\alpha \to \textit{Writer}\,(\textit{Writer}\,\alpha) \tag{35}$$

where $\star$ and η are the operator for the monad around which we wrap the morphism.[6]

In Table 2 we list the relevant lexical entries. The entries for the at-issue-only items are constructed in the same way discussed above, by applying the η operator. The only difference here is that η is the operator bringing us to the monad composed by the multidimensional monad (which starts as a monad morphism) together with the presuppositional one. *comma* is a silent operator that we borrow from Potts's and Arnold and Sadler's analysis of non-restrictive relative clauses. Looking at the associated glue logic term we see that *comma* takes 1) the resource corresponding to the NP to which the relative clauses is attached and 2) the relative clause, and returns the NP resource. These resources are consumed via the second monadic functional application, indicated in the term with the special implication $\multimap_*$. Behind the scenes, *comma* saturates the denotation of the relative clause, a one place predicate, with the denotation of the NP, stores the resulting proposition in the CI storage using the now familiar `write` function and returns the denotation of the NP as its final value.

The denotation of *also* introduces the new presuppositional monadic level. As was the case with *comma*, its glue term is built using the special implication $\multimap_*$,

[6]Notice that although we say that we wrap the monad morphism around the monad the result in our case will be "inside-out": the additional information expressed by the monad morphism will end up in the internal pair, while the information coming from the simple monad will be collected in the second component of the external pair.

a signal of the fact that this lexical item performs some additional work besides returning a value. We analyse *also* as a sentential operator that takes as arguments the verb, the subject and the object of the sentence. The result type is the one corresponding to the propositional value of the at-issue component of the sentence. The meaning term describes the semantic operations corresponding to the evaluation of *also*. The meaning of the verb, the object and the subject are extracted and bound respectively to the variables f, y and x. The next step is to perform a side-effect in the presupposition monad. As stated earlier, the presupposition monad is really just another instance of the *Writer* monad, again constructed as a pair of a value and a set of propositions. The presupposition monad is defined in exactly the same way as the multidimensional monad is. The function we see here, `check`, is really just another name for the function `write`, used here to clarify the levels at which the operations take place. `check` adds the presuppositional condition that in the model in which the sentence is evaluated there has to be an entity z such that the subject (x) likes z but z is different from the object (y). This operation is lifted to the level of the multidimensional monad via `lift` and the computation terminates by returning the application of the verb to its arguments.

The resulting proof term is show in (36):

$$A_*(A_*(A_*([\![\textit{also}]\!])([\![\textit{likes}]\!]))([\![\textit{dogs}]\!]))(A_*(A_*([\![\textit{comma}]\!])([\![\textit{john}]\!])) \\ (A([\![\textit{who}]\!])(A([\![\textit{likes}]\!])([\![\textit{cats}]\!])))) : l \tag{36}$$

After reducing the term we obtain the following pair:[7]

$$\langle\langle like(j, \iota x.dog^*(x)), \{like(j, \iota x.cat^*(x))\}\rangle, \\ \{\exists z.like(j, z) \wedge z \neq \iota x.dog^*(x)\}\rangle : l \tag{37}$$

The first component is a pair containing the at-issue proposition that John likes dogs and the side-issue proposition that John likes cats. The second component of the outer pair lists the conditions that must be met to satisfy the presuppositions triggered in the evaluation of the sentence. In this case, the only condition is that there must be something else besides dogs that John likes. The proposition that satisfies this condition can be found in the CI dimension.

5 Conclusion

In this paper we presented an analysis of multidimensional semantics based on a monadic analysis of meaning. Our approach exploits the abstraction capabilities of monadic mappings in order to maintain a largely standard, unidimensional glue logic for composition while assigning more complex meanings to the linguistic resources. The only innovation in the glue logic is the introduction of a new implication, $\multimap_*$.

[7] We use $\iota x.dog^*(x)$ and $\iota x.cat^*(x)$ to denote respectively the contextually relevant plural dog individual and the contextually relevant plural cat individual.

We started by discussing the proposal of Arnold and Sadler (2010) to model multidimensional meaning in Glue Semantics by the use of a tensor conjunction. While their approach is capable of accounting for the basic data, it does so at the price of breaking the resource sensitive contract of linear logic. Our approach does not contravene this fundamental assumption of Glue Semantics. At the same time, our approach seems more flexible and general, as it can be adapted to different scenarios, in particular if we decide to further differentiate non-at-issue contributions. Monads allows us to retain the simple, familiar compositional configurations in the unidimensional case, while at the same time composing more complex objects on the meaning side of the derivation. Another promising characteristic of the monadic approach is the possibility of using the same abstractions to deal with different semantic phenomena.

The analysis of example (34), presented in Section 4, points to the fact that the interaction between dimensions may be more complex than previously theorized. Here we have just started sketching a possible analysis in terms of layering of monadic mappings. We leave for future work the study of the different varieties of contexts that make the picture about meaning interactions more complex and how these interactions can be reconstructed in terms of unifying principles.

References

AnderBois, Scott, Brasoveanu, Adrian and Henderson, Robert. 2010. Crossing the Appositive/At-issue Meaning Boundary. In Man Li and David Lutz (eds.), *Proceedings of SALT 20*, pages 328–346.

Arnold, Doug and Sadler, Louisa. 2010. Pottsian LFG. In Miriam Butt and Tracy Holloway King (eds.), *Proceedings of LFG10*, pages 43–63, Stanford, CA: CSLI Publications.

Asudeh, Ash. 2004. *Resumption as Resource Management*. Ph. D.thesis, Stanford University.

Asudeh, Ash. 2012. *The Logic of Pronominal Resumption*. Oxford: Oxford University Press, to appear.

Barr, Michael and Wells, Charles. 1984. *Toposes, Triples, and Theories*. Springer-Verlag.

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

Dalrymple, Mary (ed.). 1999. *Semantics and Syntax in Lexical Functional Grammar: The Resource Logic Approach*. Cambridge, MA: MIT Press.

Dalrymple, Mary. 2001. *Lexical Functional Grammar*. San Diego, CA: Academic Press.

Dalrymple, Mary, Kaplan, Ronald M., Maxwell III, John T. and Zaenen, Annie (eds.). 1995. *Formal Issues in Lexical-Functional Grammar*. Stanford, CA: CSLI Publications.

Dalrymple, Mary, Lamping, John and Saraswat, Vijay. 1993. LFG Semantics via Constraints. In *Proceedings of the Sixth Meeting of the European ACL*, pages 97–105, European Chapter of the Association for Computational Linguistics, University of Utrecht.

Giorgolo, Gianluca and Unger, Christina. 2009. Coreference without Discourse Referents: a non-representational DRT-like discourse semantics. In B. Plank, T. Kim Sang and T. Van de Cruys (eds.), *Computational Linguistics in the Netherlands 2009*, LOT Occasional Series, No. 14, pages 69–81, LOT.

Girard, Jean-Yves. 1987. Linear Logic. *Theoretical Computer Science* 50(1), 1–102.

Kaplan, Ronald M. and Bresnan, Joan. 1982. Lexical-Functional Grammar: A Formal System for Grammatical Representation. In Joan Bresnan (ed.), *The Mental Representation of Grammatical Relations*, pages 173–281, Cambridge, MA: MIT Press, reprinted in Dalrymple et al. (1995, 29–135).

Moggi, Eugenio. 1989. Computational Lambda-Calculus and Monads. In *Proceedings of the Fourth Annual Symposium on Logic in computer science*, pages 14–23, IEEE Press.

Moggi, Eugenio. 1990. An Abstract View of Programming Languages. Technical Report, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, Edinburgh.

Potts, Christopher. 2005. *The Logic of Conventional Implicatures*. Oxford: Oxford University Press.

Potts, Christopher. 2007. The Expressive Dimension. *Theoretical Linguistics* 33(2), 165–197.

Shan, Chung-chieh. 2001. Monads for Natural Language Semantics. In Kristina Striegnitz (ed.), *Proceedings of the ESSLLI-2001 Student Session*, pages 285–298, 13th European Summer School in Logic, Language and Information.

Wadler, Philip. 1992a. Comprehending Monads. In *Mathematical Structures in Computer Science*, pages 61–78.

Wadler, Philip. 1992b. The Essence of Functional Programming. In *POPL '92: Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 1–14, New York, NY, USA.