Language, Lambdas, and Logic[*]

Reinhard Muskens

1 Introduction

In (van Benthem 1986) it was observed that the Curry-Howard correspondence between proofs and λ-terms can be exploited to obtain a very elegant and principled match between Lambek Categorial Grammar and Montague Semantics. The correspondence associates each proof of the calculus with a λ-term and Van Benthem shows how such terms can be used as a recipe for obtaining the meaning of a complex expression in terms of the meanings of its parts. The method is easily extended to various other forms of Lambek calculi, including multimodal calculi (see (Moortgat 1997) and references therein).

Van Benthem's original work concerned the undirected Lambek Calculus, for which the Curry-Howard correspondence is an isomorphism, but the use of undirected calculi has not caught on. Superficially it seems that using an undirected calculus entails that the permutations of any string that is predicted to be grammatical are likewise predicted to be grammatical and understandably categorial grammarians therefore have preferred directed systems. However, from (Oehrle 1994, Oehrle 1995) it is apparent that the permutation difficulty can be overcome by radicalising the approach. This is done by using the Curry-Howard isomorphism not only for obtaining a λ-term representing the semantics of an expression, but also for getting a λ-term representing its syntax. As will be argued below, the permutation problem then vanishes, for any effect of permutation on the semantic term is reflected by a similar effect on the syntactic term and vice versa. Since syntactic and semantic representations can only permute in tandem, they will not be coupled in an undesirable way.

Letting grammatical representations consist of λ-terms for (at least) syntax and semantics brings us into the realm of the 'sign-based' approaches

[*]From: Geert-Jan Kruijff and Richard Oehrle, editors, *Resource Sensitivity in Binding and Anaphora*, Studies in Linguistics and Philosophy. Kluwer, 2003, pp. 23–54.

to categorial grammar that have arisen in the last two decades ((Zeevat, Klein and Calder 1986, Oehrle 1988, Moortgat 1991b, Morrill 1994, Moortgat 1997)). The basic data structures of the grammar are multidimensional objects ('signs') in such a view; i.e. they are tuples $\langle M_1, \dots, M_n \rangle$, where each M_i is a representation in some component of the grammar (e.g. syntax or semantics) and n is the dimensionality of the grammar. The lexicon consists of such signs and some set of rules tells how signs are to be combined into more complex signs. If such a perspective is taken we are immediately confronted with a series of questions concerning the further design of the grammatical system.

1. In a sign $\langle M_1, \dots, M_n \rangle$, what is the nature of the representations M_i? I will take a radical approach here and suggest that *all* M_i are λ-terms.

2. What is the logic underlying the combination of signs? For example, if $\langle M_1, \dots, M_n \rangle$ and $\langle N_1, \dots, N_n \rangle$ are grammatically derivable and the terms in these sequences are appropriately typed, we may perhaps expect $\langle M_1(N_1), \dots, M_n(N_n) \rangle$ to also be derivable. What other principles should be adopted?

3. What are the logics of the various dimensions? If M_i and N_i are both from the i-th component of the grammar and are of the same type, there may be an entailment holding between them. What logics govern these entailment relations?

4. What part of the grammar is going to be responsible for any given linguistic task we are confronted with? This is an overall question whose answer will strongly affect answers to the previous ones.

That there is room for answering such questions in radically different ways becomes apparent when multidimensional versions of the directed Lambek Calculus, such as the ones in (Morrill 1994, Moortgat 1997), are compared with Oehrle's undirected version. The classical Lambek Calculus **L** deals with two things simultaneously: 1) the combinatorics of language, and 2) word order. Non-associative versions, such as **NL**, also deal with 3) constituency. The modern versions of Morrill and Moortgat essentially inherit this trait and the logic that combines signs also takes care of word order (and possibly constituency) in these systems. Oehrle, on the other hand, allocates the task of keeping track of word order (and constituency) to the syntactic component exclusively and factors it out from the combining calculus, thus arriving at an undirected system for the latter.

In this chapter I want to further pursue the intuition that it is really an undirected system that models the basic combinatorics of language. Word order and constituency should be dealt with on a separate level. I will discuss a grammatical formalism (Lambda Grammars) that allows one to combine signs that are sequences of λ-terms with the help of linear combinators (essentially closed pure λ-terms in which each abstractor binds exactly one variable). The grammar is a form of categorial grammar, but unlike standard categorial formalisms, such as Combinatory Categorial Grammar or the Lambek Calculus, it is non-directional and does not, on the level that deals with the basic combinatorics of language, need to make any use of derivations. Thus, my answer to question 2 above is that signs are combined with the help of linear combinators. It may be linguistically meaningful to narrow down the class of possible combinators, but I will not consider such refinements here.

The system has several virtues. One is simplification. Instead of the long, often computer-generated, proofs that are considered in the usual Lambek Categorial Grammars, the working linguist need only consider certain λ-terms that sum up such proofs much more concisely. A second virtue is greater modularity. As indicated above, the grammar distinguishes between the logic behind the combination of signs and the logics that are associated with the various dimensions of these signs. Each of these can be studied independently. A third advantage is linguistic. As has often been noted (e.g. in (Moortgat 1997)) and as will be explained below, the classical Lambek Calculus suffers from a nasty 'periphery problem'. The calculus can model extraction with the help of hypothetical reasoning, but only if the extraction site was on the periphery of a relevant subexpression. The reason behind this, informally, is that expressions can only subcategorize for expressions that lack material on their peripheries. This problem can be overcome in various ways, but always at the price of complication in directional systems. In undirected systems the 'periphery problem' does not arise. As will become clear shortly, the periphery of expressions does not play any privileged role in the present set-up.

The rest of this chapter is set up as follows. In the following section a brief historical introduction to type logical grammars is given; it is shown in more detail how directed versions suffer from a periphery problem and undirected versions from a permutation problem; the idea of having λ-terms in syntax is traced back to (Curry 1961); and we recap the Curry-Howard-Van Benthem correspondence between proofs and terms. In section 3 the essential idea behind Lambda Grammars is explained and the formalism is compared with recent independent work by Philippe de Groote on 'Abstract

Categorial Grammars'. Section 4 then continues with a closer look at possible ways to set up a particular Lambda Grammar, filling in some design choices. In particular we will opt for a three dimensional grammar there; one component will deal with dominance and precedence, one with semantics, and one with syntactic features. These choices bring us in close contact with the traditional architecture of Lexical-Functional Grammar (LFG, (Kaplan and Bresnan 1982), for further connections with LFG see (Oehrle 1999) and (Muskens 2001a), which is based upon the present system) and indeed the LFG architecture inspires our answer to question 4 above. Section 4 also works out the logics of the three grammatical components in some detail and thus illustrates one possible set of answers to question 3. For the semantic component we choose a standard type logic with possible worlds; for the feature component a type logic over the first-order theory of features ((Johnson 1991)); and the multimodal approach to grammar that is found in most modern versions of the Lambek Calculus (see (Moortgat 1997) and references therein) will serve as a basis of the component dealing with dominance and precedence. The multimodal approach is thus moved from the general level of combing signs to one of the special dimensions of the grammar, another illustration of the modularity of the set-up. The chapter ends with a short conclusion.

2 Categorial Calculi

2.1 Directed and Undirected Calculi

Although categories in categorial grammar historically derive from Husserl's *Bedeutungskategorien* and Leśniewski's semantic categories, and although the order of words is commonly taken to be semantically irrelevant, traditionally most systems of categorial grammar distinguish between functors that seek arguments to their left and those that want them on their right. An exception to this rule is the system of (Ajdukiewicz 1935), in which functors only seek arguments on their right, but which allows permutations of functors and arguments. Categories are represented as fractions $A/B_1 \cdots B_n$ in this system and the basic simplification rule is given by $(A/B_1 \cdots B_n)B_1 \cdots B_n \leadsto A$, a rule familiar from elementary arithmetic. Products commute (i.e. $AB = BA$), but are *not* associative. Using these rules, it is easily shown that Ajdukiewicz's example (1), with categorisation as indicated and with the usual surface bracketing, as in (2), is connex and obtains category s.

(1)

Die	Flieder	duftet	sehr	stark	und	die	Rose	blüht
$\frac{n}{n}$	n	$\frac{s}{n}$	$\frac{\frac{\frac{s}{n}}{\frac{s}{n}}}{\frac{\frac{s}{n}}{\frac{s}{n}}}$	$\frac{\frac{s}{n}}{\frac{s}{n}}$	$\frac{s}{ss}$	$\frac{n}{n}$	n	$\frac{s}{n}$

(2) [[[Die Flieder][duftet [sehr stark]]] und [[die Rose] blüht]]

Unfortunately, as a consequence of the commutativity of the product in Ajdukiewicz's system, many ungrammatical permutations of (1) will also be predicted to be connex.

The first bidirectional system of categorial grammar was defined in (Bar-Hillel 1953). Bar-Hillel distinguishes between functor categories A/B, which seek their argument B to the right, and categories $B\backslash A$, which seek it to the left.[1] With the help of the rules $(A/B)B \rightsquigarrow A$ and $B(B\backslash A) \rightsquigarrow A$ we find that (3) (this and some of the following examples are taken from (Moortgat 1997)) is connex. Products now are associative, i.e. we have $[AB]C = A[BC]$ and bracketings such as in the previous example can be dispensed with. But products are no longer commutative and permutations of expressions that are connex are no longer predicted to be necessarily connex themselves.

(3)

Kazimierz	talks	to	the	mathematician
np	$(np\backslash s)/pp$	pp/np	np/n	n

Word order is dealt with on the level of grammatical categories in Bar-Hillel's system and this property is retained in most modern versions of categorial grammar. This solves the permutation problem, but the solution comes at a price. Linguistically, one of the great attractions of categorial grammar is the high degree in which the system is lexicalised. Grammatical properties essentially are properties of *words* in the lexicalist's view. They are located in the lexicon, not in generative rules that form complex expressions out of simpler ones. Categorial grammar conforms to this ideal to a very high degree. Its rules of combination are few and simple and it is on the level of words that things are really happening. But Bar-Hillel's treatment of word order by means of a directional calculus necessitates an exception to this general scheme and requires special measures on the level of the calculus itself. It is not sufficient to categorise, say, *talks* as an $(np\backslash s)/pp$, signalling that the word takes a PP to its right and an NP to its left. In order for this to work we also need *two* dual rules of simplification, one for / and one for \. This means that the treatment of word order is distributed over the

[1]This is the notation of (Lambek 1958). Bar-Hillel wrote $A/[B]$ for A/B and $A/(B)$ for $B\backslash A$, but later prefered Lambek's notation.

lexicon and the rule system now, a situation not in strict agreement with the lexicalist ideal.

Ajdukiewicz's and Bar-Hillel's rules of simplification resemble Modus Ponens, as well as the familiar arithmetical rule they were inspired by. The great leap forward of defining a logical calculus for category combination was made by (Lambek 1958), who defined a system partly shown in (4) (here in a natural deduction presentation). The rules $/E$ and $\backslash E$ in this calculus are variants of the usual elimination rule for $\rightarrow$ in the propositional calculus and thus closely correspond to Modus Ponens. A main difference between Lambek's calculus and the usual calculus for intuitionistic propositional logic is that there are no structural rules. In particular, permutations, contractions and weakenings are not allowed in this system. The reader will have no difficulty in showing that $np, (np\backslash s)/pp, pp/np, np/n, n \Rightarrow s$ is provable from axioms of the form $A \Rightarrow A$ with the help of $/E$ and $\backslash E$ alone. This then gives a purely logical proof of the connexity of (3).

(4)
$$\frac{}{A \Rightarrow A}\,[Ax]$$

$$\frac{\Gamma, A \Rightarrow B}{\Gamma \Rightarrow B/A}\,[/I] \qquad \frac{\Gamma \Rightarrow A/B \quad \Delta \Rightarrow B}{\Gamma, \Delta \Rightarrow A}\,[/E]$$

$$\frac{A, \Gamma \Rightarrow B}{\Gamma \Rightarrow A\backslash B}\,[\backslash I] \qquad \frac{\Gamma \Rightarrow B \quad \Delta \Rightarrow B\backslash A}{\Gamma, \Delta \Rightarrow A}\,[\backslash E]$$

But Lambek's calculus is not merely a purely logical reformulation of the Ajdukiewicz–Bar-Hillel system. The presence of the introduction rules $/I$ and $\backslash I$ takes us from a grammar that can only handle local dependencies to one in which long distance dependencies are also naturally dealt with. For example, since $np/n, n, (n\backslash n)/(s/np), np, (np\backslash s)/pp, pp/np \Rightarrow np$ is provable, one also easily verifies that (5) is connex. Informally, since *Kazimierz talks to* followed by any np is an s, we find by $/I$ that this string itself is an s/np, and further applications of the elimination rules give the desired result. Since *whom* subcategorizes for a sentence missing a rightmost np and *Kazimierz talks to* is such a sentence, the connexity is established.

(5)	the	man	whom	Kazimierz	talks	to
	np/n	n	$(n\backslash n)/(s/np)$	np	$(np\backslash s)/pp$	pp/np

This elegant and principled treatment of long distance dependencies unfortunately leaves something to be desired, as (Moortgat 1997) and others have pointed out. Since words can only subcategorize for complements with gaps on their periphery, the theory as it stands predicts that non-peripheral gaps cannot occur. This is wrong of course. In (6), for example, the gap is naturally situated between *to* and *yesterday.*

(6) the man whom Kazimierz talked to yesterday

Solutions to this problem have been offered within the categorial paradigm. One way of dealing with it ((Morrill 1994, Moortgat 1997)) is by using some multimodal variant of the original Lambek Calculus, allowing permutations in certain controlled circumstances. But this solution requires certain complications of the theory and it is worth noting that, while Lambek's idea of adding Conditionalisation to the system immediately leads to a principled way of handling long distance phenomena, the calculus' directionality essentially spoils things. Without additional machinery, gaps are predicted to be always peripheral.

The directional approach was criticized in (Curry 1961), who proposed an interesting alternative. Curry considers *functors*, which are expressions containing subscripted blanks, such as '$—_1$ is between $—_2$ and $—_3$' or '$—_1$ were eaten by the children'. Functors can apply to arguments and arguments are to be substituted for blanks in the order of the subscripts. Essentially then, although Curry does not explicitly mention this, functors are λ-terms over syntactic objects. For example, the first of the functors just mentioned can also be written '$\lambda x \lambda y \lambda z.\, x$ is between y and z'. Curry defines a type hierarchy of functors and also considers 'what Harris and Chomsky call transformations' to be functors. Clearly, while the directional approach restricts itself to functors with gaps on the periphery, this more general approach is not so restricted.[2] We shall come back to Curry's proposal below and adopt it as essentially right.

Evidence for the view that some form of non-directional combinatorics plays an important role in grammar also comes from semantics. In Montague's pivotal work ((Montague 1973)) it was already implicit that semantic values of certain expressions were 'shifted' to values in higher types in order to get the combinatorics of the system right. For example, since quantifying noun phrases (in an extensional version of the theory) are treated as expressions of type $(et)t$, proper names are also treated as being of this type and the translation of *John* (say) is $\lambda P.P(john)$, not simply the constant *john*

[2](Curry 1961): 'A functor is *any* kind of linguistic device which operates on one or more phrases (the argument(s)) to form another phrase. A functor may, conceivably, so modify its arguments that even the notations involving blanks are inadequate to describe it.'

of type e. In a similar vein, since transitive verbs need to be combinable with noun phrases (type $(et)t$) in object position, they get type $((et)t)(et)$, not simply $e(et)$. Implicit, therefore, in Montague's system there are shifts $e \Rightarrow (et)t$ and $e(et) \Rightarrow ((et)t)(et)$.

These type raisings are 'compiled in' in Montague's original set-up. Lexical translations are simply given in the higher type, not in the simpler one. This leads to a certain awkwardness of the translations, but that in itself would perhaps be acceptable if the 'compiling in' strategy always worked, i.e. if it were always possible to avoid type shifts during the combination process by choosing high enough types in lexical entries (a strategy known as 'generalizing to the worst case'). That this is in fact not possible was shown in (Rooth and Partee 1982, Partee and Rooth 1983) on the basis of an argument taking Montague's treatment of intensionality as its point of departure, together with the widely held view that the words *and* and *or* can function as intersection and union in all categories.

If type shifts need to enter into the combinatorial process, we need rules for them, and Partee and Rooth give a simple rule system that was later generalised in (Hendriks 1988, Hendriks 1993). However, in a move reconnecting Categorial Grammar with its semantic origins, Van Benthem observed ((van Benthem 1986, van Benthem 1988, van Benthem 1991)) that the necessary type shifts are available in a non-directional version of the Lambek Calculus. This version, called **L*P**, identical in fact to the implicational fragment of Intuitionistic Linear Logic ((Girard 1987, Troelstra 1992)), is shown in (7), again in the natural deduction formulation. The system contains an introduction (I) and an elimination (E) rule for the type forming operator. Again, sequents are provable if they can be derived from axioms of the form $A \Rightarrow A$ in the usual way. A main difference with the Lambek Calculus in (4), however, is that left-hand sides of sequents are now taken to be *bags* (multisets), i.e. they are invariant under permutations. The left-hand sides of sequents are allowed to be empty here. If the restriction is made that the left-hand sides of sequents should always be non-empty, the (more familiar) system **LP** is arrived at.

(7)
$$\frac{}{A \Rightarrow A}\,Ax$$

$$\frac{\Gamma, A \Rightarrow B}{\Gamma \Rightarrow (AB)}\,I \qquad\qquad \frac{\Gamma \Rightarrow (AB) \quad \Delta \Rightarrow A}{\Gamma, \Delta \Rightarrow B}\,E$$

The motivation for these rules is very similar to the motivation for the rules of the syntactic Lambek Calculus discussed above. The basic way of combining typed λ-terms is by functional application when a term of type AB (also written $A \rightarrow B$ or $A \multimap B$) is combined with a term of type A, forming a term of type B. Essentially, this is a form of Modus Ponens and, in a natural deduction formulation, leads to the rule E. Complementing E with a rule I is, with hindsight, an obvious thing to do from a logical point of view, but is the essence of Lambek's contribution and the motor behind all type change. That left-hand sides of sequents are bags here, and that consequently there is only one implication, reflects that there is no order in semantics.

In (8) a proof is given for $e, e(et), (et)t \Rightarrow t$ in this calculus. The example shows that the semantic values for the words in *Kazimierz loves someone* combine to a logical sentence, if we take these semantic values to be of types e, $e(et)$, and $(et)t$ respectively, i.e. if we assign each word its simplest and most obvious translation. (For the question *how* translations combine, see below.)

(8)
$$\dfrac{(et)t \Rightarrow (et)t \qquad \dfrac{\dfrac{\dfrac{e(et) \Rightarrow e(et) \quad e \Rightarrow e}{e(et), e \Rightarrow et}\,E \qquad e \Rightarrow e}{e, e(et), e \Rightarrow t}\,E}{e, e(et) \Rightarrow et}\,I}{e, e(et), (et)t \Rightarrow t}\,E$$

It is attractive to assume that some form of the Lambek Calculus not only plays a role in syntax, but also in semantics. In syntax, the system can give a very elegant treatment of filler-gap constructions such as in the wh-phrase in (5). In semantics, it explains the type shifts that are needed to get the combinatorics working. This all points to a common combinatorial engine for syntax and semantics. But if the syntactic calculus is directed and the semantic calculus is not, how are we to combine the two? One answer, the answer that is essentially given in the multimodal approach, is to form some hybrid. As was already discussed in the introduction, the answer that will be pursued here is that the basic combinatorics of the grammar is non-directional but that word order should not be treated in the basic combinatorics. We will assume that all grammatical signs have a syntactic dimension dealing with word order.

2.2 The Curry–Howard–Van Benthem Correspondence

Above we have seen how the Lambek–Van Benthem calculus **L*P** can combine the types of translations of lexical expressions, but the really interesting question from a semantical point of view of course is how these translations themselves combine. We have associated the E rule with application and it is only fitting then that the I rule should be associated with the dual of application, λ-abstraction. In fact this conforms to the standard correspondence in proof theory that goes back to work of Curry, Howard, and De Bruyn in the 1960's. For many logical systems, it is possible to define an isomorphism between proofs and λ-terms, such that Modus Ponens is associated with application, conditionalisation with abstraction, and conjunction introduction with pairing (while proof normalization corresponds to β-conversion). An essential contribution of (van Benthem 1986, van Benthem 1988, van Benthem 1991) consisted in the realisation that the Curry–Howard isomorphism for **L*P** and other Lambek calculi provides a recipe for getting the meaning of a complex expression in terms of the meanings of its parts.

The Curry–Howard isomorphism for **L*P** can be obtained by annotating each type in a proof with a λ-term. Axioms are now of the form $x\colon A \Rightarrow x\colon A$ (where the variable x is fresh to the proof and of type A), the introduction rule corresponds to abstraction, and the elimination rule corresponds to application. (9) gives the annotated rules.

(9)
$$\frac{}{x\colon A \Rightarrow x\colon A}$$

$$\frac{\Gamma, x\colon A \Rightarrow M\colon B}{\Gamma \Rightarrow \lambda x.M\colon (AB)}\,I \qquad\qquad \frac{\Gamma \Rightarrow M\colon (AB) \quad \Delta \Rightarrow N\colon A}{\Gamma, \Delta \Rightarrow M(N)\colon B}\,E$$

By way of example, (10) shows the annotated form of the proof in (8). The resulting endsequent $x\colon e,\ R\colon e(et),\ Q\colon (et)t \Rightarrow Q(\lambda y.R(y)(x))\colon t$ can be used as a meaning recipe in the following way. Suppose the translations of the words in a certain sentence S are M_1, M_2, and M_3, of types e, $e(et)$, and $(et)t$ respectively. Then S itself translates as

$$Q(\lambda y.R(y)(x))[x := M_1, R := M_2, Q := M_3]\ ,$$

or,

$$M_3(\lambda y.M_2(y)(M_1))\ .$$

For example, if *Kazimierz* translates as k, *loves* as $\lambda y \lambda x.love(x, y)$, and *someone* as $\lambda P \exists x[person(x) \wedge P(x)]$ then the translation of *Kazimierz loves someone*, after conversions, is $\exists x[person(x) \wedge love(k, x)]$, as in (11).

(10)
$$\dfrac{Q\colon (et)t \Rightarrow Q\colon (et)t \qquad \dfrac{\dfrac{\dfrac{R\colon e(et) \Rightarrow R\colon e(et) \quad y\colon e \Rightarrow y\colon e}{R\colon e(et),\ y\colon e \Rightarrow R(y)\colon et}\,E \qquad x\colon e \Rightarrow x\colon e}{x\colon e,\ R\colon e(et),\ y\colon e \Rightarrow R(y)(x)\colon t}\,E}{x\colon e,\ R\colon e(et) \Rightarrow \lambda y.R(y)(x)\colon et}\,I}{x\colon e,\ R\colon e(et),\ Q\colon (et)t \Rightarrow Q(\lambda y.R(y)(x))\colon t}\,E$$

(11) $k\colon e,\ \lambda y \lambda x.love(x,y)\colon e(et),\ \lambda P \exists x[person(x) \wedge P(x)]\colon (et)t \Rightarrow$
$\exists x[person(x) \wedge love(k,x)]\colon t$

Clearly, each permutation of *Kazimierz loves someone* (and in particular *someone loves Kazimierz*) will obtain the same translation, which is obviously wrong. But below it will be explained how this closure under syntactic permutation can be gotten rid of.

In order to get a more precise grip on the Curry–Howard isomorphism we need some definitions. Firstly, let us define the notion of a *pure linear* λ*-term*. The set $\mathsf{fv}(M)$ of the free variables of such a term M is defined simultaneously.

i. If X is a variable of type A, X is a pure linear λ-term of type A and $\mathsf{fv}(X) = \{X\}$;

ii. If M is a pure linear λ-term of type AB and N is a pure linear λ-term of type A such that $\mathsf{fv}(M) \cap \mathsf{fv}(N) = \varnothing$, then $M(N)$ is a pure linear λ-term of type B and $\mathsf{fv}(M(N)) = \mathsf{fv}(M) \cup \mathsf{fv}(N)$;

iii. If M is a pure linear λ-term of type B, X is a variable of type A and $X \in \mathsf{fv}(M)$, then $\lambda X.M$ is a pure linear λ-term of type AB and $\mathsf{fv}(\lambda X.M) = \mathsf{fv}(M) - \{X\}$.

In other words a simply typed λ-term M is a pure linear λ-term if it is built up from variables using application and abstraction only and each abstractor λx in M binds exactly one variable x in M, while each variable that is free in M occurs only once in M.

A sequent $x_1\colon A_1, \ldots, x_n\colon A_n \Rightarrow M\colon A$ will be called linear if each x_i is of type A_i while M is a pure linear λ-term of type A such that $\mathsf{fv}(M) = \{x_1, \ldots, x_n\}$. It is easily seen that **L*P** provable sequents are linear. On the other hand, a simple induction shows that for any linear sequent there is a unique **L*P** proof. Since any linear pure λ-term M uniquely corresponds to a linear sequent $x_1\colon A_1, \ldots, x_n\colon A_n \Rightarrow M\colon A$, this establishes that the Curry–Howard–Van Benthem correspondence is an isomorphism in the case of **L*P**:

the term M obtained in an endsequent is a shorthand for the whole proof. The correspondence can clearly also be defined for Lambek's original calculus **L**, partly shown in (4), with the left rules corresponding to application and the right rules to abstraction, but in this case no isomorphism is obtained.[3]

One advantage of having the Curry–Howard–Van Benthem isomorphism at one's disposal is of a practical rather than of a theoretical nature: It allows the working linguist to dispense with derivations altogether. Since linear sequents correspond to **L*P** proofs in a one-to-one fashion we can do away with the proofs and retain the meaning recipes. Surely, it is much easier to check the provability of $x\colon e,\ R\colon e(et),\ Q\colon (et)t \Rightarrow Q(\lambda y.R(y)(x))\colon t$ by checking its linearity than by unfolding the proof in (10). Similarly, it is convenient that we can check at a glance that (12a) can be provided with the semantic recipes in (12b) and (12c). Both obviously are linear. If the usual translations for the sentence's words are plugged in, its $\exists\forall$ and $\forall\exists$ readings are obtained. Compare this with the procedure that needs to be followed in the case of the directional calculus **L**. Checking the provability of (12d) and (12e) in a suitably annotated version of **L** seems to crucially depend on actually unfolding the proof. At present there seems to be no procedure for checking provability in **L** that requires less work than that.

(12) a. Every man loves a woman

b. $D\colon (et)((et)t),\ P\colon et,\ R\colon e(et),\ D'\colon (et)((et)t),\ P'\colon et \Rightarrow$
$D(P)(\lambda v.D'(P')(\lambda v' R(v')(v)))\colon t$

c. $D\colon (et)((et)t),\ P\colon et,\ R\colon e(et),\ D'\colon (et)((et)t),\ P'\colon et \Rightarrow$
$D'(P')(\lambda v'.D(P)(R(v')))\colon t$

d. $D\colon (s/(np\backslash s))/n,\ P\colon n,\ R\colon (np\backslash s)/np,\ D'\colon ((s/np)\backslash s)/n,$
$P'\colon n \Rightarrow D(P)(\lambda v.D'(P')(\lambda v' R(v')(v)))\colon s$

e. $D\colon (s/(np\backslash s))/n,\ P\colon n,\ R\colon (np\backslash s)/np,\ D'\colon ((s/np)\backslash s)/n,$
$P'\colon n \Rightarrow D'(P')(\lambda v'.D(P)(R(v')))\colon s$

f. $D\colon (et)((et)t),\ P\colon et,\ R\colon e(et),\ D'\colon (et)((et)t),\ P'\colon et \Rightarrow$
$D(P')(\lambda v.D'(P)(\lambda v' R(v')(v)))\colon t$

g. $D\colon (et)((et)t),\ P\colon et,\ R\colon e(et),\ D'\colon (et)((et)t),\ P'\colon et \Rightarrow$
$D(P)(\lambda v'.D'(P')(R(v')))\colon t$

[3] It is possible to move to a variant of the standard lambda calculus with directed notions of application and abstraction and thus retain the isomorphism. But the semantic relevance of directed application and abstraction is unclear.

Of course it should be kept in mind that at this point there are also linear sequents that do not correspond to acceptable translations of (12a) in the way that (12b) and (12c) do; (12f) and (12g) are two examples and the reader will have no difficulty in finding some others. This is the permutation problem again, to which we will turn in the next section.

That (12d) and (12e) have proofs in **L** shows that the semantic ambiguity of (12a) can be accounted for within Lambek's theory. This is very fortunate, but it is contingent upon the fact that the two quantifying noun phrases are peripheral in (12a). As soon as we turn to examples in which a noun phrase semantically takes scope over a part it medially occurs in, the relevant reading is not predicted in **L**. This is explained in (Moortgat 1997) and (Hendriks 1993), who attributes the observation to Gosse Bouma. For example, there will be no **L** proof for (13) that leads to a meaning recipe which assigns *some girl* scope over *thinks*.

(13) Kazimierz thinks some girl likes Edmund

Clearly, this is a semantic variant of the periphery problem discussed in the previous section. There have been two kinds of reaction in the literature. One is exemplified by the *Flexible Montague Grammar* of (Hendriks 1988, Hendriks 1993) and offers a set of rules that can either be described as a generalization of Partee and Rooth's type shifting calculus or as a weakening of **L*P**. The rules are combined with a standard phrase structure grammar on the syntactic side. The other reaction is exemplified by (Morrill 1994), who uses a multimodal calculus to reconstruct the 'in situ binder' of (Moortgat 1988, Moortgat 1991a). The solution very cleverly uses a wrapping mode which, in combination with two other modes, an associative and a non-associative one, lets the medial quantifier 'unwrap' to a peripheral position where it can be dealt with. In the no less clever solution of (Moortgat 1995) the quantifier travels to the periphery in a step by step fashion, leaving pointers behind in the form of certain modal operators. When the quantifier is dealt with on the periphery, a 'trace' results, which can find its way back to the original position using the pointers.

All these systems are very precise, beautiful, and fun to work with. But an adoption of Hendriks' calculus, which needs a phrase structure grammar as its syntactic component, amounts to giving up the hope that there is a *single* combinatorial process driving syntax and semantics. The multimodal solutions do postulate such a single process, but it should be noted that they are much less falsifiable than the original directed and undirected approaches were. The calculus **L*P** has a permutation problem when interpreted as a syntactic calculus, while **L** is confronted with the periphery problem. We can get rid of the latter by means of allowing permutations into the calculus in a controlled way. This gets rid of the counterevidence at the price of complication. But the possibility remains that the basic combinatorics of language is insensitive to word order.

3 Lambda Grammars

3.1 Lambda Terms for Word Order

When we compare the treatment of meaning in the Lambek Calculus with its treatment of word order we are struck by the fact that the first is dealt with on the level of *terms*, the second on the level of *types*. Why this asymmetry? It seems that if we could treat word order on the level of terms as well, we might get rid of the directionality of the type system and therefore of the calculus as a whole. In fact, syntactic terms modelling linear precedence (and often also dominance) have been around in the categorial literature for a while, for various reasons. Their first appearance was in (Roorda 1991) where they played a technical role (helping to check whether proof structures are proof nets). They were drawn into the center of the system in (Moortgat 1991b), who turns the Lambek Calculus into a calculus of three-dimensional *signs* of the form in (14).

(14) $\langle$*directional type, semantic term, prosodic term*$\rangle$

The calculus now deals with sequents $\Gamma \Rightarrow \mathcal{S}$, where $\mathcal{S}$ is a sign and Γ is a sequence of signs. This follows the *Labelled Deduction* format of (Gabbay 1996) and sets up the grammar in the multidimensional form advocated in (Oehrle 1988). The set-up can be viewed as a modern version and generalization of (Saussure 1916), but also as a generalization of the 'rule-to-rule' form that we find in Montague Grammar. The format in (14) is also at the heart of (Morrill 1994), who works out the theory in great detail, giving careful treatments of many linguistic phenomena.

All this work essentially treats phrase structure terms as terms over some algebra, not as λ-terms, i.e. (Curry 1961)'s functors. This contrasts with (Oehrle 1995, Oehrle 1994), who employs (a variant of) λ-terms in the phrase structure dimension. Oehrle's signs have a form as exemplified in (15).

(15) $\lambda \mathsf{x} \lambda \mathsf{y}.\ \mathsf{y} \cdot \mathsf{questioned} \cdot \mathsf{x} : np \to (np \to s) : \lambda x \lambda y.\ question(x)(y)$

The leftmost element of this triple is what Oehrle calls a 'φ-term;' essentially a λ-term, but with a monoidal operator '$\cdot$' hard-wired into the logic and with a series of conditions in the syntactic build-up of terms ensuring that these are always linear. The second element is a (non-directional) type, and the third element is a standard semantic term. Since word order is now completely encoded in the phrase structure term, there is no longer any need for a directionality of the calculus and signs are combined using a variant of **LP**.

Since a non-directional calculus is what we are after too, we shall take Oehrle's work as our point of departure, making a few changes in the set-up, some of which are important and some unimportant. Among the unimportant changes is that we will not make any use of the dedicated 'φ-terms,' but will use ordinary λ-terms instead, using axioms for any structural requirements deemed necessary. But there are two important deviations from Oehrle's set-up. The first is that the phrase structure term associated with a sentence will *not* directly denote a string, tree, or other syntactic resource (type ν), but a set of these (type νt). This may seem a minor change, but, as will be seen in section 4 below, it will in fact enable us to essentially play the game of multimodal approaches to categorial grammar *in the phrase structure dimension.*

The second important way in which we deviate from Oehrle's set-up is that, for the basic combinatorics of the grammar, there will be *no calculus at all.* As was noticed before, a shift from the directed to the undirected calculus completely obviates the need for finding derivations. We can simply combine lexical signs using combinators. Let us see in more detail now how this can be done.

3.2 Combining Signs

Since we have decided that the basic representations in our grammar will be sequences $\langle M_1, \dots, M_n \rangle$ of λ-terms (where n is the dimensionality of the grammar), it is a good idea to develop the basics of what could be called a multi-dimensional type logic. Fortunately the multi-dimensional system we are interested in inherits most of its traits from the usual one-dimensional logic in a way that is mathematically trivial. Let us look at types first. The terms M_i in a sign $\langle M_1, \dots, M_n \rangle$ will be typed, but it will be expedient to also have *abstract types* for the n-dimensional signs themselves (by contrast, we may call the types of the M_i *concrete types*). Abstract types are formed as usual, starting with a set of basic types and letting AB be an abstract type if A and B are. For each dimension d of the grammar ($1 \leq d \leq n$)

we will have a *concretization operator* c^d sending abstract types to concrete types. The values of the c^d for basic abstract types can freely be chosen on a per grammar basis; for complex types AB we let $c^d(AB) = c^d(A)c^d(B)$. A sign $\langle M_1, \ldots, M_n \rangle$ is said to have abstract type A if each M_i is of concrete type $c^i(A)$. From now on we will only consider signs that are typed in this way.

Suppose $M = \langle M_1, \ldots, M_n \rangle$ has type AB and $N = \langle N_1, \ldots, N_n \rangle$ is of type A. Then it makes sense to define the *pointwise application* of M to N by setting

$$M(N) = \langle M_1(N_1), \ldots, M_n(N_n) \rangle \ .$$

Note that the type of the resulting sign $M(N)$ is B, as expected. It is also possible to define *pointwise abstraction*. Assuming that the variables of each concrete type come in some fixed ordering, let us call the sign $X = \langle X_1, \ldots, X_n \rangle$ the *m-th multi-dimensional variable* of type A if each of the X_i is the m-th variable of type $c^i(A)$. Let $X = \langle X_1, \ldots, X_n \rangle$ be such a variable of type A and let $M = \langle M_1, \ldots, M_n \rangle$ be a sign of type B. Then we can define

$$\lambda X.M = \langle \lambda X_1.M_1, \ldots, \lambda X_n.M_n \rangle \ ,$$

and the resulting term will be of type AB.

Pointwise application and abstraction will be used to combine elements from a *lexicon* of signs $\mathcal{L}$, the latter again given on a per grammar basis. We will assume that for each $\langle M_1, \ldots, M_n \rangle \in \mathcal{L}$, each M_i is a closed λ-term. There are no further constraints on lexical elements (in particular, linearity is not required here). The *linear combinations from* $\mathcal{L}$ are defined as follows.

i. Each $M \in \mathcal{L}$ is a [linear] combination from $\mathcal{L}$ and $\mathsf{fv}(M) = \varnothing$;

ii. If X is a multi-dimensional variable of type A, X is a [linear] combination from $\mathcal{L}$ and $\mathsf{fv}(X) = \{X\}$;

iii. If M is a [linear] combination from $\mathcal{L}$ of type AB and N is a [linear] combination from $\mathcal{L}$ of type A [such that $\mathsf{fv}(M) \cap \mathsf{fv}(N) = \varnothing$], then $M(N)$ is a [linear] combination from $\mathcal{L}$ and $\mathsf{fv}(M(N)) = \mathsf{fv}(M) \cup \mathsf{fv}(N)$;

iv. If M is a [linear] combination from $\mathcal{L}$ of type B, X is a multi-dimensional variable of type A [and $X \in \mathsf{fv}(M)$], then $\lambda X.M$ is a [linear] combination from $\mathcal{L}$ of type AB and $\mathsf{fv}(\lambda X.M) = \mathsf{fv}(M) - \{X\}$.

For a definiton of the *combinations from* $\mathcal{L}$ just skip anything between square brackets in the definition above. Suppose that $M = \langle M_1, \ldots, M_n \rangle$ and

$N = \langle N_1, \ldots, N_n \rangle$ are signs and that N and the variable $X = \langle X_1, \ldots, X_n \rangle$ have the same type. Then it makes sense to define

$$[N/X]M = \langle [N_1/X_1]M_1, \ldots, [N_n/X_n]M_n \rangle \ .$$

In this case N is said to be *free for* X in M if each N_i is free for X_i in M_i. For combinations from some lexicon $\mathcal{L}$ we have

(α) $\lambda X.M = \lambda Y.[Y/X]M$ if Y is free for X in M;

(β) $(\lambda X.M)(N) = [N/X]M$ if N is free for X in M;

(η) $\lambda X.M(X) = M$ if $X \notin \mathsf{fv}(M)$.

We are interested in linear combinations M from $\mathcal{L}$ that are *closed*, i.e. such that $\mathsf{fv}(M) = \varnothing$. These are said to be the signs *generated* from $\mathcal{L}$.

It is time for an example. In the following we will consider a two-dimensional grammar (i.e. $n = 2$) with dimensions for phrase structure ($d = 1$) and semantics ($d = 2$). There will be three basic abstract types, $\textsc{np}$, $\textsc{n}$ and $\textsc{s}$. This means that $\textsc{np}(\textsc{np}\ \textsc{s})$ and $(\textsc{np}\ \textsc{s})\textsc{s}$ are examples of complex abstract types. The concretization operators c^1 and c^2 send basic abstract types to concrete types as suggested in Table 1 below (for the moment the feature dimension considered there may be ignored) and as a result of this we have e.g. that $c^1((\textsc{np}\ \textsc{s})\textsc{s}) = ((\nu t)(\nu t))(\nu t)$ and $c^2((\textsc{np}\ \textsc{s})\textsc{s}) = (e(st))(st)$. (16) gives a mini-lexicon; three signs, of types $(\textsc{np}\ \textsc{s})\textsc{s}$, $(\textsc{np}\ \textsc{s})\textsc{s}$, and $\textsc{np}(\textsc{np}\ \textsc{s})$ respectively.

(16) a. $\langle \lambda T.T(\mathsf{every} \bullet \mathsf{man}), \lambda P \lambda i \forall x[man(x,i) \to P(x)(i)] \rangle$

b. $\langle \lambda T.T(\mathsf{a} \bullet \mathsf{woman}), \lambda P \lambda i \exists x[woman(x,i) \land P(x)(i)] \rangle$

c. $\langle \lambda t_1 \lambda t_2.(t_2 \bullet (\mathsf{loves} \bullet t_1)), \lambda x \lambda y \lambda i.love(y,x,i) \rangle$

That this typing is correct can be checked given the typographical conventions for variables that are spelled out in Table 2 and given the following information about the constants that are employed. $\bullet$ is of type $(\nu t)((\nu t)\nu t)$ and is always written between its νt arguments. The constants every, man, a, etc. denote sets of resources (intuitively every denotes the set of resources labelled *every*, etc.) and are therefore of type νt. In the semantic terms, the constants *man* and *woman* are of type $(e \times s)t$ and *love* is of type $(e \times e \times s)t$. As a consequence, the terms in (16a) are therefore of types $((\nu t)(\nu t))(\nu t)$ and $(e(st))(st)$ respectively, as are those in (16b), while the terms in (16c) are of types $(\nu t)((\nu t)\nu t)$ and $e(e(st))$.

In (17) four signs generated from this grammar are displayed. Here ζ and ζ' are two-dimensional variables of type NP.

(17) a. $(16b)\Big(\lambda\zeta.[(16a)((16c)(\zeta))]\Big)$

b. $(16a)\Big(\lambda\zeta'.\Big[(16b)\Big(\lambda\zeta.[(16c)(\zeta)(\zeta')]\Big)\Big]\Big)$

c. $(16a)\Big(\lambda\zeta.[(16b)((16c)(\zeta))]\Big)$

d. $(16b)\Big(\lambda\zeta'.\Big[(16a)\Big(\lambda\zeta.[(16c)(\zeta)(\zeta')]\Big)\Big]\Big)$

Let us work out (17a). Writing $(16b)_1$ for the first element of (16b), $(16b)_2$ for its second element, and similar for other signs, it is easily seen that (17a) can be rewritten as (18). A series of λ-conversions in both dimensions of this sign then takes us to (19a). The other items in (19) are obtained from their counterparts in (17) in an entirely similar way.

(18) $\langle (16b)_1(\lambda t.(16a)_1((16c)_1(t))), (16b)_2(\lambda x.(16a)_2((16c)_2(x))\rangle$

(19) a. $\langle(((\mathsf{every} \bullet \mathsf{man}) \bullet (\mathsf{loves} \bullet (\mathsf{a} \bullet \mathsf{woman}))),$
$\lambda i\exists y[\mathit{woman}(y,i) \wedge \forall x[\mathit{man}(x,i) \rightarrow \mathit{love}(x,y,i)]]\rangle$

b. $\langle(((\mathsf{every} \bullet \mathsf{man}) \bullet (\mathsf{loves} \bullet (\mathsf{a} \bullet \mathsf{woman}))),$
$\lambda i\forall x[\mathit{man}(x,i) \rightarrow \exists y[\mathit{woman}(y,i) \wedge \mathit{love}(x,y,i)]]\rangle$

c. $\langle(((\mathsf{a} \bullet \mathsf{woman}) \bullet (\mathsf{loves} \bullet (\mathsf{every} \bullet \mathsf{man}))),$
$\lambda i\forall x[\mathit{man}(x,i) \rightarrow \exists y[\mathit{woman}(y,i) \wedge \mathit{love}(y,x,i)]]\rangle$

d. $\langle(((\mathsf{a} \bullet \mathsf{woman}) \bullet (\mathsf{loves} \bullet (\mathsf{every} \bullet \mathsf{man}))),$
$\lambda i\exists y[\mathit{woman}(y,i) \wedge \forall x[\mathit{man}(x,i) \rightarrow \mathit{love}(y,x,i)]]\rangle$

Our λ-grammar gives us a mechanism to couple syntactic representations and semantic representations in a way that seems essentially correct. How easy is it to get incorrect couplings? Is it possible, for example, to get the patently incorrect $\langle (19a)_1, (19d)_2\rangle$? Given that the generating mechanism seems rather free, this might seem a reasonable worry. But a moment's reflection shows that such worries are groundless. In fact it is well-known (see e.g. (van Benthem 1991, pp. 117–119)) that, up to $\beta\eta$-equivalence, there are exactly four linear combinations of two quantifiers with one binary relation such that quantifiers and relation symbol each occur exactly once.

This means that (17) and therefore (19) sum up all those possibilities. The availability of syntactic λ-terms reins in the overgeneration of the traditional undirected calculi.

This shows that the permutation problem discussed before has now been overcome. What about the perifery problem? Is it possible to extract material from non-peripheral positions? The following extension of the previous example can be used to show that it is. (Here $M\colon A$ means that M is of abstract type A.)

(20) a. $\langle \mathsf{book}, \lambda x \lambda i.\mathit{book}(x, i)\rangle\colon \textsc{n}$

b. $\langle \mathsf{Bill}, b\rangle\colon \textsc{np}$

c. $\langle \mathsf{Sue}, s\rangle\colon \textsc{np}$

d. $\langle \lambda t_1 \lambda t_2 \lambda t_3.(t_3 \bullet ((\mathsf{gives} \bullet t_1) \bullet (\mathsf{to} \bullet t_2))),$
$\lambda x \lambda y \lambda z \lambda i.\mathit{give}(z, x, y, i)\rangle\colon \textsc{np}(\textsc{np}(\textsc{np}\ \textsc{s}))$

e. $\langle \lambda T \lambda t.(t \bullet (\mathsf{that} \bullet T(\mathsf{e}))),$
$\lambda P_1 \lambda P_2 \lambda x \lambda i.[P_2(x)(i) \wedge P_1(x)(i)]\rangle\colon (\textsc{np}\ \textsc{s})(\textsc{n}\ \textsc{n})$

The first element of (20e) here is a function taking functions from noun phrases to sentences as its first argument. Note that this argument is applied to a trace e (for more on traces see the next section). The net effect will be substitution of this trace for the 'missing' noun phrase. An illustration is given in (21), where we see that the generated sign (21a) reduces to (21b). Essentially, since it is possible to abstract from any position it is also possible to extract from any position.

(21) a. $(20\text{e})\Big(\lambda\zeta.[(20\text{d})(\zeta)((20\text{b}))((20\text{c}))]\Big)\Big((20\text{a})\Big)$

b. $\langle(\mathsf{book} \bullet (\mathsf{that} \bullet (\mathsf{Sue} \bullet ((\mathsf{gives} \bullet \mathsf{e}) \bullet (\mathsf{to} \bullet \mathsf{Bill}))))),$
$\lambda x \lambda i.[\mathit{book}(x, i) \wedge \mathit{give}(s, x, b, i)]\rangle$

The combinations defined above are closely related to the *combinators* of (Curry and Feys 1958), which in a general context can be taken to be closed pure linear λ-terms. Indeed, in our n-dimensional setting we can define combinators as combinations from the empty lexicon $\varnothing$ and linear combinators as linear combinations from $\varnothing$. In view of the fact that β-conversion holds it is easily seen that each generated sign M can be rewritten as an n-dimensional term $C(L^1)\cdots(L^m)$, where C is a linear combinator and $L^1, \ldots, L^m \in \mathcal{L}$.

Generated signs can be obtained from lexical elements with the help of linear combinators.

This clearly brings us close to Combinatory Categorial Grammar (CCG, see e.g. (Ades and Steedman 1982, Szabolsci 1989, Steedman 1996, Jacobson 1999)) where all possible combinators are investigated for their linguistic relevance. But CCG, like the Lambek Calculus, depends on directionality and derivations and we want to do away with those.

The definitions given thus far define the notion of Lambda Grammars that was also present in (Muskens 2001a, Muskens 2001b), but there is an alternative perspective on the formalism that arises from independent work by Philippe de Groote on *Abstract Categorial Grammars* ((de Groote 2001, de Groote 2002)). For a precise definition of Abstract Categorial Grammars, or ACGs, the reader is referred to the works mentioned, but the main idea is as follows. ACGs consist of an *abstract vocabulary* and a *concrete vocabulary.* The former is essentially a collection of constants that are assigned abstract types, as in (22a-c). Over such abstract vocabularies linear λ-terms are considered, such as (22d), which should be compared with (17a).

(22) a. EVERY-MAN: (NP S)S

b. A-WOMAN: (NP S)S

c. LOVES: NP(NP S)

d. $\text{A-WOMAN}\Big(\lambda\zeta.[\text{EVERY-MAN}(\text{LOVES}(\zeta))]\Big)$

A concrete vocabulary, on the other hand, is a collection of constants with concrete types. A type homomorphism sends abstract types to concrete types (this type homomorphism is what we have called a concretization operator) and there is also a λ-term homomorphism sending linear λ-terms over the abstract vocabulary to λ-terms[4] over the concrete vocabulary. The function F_1 in (23) is an example of such a λ-term homomorphism. Given (23a-c), (23d) must be the case.

(23) a. $F_1(\text{EVERY-MAN}) = \lambda T.T(\mathsf{every} \bullet \mathsf{man})$

b. $F_1(\text{A-WOMAN}) = \lambda T.T(\mathsf{a} \bullet \mathsf{woman})$

[4]In fact de Groote requires the values of this λ-term homomorphism to also be *linear* λ-terms. This means that e.g. $\lambda P\lambda i\forall x[man(x,i) \to P(x)(i)]$ is not strictly a legitimate value for F_2 in (24) below. I have suppressed this issue from the main text for ease of exposition.

c. $F_1(\textsc{loves}) = \lambda t_1 \lambda t_2.(t_2 \bullet (\mathsf{loves} \bullet t_1))$

d. $F_1\Big(\textsc{a-woman}\Big(\lambda\zeta.[\textsc{every-man}(\textsc{loves}(\zeta))]\Big)\Big) =$
$((\mathsf{every} \bullet \mathsf{man}) \bullet (\mathsf{loves} \bullet (\mathsf{a} \bullet \mathsf{woman})))$

This sketches the set-up of one Abstract Categorial Grammar, but we could have a second one with the same abstract vocabulary but with its concrete vocabulary now drawn from the constants we have used in the semantic component. A second λ-term homomorphism F_2 could be defined by (24a-c), so that (24d) would become true.

(24) a. $F_2(\textsc{every-man}) = \lambda P \lambda i \forall x[\mathit{man}(x, i) \to P(x)(i)]$

b. $F_2(\textsc{a-woman}) = \lambda P \lambda i \exists x[\mathit{woman}(x, i) \wedge P(x)(i)]$

c. $F_2(\textsc{loves}) = \lambda x \lambda y \lambda i.\mathit{love}(y, x, i)$

d. $F_2\Big(\textsc{a-woman}\Big(\lambda\zeta.[\textsc{every-man}(\textsc{loves}(\zeta))]\Big)\Big) =$
$\lambda i \exists y[\mathit{woman}(y, i) \wedge \forall x[\mathit{man}(x, i) \to \mathit{love}(x, y, i)]]$

These two abstract categorial grammars do what our lambda grammar with lexicon (16) does, for clearly $M = \langle M_1, M_2 \rangle$ is generated from that lexicon if and only if there is a linear term N over the abstract vocabulary in (22) such that $F_1(N) = M_1$ and $F_2(N) = M_2$. More in general, any n-dimensional lambda grammar can be interpreted by a collection of n abstract categorial grammars with the same abstract vocabulary.[5]

4 More Structure

In the previous section the main ideas behind Lambda Grammars were explained in some generality. Here we consider one possible way to make these general ideas more concrete. In particular, we will look at an implementation with three dimensions, one for phrase structure, one for semantics, and one for feature structures. Each of these dimensions will have its own logic.

[5]These statements hold modulo a minor adjustment of definitions. Either ACGs must allow the images of constants from the abstract vocabulary under the λ-term homomorphism to be non-linear, or Lambda Grammars must disallow non-linearity in the λ-terms that lexical signs consist of (see the previous note). The possibility of having non-linear terms in lexical elements seems essential given linguistic phenomena such as reflexives or anaphoric relations. A theory such as ours that treats all combination as linear must treat such phenomena, that clearly involve the identification of argument places, as lexical. Non-linearity also seems essential for semantic terms. See also the discussion in (de Groote 2001).

4.1 Phrase structure

For the logic of the phrase structure dimension we will borrow heavily from fairly recent 'multi-modal' approaches to categorial grammar such as (Morrill 1994, Moortgat 1997). The main difference with these approaches will be that we will situate the logic entirely within the phrase structure dimension.

The basic idea behind the multi-modal approach is that syntax deals with 'resources' (think of these as tree nodes or string positions, but the general notion is more abstract) that can be combined in various 'modes'. The usual operation of taking two constituents and providing them with a mother might be such a mode. Let us call it mode c, and let us associate with it a ternary relation R^c such that $R^c(k, k_1, k_2)$ is meant to express that k is the mother of k_1 and k_2, while k_1 precedes k_2. This ternary relation can now be interpreted as an *accessibility* relation and can be used to define a product $\bullet$, by writing $A \bullet B$ for $\lambda k.\exists k_1 k_2[R^c(k, k_1, k_2) \wedge A(k_1) \wedge B(k_2)]$. Readers familiar with modal logic will recognize this as a generalization of the usual interpretation of the operator $\Diamond$. The term $(\mathsf{john} \bullet (\mathsf{sees} \bullet \mathsf{mary}))$ is now shorthand for

$$\lambda k \exists k_1 k_2[R^c(k, k_1, k_2) \wedge \mathsf{john}(k_1) \wedge \exists k_3 k_4[R^c(k_2, k_3, k_4) \wedge \\ \mathsf{sees}(k_3) \wedge \mathsf{mary}(k_4)]]$$

and many of the expressions used in section 3 can now also be taken to abbreviate terms built up with the help of R^c.

More in general, whenever we want to assume that a certain mode of combination m is present in the syntactic component of the language, combining two resources into a third, we can model this by introducing a ternary relation R^m (type $(\nu \times \nu \times \nu)t$) and axiomatizing its basic properties, together with its relations to other modes of combination. Sets of resources can then be combined into other sets of resources using the following binary modality.

(25) $\lambda t_1 t_2 \lambda k.\exists k_1 k_2[R^m(k, k_1, k_2) \wedge t_1(k_1) \wedge t_2(k_2)]$

This modality will normally be written as an operator $\bullet_m$, in infix notation. The reader will recognize the standard semantics for products in the multimodal Lambek Calculus.

We must provide the relation R^c considered above with some axioms, for while terms like $(\mathsf{john} \bullet (\mathsf{sees} \bullet \mathsf{mary}))$ very much look like the usual tree structures, we have not in fact connected the relation R^c with any notion of linear precedence. There is also no connection between nodes k that the term

$(\mathsf{john} \bullet (\mathsf{sees} \bullet \mathsf{mary}))$ can be predicated of and the string *john sees mary.* As it is the grammarian's business to connect strings with their possible semantic values, there is a gap to be filled here. We will consider two ways in which this might be done.

The first way provides the ν domain with binary relations $\lhd^+$ and $\prec$, which stand for proper dominance and precedence and impose the necessary structure on these by means of axioms (see also (Cornell 1994, Backofen, Rogers and Vijay-Shankar 1995, Muskens 2001c)). Here we just adopt the requirements in (26). The two relations are strict partial orders ((26a)), related by *Inheritance* ((26b,c)), and there is an immediate dominance relation $\lhd$, defined in terms of $\lhd^+$ ((26d)).

(26) a. $\lhd^+$ and $\prec$ are irreflexive and transitive

b. $\forall k_1 k_2 k_3 \, [[k_1 \lhd^+ k_2 \wedge k_1 \prec k_3] \rightarrow k_2 \prec k_3]$

c. $\forall k_1 k_2 k_3 \, [[k_1 \lhd^+ k_2 \wedge k_3 \prec k_1] \rightarrow k_3 \prec k_2]$

d. $\forall k_1 k_2 \, [k_1 \lhd k_2 \leftrightarrow \forall k_3 \, [k_1 \lhd^+ k_3 \lhd^+ k_2 \rightarrow [k_3 = k_1 \vee k_3 = k_2]]]$

These requirements in themselves do not suffice to axiomatize the notion of linguistic tree. For instance, the usual requirement of *Rootedness* fails, as does the requirement of *Exhaustivity* in (27).

(27) $\forall k_1 k_2 \, [k_1 \prec k_2 \vee k_2 \prec k_1 \vee k_1 \lhd^+ k_2 \vee k_2 \lhd^+ k_1 \vee k_1 = k_2]$

But the axioms are sufficient for our purposes if we explicate R^c as the usual mother-daughters relationship by the following definition.

(28) $\forall k k_1 k_2 \, [R^c(k, k_1, k_2) \leftrightarrow k \lhd k_1 \wedge k \lhd k_2 \wedge k_1 \prec k_2]$

Suppose we have some νt term, t which is built up from νt terms like every, man, a, etc. with the help of $\bullet$. Then $t(k)$ holds if and only if k is the top node of the obvious tree. For example, $(\mathsf{john} \bullet (\mathsf{sees} \bullet \mathsf{mary}))(k)$ is true iff k is the root of the linguistic tree for *John sees Mary.* In general, Exhaustivity will hold for the nodes dominated by k, as the reader can easily verify.

While this first method connects our type νt terms with sets of linguistic trees (without category labels, but see the section on feature information below), the second method, which will be adopted here, more directly connects them with sets of *strings*. In fact, we can take all resources to be strings in the second approach, although R^c gives more structure than just string concatenation.

Let us axiomatize string concatenation as in (29), where the $\nu(\nu\nu)$ function '$\cdot$' is stipulated to be monoidal (with 1 a constant of type ν).

(29) a. $\forall k_1 k_2 k_3\ (k_1 \cdot k_2) \cdot k_3 = k_1 \cdot (k_2 \cdot k_3)$

b. $\forall k\ k \cdot 1 = k = 1 \cdot k$

We introduce an associative modality $\bullet_0$, written $\circ$, by adopting (30a) as a definition for the underlying relation R^0 and we connect the new modality with the old one by means of (30b).

(30) a. $\forall k k_2 k_3\, [R^0(k, k_1, k_2) \leftrightarrow k = k_1 \cdot k_2]$

b. $\forall k k_2 k_3\, [R^c(k, k_1, k_2) \to R^0(k, k_1, k_2)]$

These stipulations ensure, for example, that if $(\mathsf{john} \bullet (\mathsf{sees} \bullet \mathsf{mary}))(k)$ holds, there are k_1, k_2, and k_3, such that $k = k_1 \cdot k_2 \cdot k_3$, $\mathsf{john}(k_1)$, $\mathsf{sees}(k_2)$, and $\mathsf{mary}(k_3)$. Moreover, we have $\mathsf{john} \circ \mathsf{sees} \circ \mathsf{mary}(k)$. But, clearly, $((\mathsf{john} \bullet \mathsf{sees}) \bullet \mathsf{mary})(k)$ need not hold and the $\bullet$ modality provides a more fine-grained structuring of the relevant domain than $\circ$ does.

The trace e that we had occasion to use in the previous section can be defined in terms of the monoidal unity 1.

(31) $\mathsf{e} = \lambda k.k = 1$

Essentially, this will set e to the singleton $\{1\}$ and it will hold that $A \circ \mathsf{e} = A = \mathsf{e} \circ A$ for any A.

The reader familiar with multimodal categorial grammar has recognized (30b) as the 'frame condition' corresponding to an *inclusion postulate* connecting the $\bullet$ and $\circ$ modalities. Let us also formulate this inclusion postulate. Writing $A \sqsubseteq B$ for $\forall k\, [A(k) \to B(k)]$ if A and B are νt terms, (32) readily follows from (25) and the assumption in (30b).

(32) $A \bullet B \sqsubseteq A \circ B$

The fact that $(\mathsf{john} \bullet (\mathsf{sees} \bullet \mathsf{mary})) \sqsubseteq \mathsf{john} \circ \mathsf{sees} \circ \mathsf{mary}$ can now also be established directly on the level of the modalities, with the help of (32) and the second of the following two monotonicity rules. These rules obviously hold for all modalities $\bullet_m$ formed with (25).

(33)
$$\frac{A \sqsubseteq A'}{A \bullet_m B \sqsubseteq A' \bullet_m B} \uparrow mon \qquad \frac{B \sqsubseteq B'}{A \bullet_m B \sqsubseteq A \bullet_m B'} mon \uparrow$$

Other postulates can also be obtained by assumption of the relevant frame conditions and the multimodal game can be played. By way of example we shall introduce two more modalities and give interaction postulates. The first of these is written as $\bullet_w$ and models the *right wrap* that was discussed in (Bach 1979, Bach 1984) and has been kicking around in the categorial literature ever since. (34a) is the condition governing its interaction with $\bullet$ and (34b) is the resulting interaction principle derivable from (34a).

(34) a. $\forall k_1 k_2 k_3 k_4[\exists k[R^w(k_1, k_2, k) \wedge R^c(k, k_3, k_4)] \leftrightarrow [\exists k[R^c(k_1, k_3, k) \wedge R^c(k, k_2, k_3)]]$

b. $A \bullet_w (B \bullet C) = B \bullet (A \bullet C)$

The wrapping modality allows insertion of an element after the first element of a structure.

The last modality we want to define provides an inverse to wrapping and will be called *unwrapping*. It will be a unary modality, based on a binary underlying accessibility relation. The general way to define unary modalities $\Diamond_m$ from binary relations R^m, familiar from standard modal logic, is that in (35).

(35) $\lambda t \lambda k.\exists k'[R^m(k, k') \wedge t(k')]$

Our unwrapping modality $\Diamond_u$ will be based on the interaction constraint in (36a), from which (36b) is derivable. It essentially allows us to 'raise' the second element of a structure to a frontal position (note that $\Diamond_u(A \bullet (B \bullet C)) = B \bullet (A \bullet C)$ via (34b)).

(36) a. $\forall k_1 k_2 k_3[\exists k[R^u(k_1, k) \wedge R^w(k, k_2, k_3)] \leftrightarrow R^\bullet(k_1, k_2, k_3)]$

b. $\Diamond_u(A \bullet_w B) = A \bullet B$

The discussion makes it clear that the essentialities of the multimodal enterprise can be regained in the present set-up. But the multimodal game is best played in the phrase structure dimension, and should, I think, be factored out of the general combinatorics of language. With postulates such as (32), (34b), (36b), and the monotonicity rules in (33) we do get a calculus, but the calculus is simple. Even with more inclusion postulates and interaction postulates added, it will be much simpler than the usual multimodal calculi.

In a set-up where all elements of type ν can be taken to be strings, there is a special interest in νt terms that are built up from lexical νt terms with the help of $\circ$ alone. Let us call the latter $\circ$-*terms.* Suppose $\langle S_1, S_2, \ldots, S_n \rangle$ is a generated sign with phrase structure component S_1 and that $S_1 \sqsubseteq S_1'$

is derivable given some fixed set of axioms. Then we call $\langle S'_1, S_2, \ldots, S_n\rangle$ a *derivable* sign. In general, if a sign is generated, then any resource that belongs to the extension of the phrase structure component can have the semantics component as its associated meaning. In a set-up in which all resources are strings, the derivable signs with a $\circ$-term as phrase structure element provide a direct connection between string and meaning.

4.2 Semantics

We will take a straightforward possible worlds approach in the semantics dimension, essentially that of (Montague 1973), but streamlined in the way of (Muskens 1995). The function sending categories to types in PTQ is not strictly in agreement with our previous requirement that $c^2(AB) = c^2(A)c^2(B)$, and this brings with it certain complications (but see (Hendriks 1993)). However, (Muskens 1995) shows that a slight reformulation of the theory, which does conform to such a requirement, is in fact equivalent. The reformulation uses a many-sorted classical type logic (compare (Gallin 1975)), not Montague's **IL**.

It should be emphasized again that the constraint that $c^2(AB) = c^2(A)c^2(B)$ leaves considerable freedom in the way the semantic dimension is implemented. For example, a category-to-type function c^2 that sends NP to the type π of discourse referents and S to $s(st)$, in the way of (Muskens 1996), can be used to combine categorial grammar with Discourse Representation Theory (Kamp and Reyle 1993). See (Muskens 1994) for an elaboration of this idea in the context of the Lambek Calculus. Many other variations are possible.

4.3 Features

It is useful to have features as an extra dimension in our signs. We consider the first-order axiomatisation of features given in (Johnson 1991). A *feature structure* consists of a collection of feature nodes connected by labeled transitions, as in (37). Feature nodes will be assigned type φ; attributes labelling transitions will have type α.

(37)
$$\begin{tikzcd} & \bullet \arrow[r, "\text{agr}"'] & \bullet\ \text{3sg} \\ \bullet \arrow[ur, bend left, "\text{subj}"'] \arrow[dr, bend right, "\text{vform}"] & & \\ & \bullet\ \text{pres} & \end{tikzcd}$$

(38) $\lambda f \exists f'[arc(f, subj, f') \wedge arc(f', agr, 3sg) \wedge arc(f, vform, pres)]$

That feature nodes are connected can be expressed using the three-place relation symbol *arc* of type $(\varphi \times \alpha \times \varphi)t$, with $arc(f_1, a, f_2)$ saying that f_1 and f_2 are connected by an arc labeled a. This is illustrated in (38), a φt term satisfied by leftmost node of (37). Here constants such as *agr, subj, vform, cat,*... are of type α and denote attributes, while constants such as *3sg*, $-$*3sg, past, pres, V, N*, $+$, $-$, ... are of type φ. We typically use them to denote graph nodes that have no successors and stand for atomic feature values. The set of constants of type φ is called C_{val}.

The following three axioms are a direct adaptation from (Johnson 1991). The first puts a functionality requirement on the transition relation. The second embodies the constraint that atomic features have no further attributes. And the third axiom schema gives constant-constant clashes by requiring that *past* $\neq$ *3sg*, *3sg* $\neq -$*3sg*, *past* $\neq$ *pres*, $V \neq N$, etc.

(39) a. $\forall a \forall f_1 f_2 f_3[[arc(f_1, a, f_2) \wedge arc(f_1, a, f_3)] \rightarrow f_2 = f_3]$

b. $\forall a \forall f \neg\, arc(c, a, f)$, where $c \in C_{val}$

c. $c \neq c'$, for all syntactically distinct pairs $c, c' \in C_{val}$

The first and third of these axioms work together to obtain the effect of unification, with intersection doing all the work. For example, $\lambda f.(40a)(f) \wedge (40b)(f)$, the intersection of (40a) and (40b), will be equivalent with (38), given (39a).

(40) a. $\lambda f \exists f'[arc(f, subj, f') \wedge arc(f, vform, pres)]$

b. $\lambda f \exists f'[arc(f, subj, f') \wedge arc(f', agr, 3sg)]$

c. $\lambda f \exists f'[arc(f, subj, f') \wedge arc(f', agr, -3sg)]$

The intersection of (38) and (40c), on the other hand, will not be satisfiable. (39a) and (39c) ensure that it will denote the empty set in all models.

Defining the feature logic in this way will allow us to set up the feature dimension of our signs much in the same way as the signifier and signified dimensions were set up. We can consider lambda expressions over the feature vocabulary and combine them using application. Combinators will now also have concretizations in the feature dimension. Examples of how this works can be found in the toy grammar below.

4.4 A Toy Grammar

In order to illustrate some of the possibilities of Lambda Grammars we give a toy grammar for a fragment of English. The grammar is based on the abstract types S, NP and N. As before, concretizations of these types in each dimension can be found in Table 1.

Defining a grammar essentially amounts to providing a lexicon. Let us start with giving lexical items for some verbs. (41) lists entries for base forms of the verbs *laugh, kiss* and *think* and additionally gives two inflected forms of *kiss*. The entry for *laugh* takes a subject in order to form a sentence and therefore is assigned the abstract type (NP S). The forms of *kiss* have type (NP (NP S)) and *think* has type (S (NP S)). (We again refer the reader to Table 2 for some of the typographical conventions used in the lexical items of our toy grammar.)

(41) a. $\langle \lambda t.(t \bullet \textsf{laugh}),$
$\mathit{laugh}_{e(st)},$
$\lambda F \lambda f.\exists f_1[F(f_1) \wedge \mathit{arc}(f_1, \mathit{cat}, N) \wedge \mathit{arc}(f, \mathit{subj}, f_1) \wedge$
$\mathit{arc}(f, \mathit{vform}, \mathit{base})]\rangle$

b. $\langle \lambda t_1 t_2.(t_2 \bullet (\textsf{kiss} \bullet t_1)),$
$\mathit{kiss}_{e(e(st))},$
$\lambda F_1 F_2 \lambda f.\exists f_1 f_2[F_1(f_1) \wedge F_2(f_2) \wedge \mathit{arc}(f, \mathit{vform}, \mathit{base}) \wedge$
$\mathit{arc}(f_1, \mathit{cat}, N) \wedge \mathit{arc}(f, \mathit{obj}, f_1) \wedge \mathit{arc}(f_2, \mathit{cat}, N) \wedge \mathit{arc}(f, \mathit{subj}, f_2)]\rangle$

c. $\langle \lambda t_1 t_2.(t_2 \bullet (\textsf{think} \bullet t_1)),$
$\mathit{believe}_{(st)(e(st))},$
$\lambda F_1 F_2 \lambda f.\exists f_1 f_2[F_1(f_1) \wedge F_2(f_2) \wedge \mathit{arc}(f, \mathit{vform}, \mathit{base}) \wedge$
$\mathit{arc}(f_1, \mathit{cat}, S) \wedge \mathit{arc}(f, \mathit{comp}, f_1) \wedge \mathit{arc}(f_2, \mathit{cat}, N) \wedge \mathit{arc}(f, \mathit{subj}, f_2)]\rangle$

d. $\langle \lambda t_1 t_2.(t_2 \bullet (\textsf{kisses} \bullet t_1)),$
$\mathit{kiss},$
$\lambda F_1 F_2 \lambda f.\exists f_1 f_2[F_1(f_1) \wedge F_2(f_2) \wedge \mathit{arc}(f, \mathit{vform}, \mathit{pres}) \wedge$
$\mathit{arc}(f_1, \mathit{cat}, N) \wedge \mathit{arc}(f, \mathit{obj}, f_1) \wedge \mathit{arc}(f_2, \mathit{cat}, N) \wedge \mathit{arc}(f, \mathit{subj}, f_2) \wedge$
$\mathit{arc}(f_2, \mathit{agr}, \mathit{3sg})]\rangle$

e. $\langle \lambda t_1 t_2.(t_2 \bullet (\textsf{kissed} \bullet t_1)),$
$\mathit{kiss},$
$\lambda F_1 F_2 \lambda f.\exists f_1 f_2[F_1(f_1) \wedge F_2(f_2) \wedge \mathit{arc}(f, \mathit{vform}, \mathit{past}) \wedge$
$\mathit{arc}(f_1, \mathit{cat}, N) \wedge \mathit{arc}(f, \mathit{obj}, f_1) \wedge \mathit{arc}(f_2, \mathit{cat}, N) \wedge \mathit{arc}(f, \mathit{subj}, f_2)]\rangle$

abstract type	phrase structure ($d = 1$)	semantics ($d = 2$)	features ($d = 3$)
S	νt	st	φt
NP	νt	e	φt
N	νt	$e(st)$	φt

Table 1: Concretizations of abstract types.

The signs in (41) can be combined with arguments such as the simple NP in (42). This will lead to *untensed* sentences such as (43),[6] which was obtained by pointwise application of (41a) to (42).

(42) $\langle \mathsf{Mary}, \mathit{mary}, \lambda f.\mathit{arc}(f, \mathit{cat}, N) \wedge \mathit{arc}(f, \mathit{agr}, \mathit{3sg})\rangle$

(43) $\langle(\mathsf{Mary} \bullet \mathsf{laugh}),$
$\mathit{laugh}(\mathit{mary}),$
$\lambda f.\exists f_1[\mathit{arc}(f_1, \mathit{cat}, N) \wedge \mathit{arc}(f_1, \mathit{agr}, \mathit{3sg}) \wedge \mathit{arc}(f, \mathit{subj}, f_1) \wedge$
$\mathit{arc}(f, \mathit{vform}, \mathit{base})]\rangle$

Untensed forms have their use; (43), for example, could perhaps be used to build *John heard Mary laugh*, but it is primarily tensed forms that we are after. These can be obtained by combining an untensed sentence with one of the (S S) items in (44).[7] Among the items in (44) are some auxiliaries (the first three items), but also the simple past and simple present. At first blush it may seem that in forms such as (43) a place for an auxiliary is no longer available, but here the multimodal set-up of the framework can be used to make subjects 'raise' to their intended positions. We use the wrapping modality to enforce raising of the subject over the auxiliary. An illustration will follow shortly.

(44) a. $\langle \lambda t.\mathsf{will} \bullet_w t,$
$\lambda p \lambda i \exists j[i < j \wedge j \approx i \wedge p(j)],$
$\lambda F \lambda f \exists f_1[F(f_1) \wedge \mathit{arc}(f_1, \mathit{vform}, \mathit{base}) \wedge \mathit{arc}(f, \mathit{cat}, S)]\rangle$

[6]That (43) is untensed is here represented by the fact that it gets category V, not S. The feature descriptions in (41) have not explicitly mentioned the category V, but we adopt the following redundancy postulate for convenience: $\forall f[\exists f_1\, \mathit{arc}(f, \mathit{vform}, f_1) \leftrightarrow \mathit{arc}(f, \mathit{cat}, V)]$.

[7]The notation $i < j$ in (44a) and (44b) stands for 'the time component of i precedes the time component of j' while $i \approx j$ should be interpreted as 'i and j have the same world components.' Here the objects of type s are interpreted as world-time pairs. An axiomatization of $<$ and $\approx$ can be found in (Muskens 1995).

phrase structure	semantics	features
$t\colon \nu t$	$x, y, z\colon e$	$f\colon \varphi$
$T\colon (\nu t)(\nu t)$	$i, j\colon s$	$F\colon \varphi t$
	$p\colon st$	$\mathcal{F}\colon (\varphi t)(\varphi t)$
	$P\colon e(st)$	

Table 2: Typographical conventions used here. *Var*: *Type* means that *Var* (with or without subscripts or superscripts) always has type *Type*.

b. $\langle \lambda t.\mathsf{didn't} \bullet_w t,$
$\lambda p \lambda i \neg \exists j[j < i \wedge j \approx i \wedge p(j)],$
$\lambda F \lambda f \exists f_1[F(f_1) \wedge \mathit{arc}(f_1, \mathit{vform}, \mathit{base}) \wedge \mathit{arc}(f, \mathit{cat}, S)]\rangle$

c. $\langle \lambda t.\mathsf{doesn't} \bullet_w t,$
$\lambda p \lambda i \neg p(i),$
$\lambda F \lambda f \exists f_1 f_2[F(f_1) \wedge \mathit{arc}(f_1, \mathit{vform}, \mathit{base}) \wedge \mathit{arc}(f_1, \mathit{subj}, f_2) \wedge$
$\mathit{arc}(f_2, \mathit{agr}, \mathit{3sg}) \wedge \mathit{arc}(f, \mathit{cat}, S)]\rangle$

d. $\langle \lambda t.t,$
$\lambda p \lambda i \exists j[j < i \wedge j \approx i \wedge p(j)],$
$\lambda F \lambda f \exists f_1[F(f_1) \wedge \mathit{arc}(f_1, \mathit{vform}, \mathit{past}) \wedge \mathit{arc}(f, \mathit{cat}, S)]\rangle$

e. $\langle \lambda t.t,$
$\lambda p.p,$
$\lambda F \lambda f \exists f_1[F(f_1) \wedge \mathit{arc}(f_1, \mathit{vform}, \mathit{pres}) \wedge \mathit{arc}(f, \mathit{cat}, S)]\rangle$

Suppose (44a) is applied to (43). Limiting attention to the phrase structure dimension for the moment, we obtain (will $\bullet_w$ (Mary $\bullet$ laugh). However, by (34a), this term is equivalent to the νt term (Mary $\bullet$ (will $\bullet$ laugh)), as desired. The procedure is somewhat reminiscent of the way in which subject moves from Spec(VP) to Spec(IP) in contemporary generative grammar. The multimodal approach lets us have our cake and eat it: We can have all arguments available on the verb already in the lexicon (which seems necessary for generating the preferred readings of sentences such as *A linux box doesn't adorn every desktop*). But the VP is still available as a syntactic unit and can combine with tense and adjuncts with the help of various wrapping operations modeling local movement.

In (45) the complete result of applying (44a) to (43) is given. This sign is *admissable* in the sense that the generated feature expression is satisfiable. Clearly, not all combinations lead to an outcome that is admissable in this sense. For example, the inflected forms in (41) must combine with the right tense. The feature description of the result will not be satisfiable otherwise. Similarly (41c) selects for a sentential complement while both complements of (41b) must be nominal; (41d) and (44c) require the relevant subject to be third person singular, etc.

(45) $\langle(\mathsf{Mary} \bullet (\mathsf{will} \bullet \mathsf{laugh})),$
$\lambda i \exists j [i < j \wedge j \approx i \wedge \mathit{laugh}(\mathit{mary})(j)],$
$\lambda f \exists f_1 f_2 [\mathit{arc}(f_2, \mathit{cat}, N) \wedge \mathit{arc}(f_2, \mathit{agr}, \mathit{3sg}) \wedge$
$\mathit{arc}(f_1, \mathit{subj}, f_2) \wedge \mathit{arc}(f_1, \mathit{vform}, \mathit{base}) \wedge \mathit{arc}(f, \mathit{cat}, S)]\rangle$

Most of the attributes and feature values used here were borrowed from Lexical-Functional Grammar (Kaplan and Bresnan 1982). That this is so is largely for convenience and choices other than the ones taken here can be explored. But the general set-up of the present grammar is I think deeply related to the set-up of LFG. The LFG levels of c-structure, f-structure and semantic structure are clearly present here, but, more importantly, there is also a convergence between our use of combinators and the LFG technique of reading off semantics from f-structure with the help of linear logic. The $\multimap$ fragment of intuitionistic linear logic is identical to the product-free undirected Lambek Calculus and is therefore closely related to the class of single-bind combinators we are using here.

We turn to quantifying expressions. The items for the determiners *a* and *every* (type (N ((NP S) S))) are given in (47), while (46) provides an example of a common noun entry.

(46) $\langle \mathsf{man}, \mathit{man}_{e(st)}, \lambda f.\mathit{arc}(f, \mathit{cat}, N) \wedge \mathit{arc}(f, \mathit{agr}, \mathit{3sg})\rangle$

(47) a. $\langle \lambda t \lambda T.T(\mathsf{a} \bullet t),$
$\lambda P' P \lambda i \exists x [P'(x)(i) \wedge P(x)(i)],$
$\lambda F \lambda \mathcal{F}.\mathcal{F}(\lambda f.F(f) \wedge \mathit{arc}(f, \mathit{cat}, N) \wedge \mathit{arc}(f, \mathit{agr}, \mathit{3sg}))\rangle$

b. $\langle \lambda t \lambda T.T(\mathsf{every} \bullet t),$
$\lambda P' P \lambda i \forall x [P'(x)(i) \rightarrow P(x)(i)],$
$\lambda F \lambda \mathcal{F}.\mathcal{F}(\lambda f.F(f) \wedge \mathit{arc}(f, \mathit{cat}, N) \wedge \mathit{arc}(f, \mathit{agr}, \mathit{3sg}))\rangle$

We can apply determiners to nouns, getting signs for *every man*, *a woman*, etc. Scope variations for, say, *every man kisses a woman* can be obtained by taking (41d) and using the technique of (17) to get the quantifiers into position, after which tense can be brought in. Here we get only two readings since present tense was treated as the identity operator, in Montague's way.

But if the tense or auxiliary does make a non-trivial semantic contribution, more readings are obtained since we can quantify-in before or after the verb is combined with its tense. For example, we obtain the $\neg\forall$ reading of *Every man doesn't laugh* by applying the sign for *every man* to (41a) and then applying (44b) to the result, as in (48a). The $\forall\neg$ reading is obtained by first composing (44b) and (41a) and applying the sign for *every man* to the result, as in (48b).

(48) a. $(44b)\Big((47b)((46))((41a))\Big)$

b. $(47b)\Big((46)\Big)\Big(\lambda\zeta.\Big[(44b)\Big((41a)(\zeta)\Big)\Big]\Big)$

5 Conclusion

We have defined Lambda Grammars, a form of multi-dimensional undirected categorial grammar. Standard undirected grammars such as **L*P** have the obvious difficulty that any permutation of a generated string will also be generated, but Lambda Grammars repair this. Directed grammars, on the other hand, have a difficulty with extraction from non-peripheral positions. Such extractions are not possible without complication of these theories but are possible in the grammars considered here (and in undirected systems in general).

This means that Lambda Grammars have an empirical edge both over directed systems and over the usual undirected ones. But what is more important perhaps is that they embody a hypothesis of *radical symmetry* between syntax and semantics. In the usual set-up of a grammar, semantic values are assigned to syntactic objects. This is perhaps most clearly the case when meanings are assigned to syntactic trees in a bottom-up compositional fashion, but it is also the case in Lambek categorial grammar where meanings are assigned to proofs via the Curry-Howard correspondence. This gives a logical priority of syntax over semantics that wholly disappears in the current set-up, where syntactic and semantic terms are treated in a completely symmetrical way.

Acknowledgments

I would like to thank Mary Dalrymple, Jan van Eijck, Philippe de Groote, Ron Kaplan, Michael Moortgat, Dick Oehrle, and Willemijn Vermaat for valuable feedback.

References

Ades, A. and Steedman, M. (1982). On the Order of Words, *Linguistics and Philosophy* **4**: 517–558.

Ajdukiewicz, K. (1935). Die syntaktische Konnexität, *Studia Philosophica* **1**: 1–27. English translation in Storrs McCall, ed., *Polish Logic, 1920–1939*, Oxford, 1967, 207–231.

Bach, E. (1979). Control in Montague Grammar, *Linguistic Inquiry*.

Bach, E. (1984). Some Generalizations of Categorial Grammars, *in* F. Landman and F. Veltman (eds), *Varieties of Formal Semantics*, Foris, pp. 1–23.

Backofen, R., Rogers, J. and Vijay-Shankar, K. (1995). A First-Order Axiomatization of the Theory of Finite Trees, *Journal of Logic, Language and Information* **4**: 5–39.

Bar-Hillel, Y. (1953). A Quasi-arithmetical Notation for Syntactic Description, *Language* **29**: 47–58.

Benthem, J. v. (1986). *Essays in Logical Semantics*, Reidel, Dordrecht.

Benthem, J. v. (1988). The Semantics of Variety in Categorial Grammar, *in* W. Buszkowski, W. Marciszewski and J. v. Benthem (eds), *Categorial Grammar*, John Benjamins, Amsterdam, pp. 37–55.

Benthem, J. v. (1991). *Language in Action*, North-Holland, Amsterdam.

Cornell, T. (1994). On Determining the Consistency of Partial Descriptions of Trees, *Proceedings of ACL-94*.

Curry, H. and Feys, R. (1958). *Combinatory Logic*, Vol. I, North-Holland, Amsterdam.

Curry, H. B. (1961). Some Logical Aspects of Grammatical Structure, *in* R. O. Jakobson (ed.), *Structure of Language and its Mathematical Aspects*, Vol. 12 of *Symposia on Applied Mathematics*, American Mathematical Society, Providence, pp. 56–68.

de Groote, P. (2001). Towards Abstract Categorial Grammars, *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*, ACL, Toulouse, France, pp. 148–155.

de Groote, P. (2002). Tree-Adjoining Grammars as Abstract Categorial Grammars, *TAG+6, Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pp. 145–150.

Gabbay, D. (1996). *Labelled Deductive Systems*, Clarendon Press, Oxford.

Gallin, D. (1975). *Intensional and Higher-Order Modal Logic*, North-Holland, Amsterdam.

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

Hendriks, H. (1988). Type Change in Semantics: the Scope of Quantification and Coordination, *in* E. Klein and J. van Benthem (eds), *Categories, Polymorphism, and Unification*, Centre for Cognitive Science, Edinburgh.

Hendriks, H. (1993). *Studied Flexibility: Categories and Types in Syntax and Semantics*, PhD thesis, University of Amsterdam.

Jacobson, P. (1999). Towards a Variable-free Semantics, *Linguistics and Philosophy.*

Johnson, M. (1991). Logic and Feature Structures, *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, Sydney, Australia.

Kamp, H. and Reyle, U. (1993). *From Discourse to Logic*, Kluwer, Dordrecht.

Kaplan, R. and Bresnan, J. (1982). Lexical-Functional Grammar: a Formal System for Grammatical Representation, *in* J. Bresnan (ed.), *The Mental Representation of Grammatical Relations*, The MIT Press, Cambridge, MA, pp. 173–281.

Lambek, J. (1958). The Mathematics of Sentence Structure, *American Mathematical Monthly* **65**: 154–170.

Montague, R. (1973). The Proper Treatment of Quantification in Ordinary English, *Formal Philosophy*, Yale University Press, New Haven, pp. 247–270.

Moortgat, M. (1988). *Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus*, PhD thesis, University of Amsterdam.

Moortgat, M. (1991a). Generalized Quantification and Discontinuous Type Constructors, *in* W. Sijtsma and A. v. Horck (eds), *Discontinuous Constituency*, De Gruyter. To appear.

Moortgat, M. (1991b). Labelled Deductive Systems for Categorial Theorem Proving, *in* P. Dekker and M. Stokhof (eds), *Proceedings of the Eighth Amsterdam Colloquium*, Amsterdam, pp. 403–423.

Moortgat, M. (1995). In Situ Binding: a Modal Analysis, *in* P. Dekker and M. Stokhof (eds), *Proceedings of the Tenth Amsterdam Colloquium*, Amsterdam, pp. 539–549.

Moortgat, M. (1997). Categorial Type Logics, *in* J. v. Benthem and A. t. Meulen (eds), *Handbook of Logic and Language*, Elsevier, pp. 93–177.

Morrill, G. (1994). *Type Logical Grammar: Categorial Logic of Signs*, Kluwer, Dordrecht.

Muskens, R. (1994). Categorial Grammar and Discourse Representation Theory, *Proceedings of COLING 94*, Kyoto, pp. 508–514.

Muskens, R. (1995). *Meaning and Partiality*, CSLI, Stanford.

Muskens, R. (1996). Combining Montague Semantics and Discourse Representation, *Linguistics and Philosophy* **19**: 143–186.

Muskens, R. (2001a). Categorial Grammar and Lexical-Functional Grammar, *in* M. Butt and T. H. King (eds), *Proceedings of the LFG01 Conference, University of Hong Kong*, CSLI Publications, Stanford CA, pp. 259–279. http://cslipublications.stanford.edu/LFG/6/lfg01.html.

Muskens, R. (2001b). Lambda Grammars and the Syntax-Semantics Interface, *in* R. van Rooy and M. Stokhof (eds), *Proceedings of the Thirteenth Amsterdam Colloquium*, Amsterdam, pp. 150–155.

Muskens, R. (2001c). Talking about Trees and Truth-conditions, *Journal of Logic, Language and Information* **10**(4): 417–455.

Oehrle, R. (1988). Multi-Dimensional Compositional Functions as a Basis for Grammatical Analysis, *in* R. Oehrle, E. Bach and D. Wheeler (eds), *Categorial Grammars and Natural Language Structures*, Reidel, Dordrecht, pp. 349–389.

Oehrle, R. (1994). Term-Labeled Categorial Type Systems, *Linguistics and Philosophy* **17**: 633–678.

Oehrle, R. (1995). Some 3-Dimensional Systems of Labelled Deduction, *Bulletin of the IGPL* **3**: 429–448.

Oehrle, R. (1999). LFG as Labeled Deduction, *in* M. Dalrymple (ed.), *Semantics and Syntax in Lexical Functional Grammar*, MIT Press, Cambridge, MA, chapter 9, pp. 319–357.

Partee, B. and Rooth, M. (1983). Generalized Conjunction and Type Ambiguity, *in* R. Baüerle, C. Schwarze and A. von Stechow (eds), *Meaning, Use and Interpretation of Language*, de Gruyter, Berlin.

Roorda, D. (1991). *Resource Logics: Proof-theoretical Investigations*, PhD thesis, University of Amsterdam.

Rooth, M. and Partee, B. (1982). Conjunction, Type Ambiguity, and Wide Scope "or", *in* D. Flickinger, M. Macken and N. Wiegand (eds), *Proceedings of the 1982 West Coast Conference on Formal Linguistics*, Stanford Linguistics Department, Stanford.

Saussure, F. d. (1916). *Cours de Linguistique Générale*.

Steedman, M. (1996). *Surface Structure and Interpretation*, MIT Press.

Szabolsci (1989). Bound Variables in Syntax (Are there any?), *in* R. Bartsch, J. van Benthem and P. van Emde Boas (eds), *Semantics and Contextual Expression. Proceedings of the Sixth Amsterdam Colloquium*, Foris, Dordrecht, pp. 295–318.

Troelstra, A. (1992). *Lectures on Linear Logic*, CSLI, Stanford.

Zeevat, H., Klein, E. and Calder, J. (1986). Unification Categorial Grammar, Manuscript.