Closed Categories and Categorial Grammar

Daniel J. Dougherty
Wesleyan University
Middletown, CT 06459
ddougherty@eagle.wesleyan.edu

Abstract

Inspired by Lambek's work on categorial grammar, we examine the proposal that the theory of biclosed monoidal categories can serve as a foundation for a formal theory of natural language. The emphasis throughout is on the derivation of the axioms for these categories from linguistic intuitions. When Montague's principle that there is a homomorphism between syntax and semantics is refined to the principle that meaning is a functor between a syntax-category and a semantics-category, the fundamental properties of biclosed categories induce a rudimentary computationally oriented theory of language.

1 Introduction

This paper presents some preliminary steps in an approach to natural language via category theory. The proposal will be a conservative one in the sense that we work within the tradition of interpreting meanings of phrases as mappings, but the difference here is that we do not assume more than that — in particular, meanings need not be standard set-theoretic functions. In fact, the goal is to try to isolate the consequences of a weak claim about language (that meanings behave like functions), by working in the most general theory of functions — category theory. Issues of intensionality will not be addressed, but we do not feel that this is an essential restriction on the approach.

Some of the novelties of this setting are :

- Types are considered as syntactic entities, eligible for semantic interpretation. This allows a model-theoretic analysis of type shifting.
- The general character of the theory is computational and algebraic, rather than set theoretic.
- The goal is not an explication of what meanings *are*, but rather, how they *behave*.

Our paper attempts to show how one might discover categories for language by working from semantic intuitions, and further motivates the approach on methodological grounds (see the discussion of "model-theoretic semantics" below). In this way, we attempt to consolidate the seminal insights of Lambek and Montague [15], [16]. Montague's program for model-theoretic semantics is an active area of investigation ([7] is an introduction).

The work described here was done in 1986, inspired by the early ideas of Lambek on categorial grammar [9], [10] and the resurrection and development of these ideas by van Benthem ([5] for example), but independently of [13] (the record of a 1985 conference), in which Professor Lambek makes essentially the same proposal. The presentations there focus on a syntactic *calculus*, whereas we concentrate here on categories as a foundation for interpretation. Needless to say, the connection between the two is well-known to Professor Lambek, as the originator of the notion that cartesian closed categories and typed lambda-calculi are the same subject ([11], [12]). Nevertheless, we were persuaded that it is worthwhile to present this account, which discovers closed categories by working directly from the linguistic data, and emphasizes the model-theoretic consequences of the approach.

Lambek's calculus describes a natural class of syntactic type-computations. The ideas underlying this calculus have found application in the work of Ades and Steedman [1], Partee and Rooth [19], Bach [3] and others.

We need to confront some unfortunate clashes of terminology between category theory and categorial grammar. The term "category" will always have its mathematical meaning [17], while "syntactic category" will be used to denote NP, VP, or their categorial grammar correlates. Fortunately, category theorists say "categorical", while linguists say "categorial" — this will help.

The abstraction of type b over type a will be denoted by (b/a) (right-searching), $(a\backslash b)$ (left-searching) , or (a, b) (in an undirected system).

Montague's treatment of semantics is commonly described as "model theoretic", but this is true only in a weak sense. In that treatment, phrases take their meanings in objects in the hierarchy of function spaces constructed over a set of basic types. But note that once the sets interpreting the basic types are chosen, everything else (the nature of the function spaces, the available mappings between them, etc.) is determined by set theory. Thus, of those facts which are common to all Montague interpretations, there is no way to distinguish implicit language universals from accidents of set theory. Indeed, given that one of the features of an avowed model-theoretic semantics is that "true in all models" equals "true about language", embracing a strict set-theoretic framework amounts to a claim that accidents of set theory *are* language universals.

As an example, note that in every Montague interpretation, the cardinality of the set E of entities is strictly less than that of the set $P = \{p | p : E \to \{0, 1\}\}$ of intransitive verb phrase denotations, which is in turn strictly less than that of the set $\{g | g : P \to \{0, 1\}\}$ of generalized quantifier denotations. This is not motivated by any linguistic intuition, as far as we know. Of course, the type assignments leading to the above are subject to dispute, but the point here is that if all models are function hierarchies, a choice of type has as a side effect an assertion about cardinalities.

Another drawback to a restricted notion of "model" is that it prevents us from defining certain notions model-theoretically. Type-lifting is an example. In order to reflect the similar syntactic behavior of proper names and quantified noun phrases, Montague assigned the same type to these classes *in the lexicon*. This treatment is justified by the observation that elements of a set E naturally give rise to (principal) ultrafilters. Similarly, the Geach Rule: "raise phrases of type (a, b) to type $((c, a), (c, b))$" is justified by the canonical lifting of a function f in (A, B) to the function *compose–with–f* in $((C, A), (C, B))$.

But in every Montague interpretation, there will be (lots of very non-canonical!) functions from (A, B) to $((A, C), (B, C))$. Why then don't we agree to raise phrases of type (a, b) to, for example, $((a, c), (b, c))$? The pre-theoretic answer is, "because there is no natural way to *interpret* such a shift". Lambek's calculus provides a proof theoretic answer – it is not derivable in the system.

A properly model-theoretic answer will define the notion of "valid" type shift as one which obtains in all models, and reject the shift proposed above since there are models which provide counterexamples. For us, these models will be closed categories, and the type-shifts will be arrows in such categories. The Completeness Theorem hinted at in the last section tells us that that these answers agree — the Lambek calculus generates precisely the valid shifts.

The categories presented in the present paper will not have enough structure to reflect the subtler aspects of either syntax or semantics, but they do provide a natural, flexible interpretation of abstraction, application, type-lifting, and the like. Our claim is that categories for language should have at least the structure of those presented here, and that these latter form a framework for future work.

Very little background in category theory is assumed; all of the definitions and results about categories are standard, and proofs are omitted. In the context of categories we treat the words "map" and "arrow" as synonymous. I owe a debt to F.E.J. Linton for many discussions about category theory: errors or infelicities below are probably things we didn't talk about, and are certainly my responsibility.

2 Closed Categories

In this section we try to motivate the ingredients that should go into a categorical treatment of grammar and of meaning, and arrive at the notion of closed category, essentially due to Eilenberg and Kelly [8].

The fundamental assumption is Frege's compositionality principle. It is helpful to construe this as two assertions:

- Phrase meanings operate on phrase meanings.
- The meaning of a compound phrase is derived from the meanings of its parts and the syntactic structure of the phrase.

The first assertion, innocently interpreted, leads to the doctrine:

> "types denote sets and meanings are functions on these sets".

We want to avoid jumping to such a conclusion, for the reasons discussed in the introduction, and suggest the more general strategy:

> "types stand for objects in a category and meanings are arrows".

This is a constraint on semantics. When we embrace that aspect of Montague's methodology which proposes that meaning is a homomorphism from syntax to semantics [15] we commit ourselves to the idea that syntax is a category as well.

So far, the outline of the categorical approach to categorial syntax is as follows. Types are objects $(a, b, c, ..)$ in a category, phrases are arrows, and complex phrases are built by composition. The first step is to uncover the nature of these categories. The data for the syntax-categories we define will reflect the combinatorial principles of categorial grammar, and the semantics-categories, where meaning computations are done, will need to have enough structure to *interpret* the arrows of the syntax.

Ajdukiewicz's original system of categorial grammar [2] had undirected exponential types (a, b). Bar-Hillel [4] introduced directionality of the functions, allowing the syntax to forbid transposition of elements. The use of undirected functional types is sometimes presented as a simplifying assumption (e.g., in [5]) allowing a focus on mechanics of type shifting rather than syntax. For Ades and Steedman [1] it reflects a claim that richer modes of combination can free the grammar from consideration of order. We want to be as sensitive to the syntax as possible, and do not want to build a claim such as Ades and Steedman's into the *foundations*, and so our syntax-categories will be general enough to encompass each of these points of view.

Categorial grammar is a grammar of typed function application. Any such system has a two-layered syntax, since there are rules for constructing types as well as for building terms, the utterances of the language. (For example, the pure typed lambda calculus builds types exclusively with an exponentiation operator, while richer versions allow product types, sums, and even quantification.) We first describe the nature of the objects in our categories, then discuss the arrows.

The first part of the functionality principle requires that meanings play two roles; they must behave like functions and like inputs. Categorically, this means that for objects a and b, the set of all arrows from a to b must also appear, in some sense, as an object. This latter property is precisely what the definition of "closed category" (Definition 2.1 below) is intended to capture. This requirement is met by function spaces in the category SET upon which Montague-style semantics is founded — exponentiation is function-space construction. (Note that even though SET doesn't distinguish between the *interpretations* of $a\backslash b$ and b/a, categorial *syntax* does.)

Turning to the second of Frege's principles, a few words are in order about the notion "syntactic structure of the phrase". Pure categorial grammar uses only the exponentiation operators $\backslash$ and $/$ (or perhaps a single symmetric operator) on types, and concatenation in the term syntax. It is natural to introduce a pairing operator into the *type* syntax, to construct a type for a pair of lexical items, (such an operator has been profitably added to the lambda calculus, for example). After all, the syntax should be able to find a place for the pair $\alpha\beta$ as a phrase composed of items of types a and b. Thus the objects of our categories should be subject to a pairing operation. This signals a shift to *monoidal* categories. It would be misleading to call the operation a *product*, as will be seen later.

Ajdukiewicz's categorial grammar, which allows only simple application in its semantic component, could do without pairing in its syntax since two constituents of types, say a and $(a\backslash b)$, are always immediately combined, and always in the same way. For us, the most significant consequence of the introduction of pairing will be that application can be represented as an operation in the system, specifically as an arrow in the syntax category, out of a pair made from types such as a and $(a\backslash b)$.

When we look ahead to interpreting the types and phrases in a semantics-category, we can see that in fact the operators $*$, $\backslash$, and $/$ should actually be *functors*, that is, that they induce operations on arrows as well. Consider $*$ first. Suppose $f : a{\rightarrow}a'$ and $g : b{\rightarrow}b'$ are phrases. The map f is an indication that there is a computation from meanings of phrases of type a to meanings of phrases of type a', and similarly for g. The meaning of the map $(f * g)$ can be thought of simply as the joint computation. Functoriality requires that this pairing respects composition. This is nothing more than a weak version of compositionality (in this instance the confluence of mathematics and linguistics jargon is felicitous).

Exponentiations should naturally extend to functors as well, that are, in a/b and $b\backslash a$, *contravariant* in the argument b and *covariant* in a. This means that for $f : b{\rightarrow}a$ and $g : c{\rightarrow}d$ there is an arrow $(f\backslash g)$ from $(a\backslash c)$ to $(b\backslash d)$, and an arrow $(g/f)$from (c/a) to (d/b). The construction corresponds to the following intuition. Suppose phrases of type b can shift to type a and that type c shifts to d. Then a phrase γ of type $(a\backslash c)$ can be thought of as also belonging to type $(b\backslash d)$: when such a phrase is presented with a type b item, that item can be promoted to type a, then γ is applied, then the type c result shifted to type d.

It is convenient to recognize the empty string as the identity element for pairing. This should live in some type: let $\mathbf{1}$ denote the categorical object interpreting that type. Intuitively, we think of the empty string as being the only inhabitant of that type, but all that is required is that the object $\mathbf{1}$ serve as the identity for object pairing.

The final piece of data to go into the definition of closed category reflects the insight that led to Lambek's revival and generalization of Ajdukiewicz's original categorial grammar. We motivate it here by an example in the category of sets. Suppose h is a function of two variables, from $X \times Y$ into A. Then for a fixed y in Y one can define a function h_y from X to A, defined by $h_y(x) = h(x, y)$. We have just described a process (a function) by which elements of Y yield functions from X to A. The process itself depended on h, and it is traditionally called $curry(h)$ after the logician Haskell Curry.

Definition 2.1 [Eilenberg and Kelly (1965)]

A *biclosed monoidal category* (or simply *closed category*) $\mathcal{C}$ is a category with the following data:

Pairing: a functor from $\mathcal{C} \times \mathcal{C}$ to $\mathcal{C}$,
 mapping X, Y to $(X * Y)$,

Exponentiation: two functors from $\mathcal{C}^{op} \times \mathcal{C}$ to $\mathcal{C}$,
 mapping X, Y to $(X \Rightarrow Y)$, and X, Y to $(Y \Leftarrow X)$, respectively,

Unit: an object $\mathbf{1}$, and for each X, two isomorphisms :
 $lu_X : (\mathbf{1} * X) \rightarrow X$, and $ru_X : (X * \mathbf{1}) \rightarrow X$.

Adjointness: a collection of natural isomorphisms :
 $curry_{X,Y,A} : Hom(X * Y, A) \cong Hom(Y, (X \Rightarrow A))$, and
 $curry'_{X,Y,A} : Hom(X * Y, A) \cong Hom(X, (A \Leftarrow Y))$.

As a first application of the axioms, consider the adjointness

$$Hom(X * Y, A) \cong Hom(Y, X \Rightarrow A)$$

and set Y equal to $(X \Rightarrow A)$. The conclusion is that

$$Hom(X * (X \Rightarrow A), A) \cong Hom((X \Rightarrow A), (X \Rightarrow A)).$$

But the second Hom-set always has the identity map in it, so we conclude that there is always a map in $Hom(X * (X \Rightarrow A), A)$, and a special one at that, since it corresponds to an identity under a natural isomorphism. We shall call this map $app_{X,A}$. Clearly we could play the same game using the other adjointness, and we would derive a map $app'_{X,A}$ from $(A \Leftarrow X) * X$ to A.

We can begin to support the claim that the objects $(X \Rightarrow A)$ "behave like" function spaces. In the category of sets, each *app* is indeed application, and, when h is a function from $X \times Y$ to A and $curry(h)$ maps Y to the function-space $X \Rightarrow A$, the application operator *undoes* the currying. This situation is faithfully reflected in any closed category:

Proposition 2.2 (Universality of *app* **and** *app'*) *For each two objects X and A in a closed category, there are maps*

- $app_{X,A} : X * (X{\Rightarrow}A){\rightarrow}A$, *and*
- $app'_{X,A} : (A{\Leftarrow}X) * X{\rightarrow}A$,

which are unique with respect to the following properties (respectively):

- *for every* $h : (X * Y){\rightarrow}A$, $app \circ (id_X * curry(h)) = h$, *and*
- *for every* $h : (X * Y){\rightarrow}A$, $app' \circ (curry'(h) * id_X) = h$.

As it happens, the property above is equivalent to adjointness, and so could have been taken as the key defining property of closed categories.

Another interesting map is determined if one returns to the *Hom*-set equation and substitutes $(X * Y)$ for A. We leave this to the reader.

Definition 2.3 A closed category is

associative if it has a collection of natural isomorphisms:
$assoc_{A,B,C} : (A * (B * C)){\rightarrow}((A * B) * C)$, and

symmetric if it has a collection of natural isomorphisms:
$sym_{A,B} : (A * B){\rightarrow}(B * A)$.

All of the data above is subject to a COHERENCE restriction, which means that all reasonable diagrams involving the maps defined above commute [17].

Lemma 2.4 *In a* symmetric *closed category, there are natural isomorphisms*

$$absym_{X,Y} : (X{\Rightarrow}Y){\rightarrow}(Y{\Leftarrow}X).$$

It follows that in a symmetric closed category any maps to or from some $(X{\Rightarrow}Y)$ induce maps to or from $(Y{\Leftarrow}X)$, by composing with *absym* or its inverse. So an undirected system can be thought of as a choice of one of the operators, say $\Rightarrow$, and an automatic translation of any computation involving $\Leftarrow$ into one involving $\Rightarrow$. An undirected type *syntax* is the reflection of such a decision; the symmetry is advertised by notation such as (x, y).

The issue of associativity is related to the methodological question of whether the input to the semantic component is considered to be an unstructured string or a syntax tree. It turns out that associativity is needed in order to have a composition arrow from objects $(A{\Rightarrow}B)$ and $(B{\Rightarrow}C))$ to $(A{\Rightarrow}C)$. This latter is the embodiment of a proposal made by several authors (for example, Ades and Steedman's Forward Partial Combination).

Examples. Each of the following is a symmetric associative closed category.

- *SET*, the category of sets and functions between them.

- CAT, the category of all small categories (those with only a set's worth of objects and maps), with functors between them.
- CL, the category of complete lattices and complete lattice homomorphisms. This suggests the intriguing possibility of using *Scott domains* for natural language semantics. Related examples include the category of Boolean algebras and the category of Heyting algebras. Keenan and Faltz [14] have made a proposal making extensive use of Boolean-algebraic models.
- Generalizing all of the above, any cartesian closed category satisfies Definition 2.1. In fact, the categories of Definition 2.1 will fail to be cartesian closed just when pairing is not a categorical product. This generality is crucial for categories which are to serve for syntax, as will be seen in the next section.
- The example which has been the motivation for this entire project is this: The types in a lexicon serve as objects, and proofs in the Lambek calculus serve as arrows. The rules of the Lambek calculus correspond precisely to the requirements for a closed category.

Hom sets as objects.

We have not yet made any use of $\mathbf{1}$. In the category SET, when pairing is interpreted as cartesian product, any one-element set can serve as $\mathbf{1}$; let us assume one has been fixed. Now note that arrows from $\mathbf{1}$ to a set S pick out a single element of S, and indeed it is a standard trick to *identify* elements in SET with arrows from $\mathbf{1}$. Remarkably, the axioms for closed categories allow us do that generally, as we now show.

The fact that $\mathbf{1}$ is the identity for $*$ generates some natural isomorphisms between Hom-sets:

Proposition 2.5 *For any X and A,*

$$Hom(X, A) \cong Hom(\mathbf{1}, X \Rightarrow A) \cong Hom(\mathbf{1}, A \Leftarrow X).$$

This is an immediate consequence of the basic adjointness condition from Definition 2.1 and the facts that $\mathbf{1} * X$ and $X * \mathbf{1}$ are each isomorphic with X.

Notation. Suppose $f : X \rightarrow A$, and let $g : \mathbf{1} \rightarrow (X \Rightarrow A)$ or $g : \mathbf{1} \rightarrow (A \Leftarrow X)$. Then we write

- $\lceil f : \mathbf{1} \rightarrow (X \Rightarrow A)$, and
- $f \rceil : \mathbf{1} \rightarrow (A \Leftarrow X)$
- $\lambda g : X \rightarrow A$

for the maps given by the bijections from Proposition 2.5.

From now on, the phrase "$\mathbf{1}$-element of A" will mean "arrow from $\mathbf{1}$ to A."

Now if $h : X \rightarrow A$, then $\lceil h : \mathbf{1} \rightarrow (X \Rightarrow A)$. The sense in which the objects $(X \Rightarrow A)$ represent the Hom-sets $Hom(X, A)$ can now be stated :

elements of $Hom(X, A)$ correspond to $\mathbf{1}$-elements of $(X \Rightarrow A)$

As it stands, this is pure convention. We need to go further and justify our earlier informal claim that the exponentiation objects behave like *Hom*-sets, now better stated as "the $\mathbf{1}$-elements of $(X \Rightarrow A)$ behave like arrows from X to A".

Suppose x is a $\mathbf{1}$-element of X and $h : X \rightarrow A$. Then $h \circ x$ is a $\mathbf{1}$-element of A. But in the category we can use our application arrow *app* on the pair h and x, or more precisely, the $\mathbf{1}$-element $(x * \ulcorner h)$ of the object $X * (X \Rightarrow A)$. This also yields a $\mathbf{1}$-element of A. The next lemma says that these will always coincide:

Proposition 2.6 *Let $x : \mathbf{1} \rightarrow X$ and $h : X \rightarrow A$. Then*

$$h \circ x = app_{(X,A)} \circ (x * \ulcorner h), \text{ and}$$

$$h \circ x = app'_{(X,A)} \circ (\urcorner h * x).$$

If x is a $\mathbf{1}$-element of X and h a map from X to A, then we will often think of $h \circ x$ as the result of "applying" h to x – this usage is justified by the previous lemma.

The essence of what has been abstracted from the category of sets is this very fruitful ambiguity: we interpret types as certain objects (such as (A⇒B)), but these can also be regarded as collections of arrows (such as *Hom*(A, B)). Thus an individual phrase meaning can be treated as an element (e.g., when it is input to another meaning), or as an arrow (e.g., when it is thought of as an operator).

In the next section we show how phrases of basic types such as e, t, etc., may now be interpreted as $\mathbf{1}$-elements of the appropriate category objects. In this way, we uniformly treat *all* meanings as maps.

3 Syntax-categories

The previous section defined and attempted to motivate the kind of category structure which seems appropriate to syntax. In this section we work out some small examples to get a sense of the phenomena our program might cover.

Definition 3.1 Fix B, a set of *base type symbols.*

- *Typ* is the set of type expressions generated from B and the constant symbol $\mathbf{1}$ by the binary operations / and \ .
- A *lexicon* is a set of entries of the form $[\alpha : a]$, where α is a *lexical item* and $a \in Typ$.
- A *syntax tree* is a binary tree whose leaves are lexical items.
- A *phrase* is a syntax tree with a linear ordering on its leaves.

Some notes: The lexicon is not assumed to be a function, that is, a given α can be assigned different types. The order on the leaves of a phrase represents the order of the words as an utterance; it need have no relation to the tree order. The interior nodes of a phrase are not labeled, and in particular, there are no non-terminal symbols being used. The fact that our syntax trees are binary is a reflection of the fact that the categorical pairing is a binary operation, and ultimately from the fact that function application, the true primitive operation, is binary.

Example. Let $B = \{e, t\}$.

Take the lexicon L to contain the following (writing p for $(e\backslash t)$):

- [John : e], [Ann : e],
- [walk : p],
- [loves : (p/e)],
- [every : $((t/p)/p)$].

Both *John loves Ann* and *Ann John walk* are phrases, the significant difference being that the only the first will be naturally associated with an arrow from $\mathbf{1}$ to t in every closed category (as we will demonstrate). This is what qualifies the former phrase to be a sentence.

Definition 3.2 Let B be given and let Lex be a lexicon over B. A *syntax-category* over Lex is a triple $< \mathcal{C}, \omega, \mu >$, where

- $\mathcal{C}$ is a closed category,
- ω is a function from B to the objects of $\mathcal{C}$, and
- μ is a function from Lex to the arrows of $\mathcal{C}$, such that when $[\alpha : a] \in$Lex, $\mu([\alpha : a])$ is an arrow from $\mathbf{1}$ to $\omega(a)$.

Note that since Typ is freely generated over the type symbols B, any function ω from B into a closed category $\mathcal{C}$ will extend uniquely to a homomorphism from all of Typ (with / and \) to $\mathcal{C}$ (with $\Leftarrow$ and $\Rightarrow$). The same name, ω, will be used for the extension.

Example. Given a lexicon, there is a particular syntax-category which arises naturally, called the syntax-category *generated by* that lexicon. The underlying closed category is, informally, the smallest closed category containing the type symbols from B as objects, and the objects of Lex as arrows. In the notation of the previous definition, we take ω and μ each to be the identity. We call this the *initial model* for syntax given by the lexicon.

Using the lexicon of the previous example, let $\mathcal{C}$ be given, and consider the initial model for syntax. Again writing p for $(e\backslash t)$, we have the following:

- μ ([John : e]) : $\mathbf{1}\rightarrow e$, and μ([Ann : e]) : $\mathbf{1}\rightarrow e$,

- μ ([walk : p]): $\mathbf{1} \rightarrow p$,
- μ ([loves : p/e]) : $\mathbf{1} \rightarrow (p/e)$
- μ ([every] : $(t/p)/p$) : $\mathbf{1} \rightarrow (t/p)/p$)

and so forth.

Notation. Given a syntax category, let us call an arrow $f : \mathbf{1} \rightarrow A$ a *term of type* A.

Unfortunately, phrases of length greater than 1 will be not associated with terms without invoking a technicality. In any closed category, if $f : \mathbf{1} \rightarrow A$ and $g : \mathbf{1} \rightarrow B$, then the paired arrow $(f * g)$ maps $(\mathbf{1} * \mathbf{1})$ to $(A * B)$. Since $(\mathbf{1} * \mathbf{1})$ is isomorphic to $\mathbf{1}$, we naturally get from $f * g$ a $\mathbf{1}$-element, that is, a term.

Notation. Use $(f \star g)$ to stand for the term just described.

Now we are in a position to associate with phrases an arrow out of $\mathbf{1}$, in a manner which extends the data in the lexicon:

Definition 3.3 Let $< \mathcal{C}, \omega, \mu >$ be a syntax-category and Σ a phrase.
Then the term corresponding to the phrase Σ, $\mu(\Sigma)$, is defined by induction on the tree underlying Σ:

1. if Σ is a single node, say $[\sigma : a]$, then
$$\mu(\Sigma) = \mu(\sigma)$$
2. if Σ has Γ and Δ as its left and right subphrases, then
$$\mu(\Sigma) = (\mu(\Gamma) \star \mu(\Delta)).$$

When a syntax-category is fixed, phrases determine terms uniquely, so we will often speak of (for example) "the term *John walks*" rather than "the term corresponding to the phrase *John walks*."

Note that *every* string corresponds to a term, whether it yields an "interesting" arrow in the syntax-category or not. For example the phrases shown in the example earlier are associated with terms as follows:

- *John (loves Ann)* corresponds to $(j \star (l \star m))$, an arrow from $\mathbf{1}$ to the object $(e * (((e \backslash t)/e) * e))$.
- *(Ann John) walk* corresponds to $(m \star (j \star w))$, a term of type $(e * (e * (e \backslash t)))$.

We have not yet explained the sense in which the first phrase qualifies as a sentence and the second "doesn't parse". We prefer to present this in the context of semantics, in the next section.

4 Semantics as a functor

The reflection of Montague's idea of a *homomorphism* between syntax and semantics in the present setting is the notion of a *functor* between a syntax-category and a semantics-category.

Definition 4.1 Let $\mathcal{C}$ be a syntax-category. A *semantics functor* is any functor $\mathcal{F} : \mathcal{C} \overset{\cdot}{\longrightarrow} \mathcal{S}$ from $\mathcal{C}$ to a closed category $\mathcal{S}$ which preserves the structure of Definition 2.1.

The value of an arrow in $\mathcal{C}$ is the image of the arrow in $\mathcal{S}$.

Informally, we will refer to the image of a semantics-functor as a semantics-category. We naturally view the image of t under a semantics functor as the object of truth-values, and will designate it T. The value of a term will be an arrow from the $\mathbf{1}$ of the semantics-category to T — such arrows are the *meanings* in our semantics-category.

We now give some examples to indicate how computation in a semantics-category might proceed, and in particular how some type shifting is inherent in the structure of closed categories. Since semantics-categories are closed, they comes equipped with arrows *app*, *curry*, etc. These are crucial in computing meanings for terms.

Some meaning computations

Consider the lexicon of the previous section, let $\mathcal{C}$ be the initial model for syntax, and fix a semantics-functor for $\mathcal{C}$. To simplify notation, suppose that the semantics-functor maps the syntax-object e to semantics-object E, etc, and that term j is mapped to semantics-arrow $\boldsymbol{j}$, and so forth.

1) Consider *John walks.* The value of this term will be

$$\boldsymbol{j} \star \boldsymbol{w} : \mathbf{1} \rightarrow (E * (E \Rightarrow T)).$$

We also have $app_{E,T} \circ (\boldsymbol{j} \star \boldsymbol{w}) : \mathbf{1} \rightarrow T$. So the value of *John* and the value of *walk* combine via application to yield a T-value.

Another construction is to consider $\lambda \boldsymbol{j}$, which is explicitly an arrow from E to T. This *composes* with $\boldsymbol{w}$ to yield an arrow from $\mathbf{1}$ to T. Proposition 2.5 assures us that these are the same map.

2) Next consider *(Every man) walks.* This will have value

$$(\boldsymbol{v} \star \boldsymbol{m}) \star \boldsymbol{w} : \mathbf{1} \rightarrow ((((T \Leftarrow P) \Leftarrow P) * P) * P),$$

which reduces to a $\mathbf{1}$-element of T by two uses of app'.

3) No type-lifting was involved the first example above. The fact that the same word can sometimes behave as a function and sometimes as an argument, without changing type, is built into the system. Type-lifting is available, though, in a natural way. We present Montague's treatment of names as an example. We have already seen that $Hom((E * (E \Rightarrow T)), T)$ is not empty — it contains $app_{E,T}$. Now by $curry'$, we conclude that $Hom(E, T \Leftarrow (E \Rightarrow T))$ is not empty, in *every* closed category.

Let us denote by *mont* the map which is $curry'(app_{E,T})$. So $mont \circ \boldsymbol{j} : \mathbf{1} \to (T \Leftarrow (E \Rightarrow T))$. This is the generalized quantifier meaning of *John*. We calculate:

$$(mont \circ \boldsymbol{j}) \star \boldsymbol{w} : \mathbf{1} \to (T \Leftarrow (E \Rightarrow T)) * (E \Rightarrow T),$$

and so app' immediately returns a $\mathbf{1}$-element of T. The naturality of all of our canonical arrows implies that this gives the same result as the first one. The use of app' rather than *app* signals the fact that after the type-shift on *John*, the function is on the left of its argument.

4) Finally, look at the term introduced earlier corresponding to the string *(Ann John) walk.* This will get a value which is a map from $\mathbf{1}$ to $((E * E) * (E \Rightarrow T))$. There is no natural map which will interact with this. Of course one always has "non-standard models", categories in which there are arrows with which this phrase's value may combine, even models in which this phrase may be a component of a "truth value" arrow. But a string will parse correctly as a sentence in the present conception precisely when there are maps in *every* model which lead it to a truth value.

This last example indicates why we *do not* want to take cartesian closed categories for syntax, and in particular, why there should not be definable projections from $(X * Y)$ to X and to Y. If there were, we could apply these to the type $(E * E)$ inside of $((E * E) * (E \Rightarrow T))$ above, shifting this latter type to $(E) * (E \Rightarrow T)$, then evaluate to T as usual. The effect of such a projection is to *ignore* one of the words in the sentence. In general, if there is a projection map from a type $(X * Y)$, say to X, then a phrase of type y, when combined with an x-type, can have no effect! This doesn't seem to happen with any regularity in language.

The Lambek calculus

Every model comes equipped with some arrows, such as *id*, *app*, *mont*, etc., just by virtue of its being a closed category, We might call these the *valid*, or the *logical* arrows. The lexicon determines other arrows as interpretations of the "dictionary" values; these might be described as "contingent". Meaning in a model is then computing with the arrows at hand - the contingent meanings in the language interacting with the logical arrows.

The question naturally arises - what are the logical arrows? The answer is, precisely those derivable in the Lambek calculus. The argument is based on general considerations of universal algebra, if we define the class of closed categories as an equational variety. A very clear presentation of this point of view in the context of *cartesian closed* categories will be found in [LS86]. The modifications required to treat (simply) closed categories are routine.

Hint of proof of Completeness: For type expressions x and y, there will be a map in all closed categories from $\omega(x)$ to $\omega(y)$ just when there is a closed Σ-term of type $x \to y$. But this is just what "valid" will mean, when x and y are (paired types corresponding to) the left and right sides of a Lambek sequent. The result follows by a Curry-Howard style "types as formulas" observation: the rules for building Σ terms mimic the proof rules exactly.

Completeness of the Lambek calculus for a different notion of semantics is found in [6].

Thus the theory of syntax outlined here is first-order, even equational (and decidable, as shown by Lambek). Passing to viewing strings as arrows in a category provides a refined point of view on the type shifting of categorial grammar, in that the canonical type shifts whose existence is licensed there are all *definable* by terms in the equational logic.

Richer categories

The present paper has adopted a moderate approach; we only committed ourselves to the assumptions necessary to make sense of the notion of meanings as maps between types. In order for this setting to incorporate the insights of linguistics research, each of the syntax-categories and semantics-categories will need to be enriched. Some initial ideas:

The interpretation of the basic types certainly will admit more structure. Probably the closed categories serving for semantics should have a designated object Ω to serve as the interpretation of the type t, and axioms formulated to induce Ω to behave properly. The obvious way to arrange this would be to use *toposes* as semantics-categories. In a topos, Ω is the *subobject classifier*, which is the topos theory notion of "set of truth values".

To begin to treat intensionality, we might designate an object S in each category to correspond to a possible worlds index type s in the lexicon.

Another idea for treating intensionality which is compelling at first glance is to use functor categories. These are categories of the form $SET^{\mathcal{A}}$, where $\mathcal{A}$ is a small category. A standard intuition is to view the objects of $\mathcal{A}$ as states of knowledge, and the objects of $SET^{\mathcal{A}}$ as variable sets, with truth in $SET^{\mathcal{A}}$ being parameterized by the structure on the states imposed by $\mathcal{A}$. The nature of truth in a functor category, however, is such that if we are to treat the the objects of $\mathcal{A}$ as possible worlds, there can be a map (in $\mathcal{A}$) between worlds A and B only if the true assertions in world A (as reflected in $SET^{\mathcal{A}}$) are contained in the true assertions in world B. So, for instance, the maps in $\mathcal{A}$ should not correspond to the passage of time.

On the syntactic side, the presence of a pairing operator in the type syntax now permits a treatment of functions of more than one variable, for example transitive verb phrases as functions of two e arguments, or determiners taking two $((t\backslash e)\backslash e)$ arguments. Our categories are already set up to interpret a word whose intuitive type is, say, "from $((t\backslash e)\backslash e)$ and $((t\backslash e)\backslash e)$ to t", as an arrow from $(T{\Leftarrow}E){\Leftarrow}E) * (T{\Leftarrow}E){\Leftarrow}E)$ to T.

Extensions to the type lifting mechanism may also be in order, especially in the presence of several-variable functions. One deficiency of the Lambek calculus is that it does not provide for the shift of *and* or *or* to higher types. If we assume that there is a diagonal functor $\mathcal{D}$, sending object X to $(X * X)$ we can justify type lifting on types that are (intuitively) functions of two variables. For instance, maps such as *and* and *or* from $(T * T)$ to T will then naturally lift to $((E{\Rightarrow}T) * (E{\Rightarrow}T))$ to T.

In any event, one of the key features of the program outlined in this paper is that a change in one of syntax or semantics *automatically* has consequences for the other if we require meaning to be a functor. The idea to build the nature of the this link into the foundations is Montague's, of course, but it came at the cost of a somewhat rigid theory of syntax.

We have tried to suggest a foundation with enough structure to coordinate the insights of researchers in syntax and in semantics.

References

[1] Ades, A. E., and M. J. Steedman, *On the Order of Words*, Linguistics and Philosophy 4, 517-558, 1982.

[2] Ajdukiewicz, K. Die syntaktische Konnexität, *Studia Philosophica* 1, pp. 1-27, 1935.

[3] Bach, E. Some Generalizations of Categorial Grammars. In Landman and Veltman (eds), *Varieties of Formal Semantics*, Foris, Dordrecht (GRASS series, V. 3), pp. 1-23, 1984.

[4] Bar-Hillel, Y. A Quasi-Arithmetical Notation for Syntactic Description, *Language* 29, pp.47-58, 1953.

[5] van Benthem, J. *Essays in Logical Semantics*, Reidel, 1986.

[6] Buszkowski, W. Completeness Results for Lambek Syntactic Calculus, *Zeitschr. fur math. Logik und Grunclagen d. Math. 32*, 13-28, 1986.

[7] Dowty, D. R., R. E. Wall, and S. Peters. *Introduction to Montague Semantics*, Reidel 1981.

[8] Eilenberg, S., and G. M. Kelly. Closed Categories, *Proceedings of the Conference on Categorical Algebra, La Jolla 1965*, Springer-Verlag, 421-562, 1966.

[9] Lambek, J. The Mathematics of Sentence Structure, *American Mathematical Monthly 65*, 154-170, 1958.

[10] Lambek, J. On the Calculus of Syntactic Types, in R. Jakobsen, ed., *Structure of Language and its Mathematical Aspects*, Proceedings of Symposia in Applied Mathematics XII, American Mathematical Society, 1961.

[11] Lambek, J. Deductive systems and categories III, Lecture Notes in Mathematics 274, 57–82.

[12] Lambek, J. Functional completeness of cartesian closed categories, Annals of Math. Logic 6 (1074), 259–292.

[13] Lambek, J. Categorial and Categorical Grammars. In Oehrle, Bach, and Wheeler (eds), *Categorial Grammars and Natural Language Structures*, Reidel 1988.

[LS86] Lambek, J., and P. J. Scott. *Introduction to Higher Order Categorical Logic*, Cambridge University Press, Cambridge, 1986.

[14] Keenan, E. L. and L. M. Faltz. *Boolean Semantics for Natural Language*, Reidel, 1986.

[15] Montague, R. Universal Grammar, *Theoria* 36, pp. 373- 398, 1970. Reprinted in [20].

[16] Montague, R. The Proper Treatment of Quantification in Ordinary English, In J. Hintikka et. al. (eds.), *Approaches to Natural Language*, Dordrecht, 1973. Reprinted in [20].

[17] MacLane, S. *Categories for the Working Mathematician*, Graduate Texts in Mathematics 5, Springer-Verlag, 1971.

[18] Meseguer, J., and J. A. Goguen. Initiality, Induction, and Computability, in M. Nivat and J. Reynolds, eds., *Algebraic Methods in Semantics*, Cambridge University Press, 1984.

[19] Partee, B., and M. Rooth. Generalized Conjunction and Type Ambiguity, in R. Baurle, C. Schwarze, and A. von Stechow, eds., *Meaning, Use, and Interpretation of Language*, de Gruyter, 361-383, 1983,

[20] Thomason, R. H. (ed.) *Formal Philosophy: Selected Papers of Richard Montague*, Yale University Press, 1974.