Research on Language and Computation **00:** 1–22, 2007.

Separating Syntax and Combinatorics in Categorial Grammar

REINHARD MUSKENS (r.a.muskens@uvt.nl)
Department of Philosophy, Tilburg University, The Netherlands

Key words: categorial grammar, lambda grammar

1. Introduction

The 'syntax' and 'combinatorics' of my title are what Curry (1961) referred to as *phenogrammatics* and *tectogrammatics* respectively. Tectogrammatics is concerned with the abstract combinatorial structure of the grammar and directly informs semantics, while phenogrammatics deals with concrete operations on syntactic data structures[⋆] such as trees or strings.[⋆⋆] In a series of previous papers (Muskens, 2001a; Muskens, 2001b; Muskens, 2003) I have argued for an architecture of the grammar in which finite sequences of lambda terms are the basic data structures, pairs of terms $\langle \text{syntax}, \text{semantics} \rangle$ for example. These sequences then combine with the help of simple generalizations of the usual abstraction and application operations. This theory, which I call *Lambda Grammars* and which is closely related to the independently formulated theory of *Abstract Categorial Grammars* (de Groote, 2001; de Groote, 2002), in fact is an implementation of Curry's ideas: the level of tectogrammar is encoded by the sequences of lambda-terms and their ways of combination, while the syntactic terms in those sequences constitute the phenogrammatical level. In de Groote's formulation of the theory, tectogrammar is the level of *abstract* terms, while phenogrammar is the level of *object* terms.

⋆ Curry thought of tectogrammatics in terms of operations on *structures*, but since his writing another perspective has gained popularity. This is the *descriptions* approach to grammatical representation (pioneered in Kaplan and Bresnan, 1982 and Marcus et al., 1983). In this paper we will take the descriptions perspective, but will consider descriptions of trees, strings, and the like as belonging to the phenogrammar.

⋆⋆ Curry's distinction had almost been forgotten when attention to it was drawn in Dowty (1982). See also the highly interesting Dowty (1995), which was presented at the 1989 Tilburg conference on discontinuous constituency.

While my previous papers on the subject mainly concentrated on the tectogrammatical level of the theory and the theory's overall architecture[‡] (as is perhaps a natural start), I now want to focus on phenogrammar in some more detail. Many ways in which this dimension of the grammar could be modeled are consistent with the overall theory, but I will opt here for a *multimodal* approach that directly derives from existing work in categorial grammar (see e.g. Morrill, 1994, Moortgat, 1997). Syntax and combinatorics are interleaved in existing work on multimodal categorial grammar and are dealt with within a single generalization of the Lambek Calculus (Lambek, 1958; Moortgat, 1997), but on the present account the two will be separated. Each will each get its own calculus. In the case of tectogrammar this will be the $\multimap$ fragment of linear logic, or, equivalently, the set of linear lambda terms; for phenogrammatics we will have a pure multimodal logic[‡‡] The result is a much-needed simplification: splitting multimodal categorial grammar into a multimodal and a categorial part makes working with each of these parts humanly feasible. The categorial part will be extremely simple, and inferences in the multimodal calculus will in fact resemble derivations in generative syntax to some extent. The resemblance can be made closer or less close, depending on which phenogrammar postulates are adopted.

The approach will be illustrated with the help of a treatment of some aspects of *Dutch word order*. Within multimodal categorial grammar very interesting accounts of Dutch verb clustering and verb second have been worked out in Oehrle (1998) and Moortgat (1999)[⋆] and we will consider in some detail how a similar analysis can be carried out within the present setting. Descriptive originality will not be our aim, as the point we want to make is purely architectural.[⋆⋆] The objectives here are to recast Oehrle's and

[‡] Muskens (2001a) additionally focuses on the relation between Lambda Grammars and Lexical-Functional Grammar (Kaplan and Bresnan, 1982) and argues that the resemblance is close, especially if the theory is set up three-dimensionally, with sequences of λ-terms $\langle C, F, S\rangle$ in which C describes c-structure, F f-structure, and S is the semantics. Muskens (2003) works out an implementation of phenogrammar as a multimodal logic, as is done here, but in considerably less detail.

[‡‡] 'Pure' in the sense that the logic will be a straightforward generalization of the usual modal logics, not only model-theoretically, but also proof-theoretically (i.e. there will be no resource-sensitivity). Modal logics will be introduced as fragments of classical logic, i.e. by transcribing their Kripke-semantics.

[⋆] The work of Oehrle and Moortgat cited here goes back to Moortgat and Oehrle (1993) and joint presentations in a number of other venues (particularly, ESSLLI courses in Barcelona (1995), Aix-en-Provence (1997), and Utrecht (1999), were numerous fragments were discussed and implemented in Richard Moot's GRAIL theorem prover, `www.labri.fr/perso/moot/grail.html`).

[⋆⋆] Although I do claim that Lambda Grammars have an empirical edge over directed forms of categorial grammar (extraction from medial positions is treated without any difficulty, see Muskens (2003) and below), I do not believe that the perspective has any conclusive empirical advantages in the domain of description (Dutch word order) that was chosen for illustration here. Things do seem to become much simpler though.

Moortgat's work in a Lambda Grammars setting, to identify the technical changes to this work that are needed in order to do this and get the logical machinery going, and, hopefully, to convince the reader that conceptual simplifications are to be gained in this direction. These conceptual simplifications will also to some extent clarify the relation between multimodal categorial grammar and other approaches that are less logically oriented.

The rest of the paper will be organized as follows. The next section will introduce the overall theory. In section 3 it is explained how Lambda Grammars can be provided with a multimodal component in the syntactic dimension and how such a multimodal component can be used to obtain a treatment of some aspects of Dutch word order. A conclusion ends the paper.

2. Lambda Grammars

Lambda Grammars are a variant of Categorial Grammar (CG) that differs from existing accounts of CG in that it treats syntactic and semantic information as completely on a par in the sense that there is no asymmetric dependency of semantics upon syntax as there is in most theories of grammar. The theory also differs from standard forms of CG in that its core combinatorial engine is essentially *undirected*. No distinction is made between categories that seek material on their right and those that seek it on their left, as far as the core logical engine is concerned. It is only in one of the specialised *dimensions* of the grammar that word order (and dominance) can be brought into the picture. There are several advantages to this. One linguistic advantage is that *extraction from medial positions* becomes possible without any further addition to the theory. It is well known that directed systems such as the Lambek Calculus (Lambek, 1958; Moortgat, 1997) can handle expressions that in a transformational account would involve movement, such as *the philosopher who Plato wrote about*. This is an important advantage of the Lambek Calculus over the basic AB system (Bar-Hillel, 1953), which does not allow for the hypothetical reasoning needed here. But unfortunately, the directed character of the calculus precludes a straightforward analysis of movement from medial positions, such as in *the philosopher who Plato wrote about in the Timaeus* (see Moortgat (1997) for further discussion of the problem). There are extensions of the basic Lambek system (Morrill, 1994; Moortgat, 1997) that can deal with medial extraction, but at the price of complication and not as straightforwardly as peripheral extraction is dealt with in the original calculus. I interpret this as a sign that directionality should not be part of the basic calculus.

Modern versions of the Lambek Calculus, such as those discussed in Morrill (1994) and Moortgat (1997), are almost always *multidimensional* (Oehrle, 1988). The basic data structures of these grammars, called *signs*,

are n-tuples, where n is the (fixed) *dimensionality* of the grammar and each element of an n-tuple corresponds to a component of the grammar, e.g. ⟨syntax, semantics, features⟩. Such signs are then combined using the calculus. It can be argued almost a priori that, since signs in fact always have a syntactic (or prosodic) component, this is the place where word order information preferably should be represented. Representing directionality in the core calculus by means of the usual slashes (\ and /) is therefore unnecessary (see also the comments on the Lambek calculus in Curry, 1961).

Moving to an undirected calculus allows us to restrict ourselves to the $-\circ$ fragment of linear logic (= the calculus **L*P** of (van Benthem, 1986)), or, what boils down to the same thing, the linear (pure) λ-terms, which is a pleasantly simple system.

Before moving to our main topic, a multimodal treatment of the syntax, or phenogrammar, dimension of Lambda Gammars, we give a short introduction to the overall theory. For a fuller exposition the reader is referred to (Muskens, 2003) and to (Muskens, 2001a), which also explores the connection to Lexical-Functional Grammar (LFG, Kaplan and Bresnan, 1982). The system builds upon earlier work in CG, especially Curry (1961), Oehrle (1994) and Oehrle (1995). For more on the relation to this earlier work, again see Muskens (2003).

2.1. THE FORMAL DETAILS

The basic data structures of Lambda Grammars are n-tuples of typed λ-terms and the grammar's core logical machinery is obtained by generalizing operations on typed λ-terms in an obvious way. It will be expedient to have two kinds of types: *concrete* types for typing λ-terms and *abstract* types to type n-tuples of these. Both kinds of types are obtained by starting from a pre-given set of basic types and using the rule that (AB) is a concrete (abstract) type if A and B are concrete (abstract) types. In the examples below, signs will be pairs of λ-terms, basic concrete types will be ν (node or resource), e (entity), t (truth value), and s (world), and basic abstract types will be S, NP and N. For each dimension d (with $1 \leq d \leq n$), a *concretization operator* c^d sends abstract types to concrete types. The values of the c^d for basic abstract types can be chosen freely and in this section will be as in Table I; for complex types AB we let $c^d(AB) = c^d(A)c^d(B)$, i.e. the c^d are *type homomorphisms*. A tuple $\langle M_1, \ldots, M_n \rangle$ is said to have abstract type A if each M_i is of concrete type $c^i(A)$. *Signs* are n-tuples typed in this way.

Table I. Concretizations of abstract types used in this section

abstract type	syntax (d=1)	semantics (d=2)
S	νt	st
NP	νt	e
N	νt	$e(st)$

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 the *pointwise application* of M to N is defined as[⋆]

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

It is also possible to define *pointwise abstraction*. Call $X = \langle X_1, \ldots, X_n \rangle$ the *m-th multi-dimensional variable* of type A if each of the X_i is the m-th variable of type $c^i(A)$ (in some given ordering). 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

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

is of type AB.

We can use these pointwise application and abstraction operators to combine elements from a *lexicon* of signs. For example, supposing that the signs in (1) are in our lexicon,[⋆] we can, using (pointwise) application obtain the signs in (2), i.e. (2a) is (1c) applied to (1a) and $(2b) = ((1d)(1b))$.

(1) a. $\langle \mathsf{boy}, \mathit{boy} \rangle$: N

b. $\langle \mathsf{girl}, \mathit{girl} \rangle$: N

c. $\langle \lambda t \lambda T.T(\mathsf{every} \bullet t), \lambda P' P \lambda i \forall x[P'xi \rightarrow Pxi] \rangle$: N((NP S)S)

d. $\langle \lambda t \lambda T.T(\mathsf{a} \bullet t), \lambda P' P \lambda i \exists x[P'xi \wedge Pxi] \rangle$: N((NP S)S)

e. $\langle \lambda t_1 \lambda t_2.(t_2 \bullet (\mathsf{kisses} \bullet t_1)), \lambda x \lambda y.\mathit{kiss}\ yx \rangle$: NP(NP S)

(2) a. $\langle \lambda T.T(\mathsf{every} \bullet \mathsf{boy}), \lambda P \lambda i \forall x[\mathit{boy}\ xi \rightarrow Pxi] \rangle$: (NP S)S

[⋆] We write (AB) for the result of applying A to B and follow the usual notational conventions with respect to this notation, i.e. parentheses may be omitted when no ambiguity results and association is to the left, so that ABC is $(AB)C$.

[⋆] Some typing conventions used here can be found in Table II. Note that, although $\bullet$ is of type $(\nu t)((\nu t)(\nu t))$ and should therefore combine with two arguments of type νt to its right, we employ infix notation and write $A \bullet B$ instead of $\bullet AB$. A similar convention will hold for other operators of this type.

Table II. Some variables and constants and their types.

	syntax	semantics
variables:	$k\colon \nu$	$x, y, z\colon e$
	$t\colon \nu t$	$i, j\colon s$
	$T\colon (\nu t)(\nu t)$	$p\colon st$
		$P\colon e(st)$
constants:	$\bullet\colon (\nu t)((\nu t)(\nu t))$	$boy, girl, album, student, teacher, sleep\colon e(st)$
	$\mathsf{every}\colon \nu t$, $\mathsf{kisses}\colon \nu t, \ldots$	$kiss\colon e(e(st))$
		$show\colon e(e(e(st)))$
		$help\colon (e(st))(e(e(st)))$

b. $\langle \lambda T.T(\mathsf{a} \bullet \mathsf{girl}), \lambda P \lambda i \exists x[girl\, xi \wedge Pxi] \rangle\colon (\textsc{np}\ \textsc{s})\textsc{s}$

This can be carried further and, now using pointwise abstraction as well as application, the signs in (3) are formed (here the ζ are variables of type NP). These can then be shown to be equivalent to the signs in (4), which say that the syntax of a certain complex expression is to be associated with a certain semantics. In the example two semantic forms associate with one and the same syntactic form because the latter is ambiguous.

(3) a. $(2\mathrm{b})\lambda\zeta.[(2\mathrm{a})((1\mathrm{e})\zeta)]\colon \textsc{s}$

b. $(2\mathrm{a})\lambda\zeta'.[(2\mathrm{b})\lambda\zeta.[(1\mathrm{e})\zeta\zeta']]\colon \textsc{s}$

(4) a. $\langle (((\mathsf{every} \bullet \mathsf{boy}) \bullet (\mathsf{kisses} \bullet (\mathsf{a} \bullet \mathsf{girl}))),$
$\lambda i \exists y[girl\, yi \wedge \forall x[boy\, xi \rightarrow kiss\, xyi]] \rangle\colon \textsc{s}$

b. $\langle (((\mathsf{every} \bullet \mathsf{boy}) \bullet (\mathsf{kisses} \bullet (\mathsf{a} \bullet \mathsf{girl}))),$
$\lambda i \forall x[boy\, xi \rightarrow \exists y[girl\, yi \wedge kiss\, xyi]] \rangle\colon \textsc{s}$

The signs in (3) were obtained from those in (1) and (2) (and ultimately from those in (1) alone) by forming *linear* λ-terms over them: each abstractor λX (with X multidimensional) must bind exactly one X.[⋆] This is our general rule for generating signs, by considering linear combinations over a given

[⋆] The linearity constraint captures the *resource-sensitivity* of language. Prohibiting multiple binding of variables will prevent arbitrary duplication of linguistic material and disallowing vacuous binding prevents material to disappear altogether. This approach to resource-sensitivity is inherited from Lambek Categorial Grammar and is akin to the approach to semantic interpretation in LFG (Dalrymple et al., 1993) that uses linear logic as a 'glue' logic.

lexicon.** Signs obtained by such linear combination will be called *generated signs*.

2.2. PERMUTATION AND MEDIAL EXTRACTION

In contrast to most modern versions of categorial grammar (but in line with Ajdukiewicz, 1935) all types in Lambda Grammars are *undirected*: the application and abstraction rules make no mention of relative order of the premises. This might, at first blush, create a worry that the formalism overgenerates and does not distinguish between syntactic forms and their permutations. But such worries are unfounded. Consider the linear λ-terms over (2a), (2b) and (1e) in which each of these signs occurs exactly once. (3) gives two examples and (5) gives two more:

(5) a. $(2a)\lambda\zeta.[(2b)((1e)\zeta)]\colon s$

b. $(2b)\lambda\zeta'.[(2a)\lambda\zeta.[(1e)\zeta\zeta']]\colon s$

If the signs in (5) are worked out one gets syntax–semantics pairs for the sentence *a girl kisses every boy*, entirely as expected. But will the system overgenerate and associate (say) the syntax of *every boy kisses a girl* with the semantics of *a girl kisses every boy* or vice versa? In order to see that it does not, let us recall the well-known fact (discussed e.g. in 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 the two quantifiers and the relation each occur exactly once in the combination. In other words, (3) and (5) together exhaust the combinatorial possibilities and no unwanted syntax–semantics pairs are generated here. Although types are undirected and arguments may be permuted freely on the level of signs, such permutations always involve both the syntactic and the semantic dimension. Since syntax and semantics permute, but permute in tandem, no undesired combinations arise.

This shows that the tight coupling of syntax and semantics in Lambda Grammars manages to rein in the effects of permutation and to ensure that we do not make the bad prediction that undirected categorial grammars usually make: any permutation of a well-formed string is well-formed. Do we also get predictions that improve upon directed systems? For these we turn to extractions from medial positions. For the predictions of the standard Lambek Calculus with respect to these see Moortgat (1997), where it is

** In fact the signs generated in this way may form a superset of the signs we actually want. E.g. if we let a third grammatical component consist of λ-terms over some feature logic, as was done in (Muskens, 2001a; Muskens, 2003), we may restrict interest to those generated signs whose feature component is consistent with an axiomatisation of features such as the one in (Johnson, 1991). In the next section we will restrict our interest to signs whose syntactic dimension can be shown to consist of a property of strings.

shown that extra work is needed to get these right. On the other hand, medial extraction is no problem in Lambda Grammars, as the next example shows. Add the signs in (6) to the previous lexicon* and consider the linear combination in (7a). This reduces to (7b) and we have 'extracted' from the position directly after shows.

(6) a. $\langle \mathsf{album}, \mathit{album} \rangle$: N

b. $\langle \mathsf{Aad}, a \rangle$: NP

c. $\langle \mathsf{Marie}, m \rangle$: NP

d. $\langle \lambda t_1 \lambda t_2 \lambda t_3.(t_3 \bullet ((\mathsf{shows} \bullet t_1) \bullet (\mathsf{to} \bullet t_2))),$
$\lambda x \lambda y \lambda z \lambda i.\mathit{show}\ zxyi \rangle$: NP(NP(NP S))

e. $\langle \lambda T \lambda t.(t \bullet (\mathsf{which} \bullet (T\mathsf{e}))), \lambda P \lambda P' \lambda x \lambda i.[P'xi \wedge Pxi] \rangle$: (NP S)(N N)

(7) a. (6e)($\lambda\zeta$.(6d)ζ(6b)(6c))(6a): N

b. $\langle (\mathsf{album} \bullet (\mathsf{which} \bullet (\mathsf{Marie} \bullet ((\mathsf{shows} \bullet \mathsf{e}) \bullet (\mathsf{to} \bullet \mathsf{Aad}))))),$
$\lambda x \lambda i.[\mathit{album}\ xi \wedge \mathit{show}\ mxai] \rangle$: N

What these data and analyses seem to suggest is that the move of placing word order information in a separate syntactic dimension and freeing the type system from its usual directedness (a move already present in Oehrle, Oehrle, 1994, 1995) gives a better fit with the data. There is enough flexibility to allow extraction from medial as well as from peripheral positions,* but arbitrary permutation is avoided.

3. Multimodality

An n-dimensional Lambda Grammar combines $n+1$ logics in a completely modular way. There is one core logic of combination, 'taking linear lambda terms over the lexicon', which essentially corresponds to the $\multimap$ fragment of intuitionistic linear logic, or the logic of the combinators **B**, **C** and **I**. Moreover, each of the n dimensions consists of (closed) λ-terms over some logic. I make it a strategy to use classical type theory in each dimension and to impose any needed structure with the help of axioms. For example, in Muskens (2001a) and Muskens (2003) a feature dimension was obtained by taking λ-terms over the first-order feature logic of Johnson (1991), who

* For the definition of the empty word e, see 3.1 below.

* There are clearly many positions from which extraction is impossible and this needs to be accounted for, in directed as well as in undirected systems. What I claim here is that the medial/peripheral distinction that directed systems make is not the right generalization on which such an account should be built.

axiomatizes features using a simple set of axioms. For the semantic component we can take translations as in (Muskens, 1995) (without necessarily partializing the logic as is done there), allowing any axioms that might be needed.

The syntactic component deserves some special attention. Since phrase structure is no longer dealt with on the combinatorial, tectogrammatical, level of the grammar we must deal with it separately. But the multimodal analyses of movement and general restructuring that we find in modern versions of CG are still available if we decide that the concretizations of types such as S, NP and N should not simply denote nodes (or resources), but *sets* of these, as in Table II. Binary operators such as $\bullet$ then essentially get the type of binary modalities over the ν domain: $(\nu t)((\nu t)(\nu t))$. We can flesh this out by considering $\nu(\nu(\nu t))$ relations R^m for *modes* m and letting $\bullet_m$ be an abbreviation of

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

This is a straightforward transcription of the usual clause for a binary possibility operator in a Kripke style truth definition (see Kurtonina (1995) for the treatment of $\bullet$ as a binary modality). It is also easy to obtain unary modalities $\Diamond_m$ by transcribing the clauses for unary possibility operators, using an accessibility relation of type $\nu(\nu t)$ this time. If R^m is such a binary relation, write $\Diamond_m$ for

$$\lambda t \lambda k.\exists k_1[R^m k k_1 \wedge t k_1] \ .$$

We may also write $\Box_m$ for*

$$\lambda t \lambda k.\forall k_1[R^m k_1 k \rightarrow t k_1] \ .$$

The move to let categories such as S, NP and N (and perhaps all phrasal projections) denote sets of resources also immediately provides us with a Boolean structure and what is in effect a notion of *consequence* in the syntactic domain. Here are two abbreviations that will come in handy.

(8) a. $A \sqcap B$ abbreviates $\lambda k.Ak \wedge Bk$

b. $A \sqsubseteq B$ abbreviates $\forall k[Ak \rightarrow Bk]$

An immediate result of the previous definitions is the validity of (9).

(9) $\Diamond_m \Box_m A \sqsubseteq A$

* Note that this definition of $\Box_m$ is not literally a transcription of the usual Kripke semantics but is what we would get if the *converse* of R^m were our accessibility relation. Such modalities are often denoted $\Box_m^{\downarrow}$, but we will drop the superscript.

As is usual in modal logic, constraints on accessibility relations may be stipulated to hold in order to get an interesting consequence relation. For example, interaction between various modalities may come from *interaction postulates* as in (Moortgat, 1997). Suppose that $\bullet$, our default phrasal composition, is considered to be short for $\bullet_c$ and comes from an underlying R^c, while another ternary relation R^0 underlies an operator $\bullet_0$, which will stand for the 'head composition' of Moortgat and Oehrle (1993) and Oehrle (1998). Suppose, moreover, that the interaction postulate (10a) is adopted. Then (10b) will be an immediate consequence.

(10) a. $\forall k_1 k_2 k_3 k_4[\exists k[R^c k_1 k k_2 \wedge R^0 k k_3 k_4] \rightarrow \exists k[R^c k_1 k_3 k \wedge R^0 k k_4 k_2]]$

b. $(A \bullet B) \bullet_0 C \sqsubseteq A \bullet (B \bullet_0 C)$

When reasoning with such statements certain *monotonicity properties* are all-important. It is easy to verify that the following hold.

(11) $A \sqsubseteq A'$ entails $\Diamond_m A \sqsubseteq \Diamond_m A'$
$A \sqsubseteq A'$ entails $\Box_m A \sqsubseteq \Box_m A'$
$A \sqsubseteq A'$ entails $A \bullet_m B \sqsubseteq A' \bullet_m B$
$B \sqsubseteq B'$ entails $A \bullet_m B \sqsubseteq A \bullet_m B'$
$A \sqsubseteq A'$ entails $A \sqcap B \sqsubseteq A' \sqcap B$
$B \sqsubseteq B'$ entails $A \sqcap B \sqsubseteq A \sqcap B'$

We now have a νt domain with Boolean and modal operators, a notion of consequence (inclusion), the means to restrict the class of models with the help of postulates, and certain monotonicity properties. Enough to get some work done; let the multimodal game begin.

3.1. TREES AND STRINGS

It is the business of a grammar to connect strings with their meanings and we shall have an operator $\circ$ that is directly defined in terms of strings. Its underlying accessibility relation R° can be viewed as a partial concatenation operation on the ν domain. If $R^\circ k k_1 k_2$ is read as 'k is the result of concatenating k_1 and k_2', the following axioms (in which 1 is a type ν constant) are natural.

(12) a. $\forall k k' k_1 k_2[[R^\circ k k_1 k_2 \wedge R^\circ k' k_1 k_2] \rightarrow k = k']$

b. $\forall k k_1 k_2 k_3[\exists k'[R^\circ k k' k_3 \wedge R^\circ k' k_1 k_2] \leftrightarrow [\exists k'[R^\circ k k_1 k' \wedge R^\circ k' k_2 k_3]]$

c. $\forall k k'[R^\circ k k' 1 \rightarrow k = k']$

d. $\forall k k'[R^\circ k 1 k' \rightarrow k = k']$

The first of these is a functionality requirement; the second expresses associativity of concatenation; and the last two say that 1 is a unity element, so that the operation becomes a monoid on the subset of the ν domain for which it is defined.

Note that it need not be the case that all objects in the type ν domain can be concatenated. Some objects may be tree nodes or other 'resources' for which concatenation is unnatural. Define e to be $\lambda k.k = 1$. Then the following are direct consequences of (12).

(13) a. $(A \circ B) \circ C = A \circ (B \circ C)$

b. $A \circ \mathsf{e} \sqsubseteq A$

c. $\mathsf{e} \circ A \sqsubseteq A$

Note that a νt term such as $\mathsf{Aad} \circ \mathsf{kust} \circ \mathsf{Marie}$ can be predicated of a string k if and only if k is the concatenation of three substrings k_1, k_2 and k_3 such that $\mathsf{Aad}(k_1)$ (i.e. k_1 is a token of the string type Aad), $\mathsf{kust}(k_2)$, and $\mathsf{Marie}(k_3)$.

A natural relation connecting trees and strings is that of *yield*: string k_1 is the yield of tree k_2 if k_1 may be read off from the leaves of k_2 in the usual way. We will take it that R^y represents a slight generalization of this relation, with $R^y k_1 k_2$ standing for 'string k_1 is the yield of tree k_2 or k_1 and k_2 are both strings and $k_1 = k_2$'. The following postulates are acceptable.*

(14)
$\Diamond_y(A \bullet B) \sqsubseteq \Diamond_y A \circ \Diamond_y B$	TS1
$\Diamond_y A \sqsubseteq A$, if $A \in \mathsf{Lex}$ or $A = \mathsf{e}$	TS2
$\Diamond_y \Diamond_y A \sqsubseteq \Diamond_y A$	TS3

The first of these, TS1 (TS stands for 'Tree-String'), says that the yield of $A \bullet B$ is the yield of A concatenated with that of B. TS2 says that the yield of a lexical expression, or the expression e, is just that expression itself** and TS3, the usual 4 axiom for $\Diamond_y$, states that the yield of the yield of a tree or string is just its yield.

Now consider the term in (15).

(15) $\Diamond_y(\mathsf{Aad} \bullet (\mathsf{denkt} \bullet \Diamond_y(\mathsf{dat} \bullet (\mathsf{Marie} \bullet \mathsf{slaapt}))))$

The following derivation, which uses the monotonicity properties cited in (11) several times, shows that (15) $\sqsubseteq$ (17).

* From here on we leave it to the reader to formulate underlying postulates in terms of accessibility relations.

** Here the set Lex is defined as consisting of all those νt terms that appear as sans serif constants in our lexicon in Table IV below, minus vc and fin, which stand for features.

Table III. Concretizations of abstract types.

abstract type	syntax	semantics	abstract type	syntax	semantics
S	νt	t	INF	νt	$e(st)$
QP	νt	st	NP	νt	e
CP	νt	st	N	νt	$e(st)$
IP	νt	st			

(16)
$$\begin{array}{ll}
\Diamond_y(\mathsf{Aad} \bullet (\mathsf{denkt} \bullet \Diamond_y(\mathsf{dat} \bullet (\mathsf{Marie} \bullet \mathsf{slaapt})))) & \\
\Diamond_y\mathsf{Aad} \circ \Diamond_y(\mathsf{denkt} \bullet \Diamond_y(\mathsf{dat} \bullet (\mathsf{Marie} \bullet \mathsf{slaapt}))) & \text{TS1} \\
\mathsf{Aad} \circ \Diamond_y(\mathsf{denkt} \bullet \Diamond_y(\mathsf{dat} \bullet (\mathsf{Marie} \bullet \mathsf{slaapt}))) & \text{TS2} \\
\mathsf{Aad} \circ \Diamond_y\mathsf{denkt} \circ \Diamond_y\Diamond_y(\mathsf{dat} \bullet (\mathsf{Marie} \bullet \mathsf{slaapt})) & \text{TS1} \\
\mathsf{Aad} \circ \mathsf{denkt} \circ \Diamond_y\Diamond_y(\mathsf{dat} \bullet (\mathsf{Marie} \bullet \mathsf{slaapt})) & \text{TS2} \\
\mathsf{Aad} \circ \mathsf{denkt} \circ \Diamond_y(\mathsf{dat} \bullet (\mathsf{Marie} \bullet \mathsf{slaapt})) & \text{TS3} \\
\mathsf{Aad} \circ \mathsf{denkt} \circ \mathsf{dat} \circ \mathsf{Marie} \circ \mathsf{slaapt} & \text{etc.}
\end{array}$$

(17) $\mathsf{Aad} \circ \mathsf{denkt} \circ \mathsf{dat} \circ \mathsf{Marie} \circ \mathsf{slaapt}$

The type νt term in (17) is built from elements of Lex with the help of $\circ$ alone. Let us call such terms $\circ$-*terms*. Suppose that, given some two-dimensional lexicon with a syntactic component in the first dimension and semantics in the second, $\langle S_1, S_2\rangle$ is a generated sign and that, given a fixed set of postulates, $S_1 \sqsubseteq S_1'$ is valid. Then $\langle S_1', S_2\rangle$ is called a *derivable* sign. A sign $\langle S_1, S_2\rangle$ such that S_1 is a $\circ$-term will be called a *string-meaning* sign. We are especially interested in the derivable string-meaning signs. A string-meaning sign hypothesizes a direct relation between a certain string and one of its possible semantic readings and the set of derivable string-meaning signs that is obtained from any given lexicon constitutes a theory of the string-meaning relation found in language.

Let us give an example. Suppose that a lexicon is given which makes (18) a generated sign* and also suppose that the postulates in (14) are in force.

(18) $\langle \Diamond_y(\mathsf{Aad} \bullet (\mathsf{denkt} \bullet \Diamond_y(\mathsf{dat} \bullet (\mathsf{Marie} \bullet \mathsf{slaapt})))), \forall j[Bajw_0 \rightarrow \mathit{sleep}\ mj]\rangle$

(19) $\langle \mathsf{Aad} \circ \mathsf{denkt} \circ \mathsf{dat} \circ \mathsf{Marie} \circ \mathsf{slaapt}, \forall j[Bajw_0 \rightarrow \mathit{sleep}\ mj]\rangle$

Then the derivation in (16) shows that (19) is a derivable string-meaning sign. It establishes a connection between the Dutch sentence *Aad denkt dat*

* This is not the case for our lexicon in Table IV, which turns (18) into a *derivable* sign, not a generated sign.

Marie slaapt (*'Aad thinks that Marie is asleep'*) and the semantic term $\forall j[Bajw_0 \rightarrow sleep\, mj]$, which states that Marie is asleep in all worlds j that are belief options (B) for Aad (a) in the actual world (w_0).

Our focus on derivable string-meaning signs brings with it a special interest in derivations such as the one in (16) in which the last line is a $\circ$-term. In such derivations it is our task to get rid of all modal operators except $\circ$. With only the TS postulates of (14) in force this task is trivial, but more modal operators and more interaction postulates may bring more life into the game, as we shall see below.

3.2. GOING DUTCH

In Table IV a lexicon for a fragment of Dutch is given that will be explained in the present section. We will also give interaction postulates for some of the modalities found in this lexicon. As the reader will already have noticed, a repository of basic abstract types slightly larger than the one present in Table I is in use now. The names of these abstract types are self-explanatory, except perhaps QP, which is the category of questions (here: yes/no questions). Table III gives concretizations of these types in the syntactic and semantic dimensions.*

The third column in Table IV gives $\langle \text{syntax}, \text{semantics} \rangle$ pairs, as before. For convenience we abbreviate these with a mnemonic name in the first column, so e.g. `aad` is short for $\langle \mathsf{Aad}, a \rangle$. This in turn enables us to write derived signs such as (20) that can then be worked out in each dimension. For example, (20) turns out to be identical to the pair $\langle (21a), (21b) \rangle$.

(20) $\texttt{?}((\texttt{een}\,\texttt{docent})\lambda\zeta.(\texttt{een}\,\texttt{student})(\texttt{mag}(\texttt{kussen}\,\zeta)))$

(21) a. $\Diamond_y\Diamond_1((\mathsf{een} \bullet \mathsf{student}) \bullet (((\mathsf{een} \bullet \mathsf{docent}) \bullet \mathsf{kussen}_{\mathsf{vc}}) \bullet_0 \mathsf{mag}_{\mathsf{vc,fin}}))$

b. $\lambda i.[\exists y[teacher\, yi \wedge \exists z[student\, zi \wedge \exists j[Mji \wedge kiss\, zyj]]] \leftrightarrow$
$\exists y[teacher\, yw_0 \wedge \exists z[student\, zw_0 \wedge \exists j[Mjw_0 \wedge kiss\, zyj]]]]$

How (21a) connects to the Dutch question *Mag een student een docent kussen?* (*'May a student kiss a teacher?'*) and why (21b) is the Groenendijk-Stokhof semantics for that question (Groenendijk and Stokhof, 1984) will be seen shortly. For the moment let us just mention some of the modal operators and conventions used in (21a) and in the rest of the lexicon. The operators $\bullet$, $\Diamond_y$ and $\bullet_0$ (Oehrle's 'head adjunction') we have met before. In (21a) $\Diamond_1$ is new and is used to enforce placement of the finite verb in verb initial position, while vc and fin are terms of type νt that will act as features.

* As a minor point, note that type S now goes to t in the semantic dimension, not to st as in the previous set-up. See the discussion of the `assert` sign below.

Table IV. The Lexicon.

abbr.	abstract type	⟨syntax, semantics⟩
aad	NP	$\langle \mathsf{Aad}, a \rangle$
ben	NP	$\langle \mathsf{Ben}, b \rangle$
marie	NP	$\langle \mathsf{Marie}, m \rangle$
student	N	$\langle \mathsf{student}, \mathit{student} \rangle$
docent	N	$\langle \mathsf{docent}, \mathit{teacher} \rangle$
elke	N((NP IP)IP)	$\langle \lambda t \lambda T. \Diamond_{sc}(T \Box_{sc}(\mathsf{elke} \bullet t)), \lambda P' P \lambda i \forall x [P' x i \rightarrow P x i] \rangle$
een	N((NP IP)IP)	$\langle \lambda t \lambda T. T(\mathsf{een} \bullet t), \lambda P' P \lambda i \exists x [P' x i \wedge P x i] \rangle$
slaapt	NP IP	$\langle \lambda t. (t \bullet \mathsf{slaapt}_{\mathsf{vc},\mathsf{fin}}), \mathit{sleep} \rangle$
slapen	INF	$\langle \mathsf{slapen}_{\mathsf{vc}}, \mathit{sleep} \rangle$
kust	NP(NP IP)	$\langle \lambda t \lambda t'. (t' \bullet (t \bullet \mathsf{kust}_{\mathsf{vc},\mathsf{fin}})), \lambda xy. \mathit{kiss}\ yx \rangle$
kussen	NP INF	$\langle \lambda t. (t \bullet \mathsf{kussen}_{\mathsf{vc}}), \lambda xy. \mathit{kiss}\ yx \rangle$
helpt	INF(NP(NP IP))	$\langle \lambda t t' t''. (t'' \bullet (t' \bullet (t \bullet_0 \mathsf{helpt}_{\mathsf{vc},\mathsf{fin}}))), \mathit{help} \rangle$
helpen	INF(NP INF)	$\langle \lambda t t'. (t' \bullet (t \bullet_0 \mathsf{helpen}_{\mathsf{vc}})), \mathit{help} \rangle$
mag	INF(NP IP)	$\langle \lambda t t'. (t' \bullet (t \bullet_0 \mathsf{mag}_{\mathsf{vc},\mathsf{fin}})), \lambda P \lambda x \lambda i. \exists j [M j i \wedge P x j] \rangle$
mogen	INF INF	$\langle \lambda t. (t \bullet_0 \mathsf{mogen}_{\mathsf{vc}}), \lambda P \lambda x \lambda i. \exists j [M j i \wedge P x j] \rangle$
moet	INF(NP IP)	$\langle \lambda t t'. (t' \bullet (t \bullet_0 \mathsf{moet}_{\mathsf{vc},\mathsf{fin}})), \lambda P \lambda x \lambda i. \forall j [M j i \rightarrow P x j] \rangle$
moeten	INF INF	$\langle \lambda t. (t \bullet_0 \mathsf{moeten}_{\mathsf{vc}}), \lambda P \lambda x \lambda i. \forall j [M j i \rightarrow P x j] \rangle$
kan	INF(NP IP)	$\langle \lambda t t'. (t' \bullet (t \bullet_0 \mathsf{kan}_{\mathsf{vc},\mathsf{fin}})), \lambda P \lambda x \lambda i. \exists j [C j i \wedge P x j] \rangle$
kunnen	INF INF	$\langle \lambda t. (t \bullet_0 \mathsf{kunnen}_{\mathsf{vc}}), \lambda P \lambda x \lambda i. \exists j [C j i \wedge P x j] \rangle$
wil	INF(NP IP)	$\langle \lambda t t'. (t' \bullet (t \bullet_0 \mathsf{wil}_{\mathsf{vc},\mathsf{fin}})), \lambda P \lambda x \lambda i. \forall j [W x j i \rightarrow P x j] \rangle$
willen	INF INF	$\langle \lambda t. (t \bullet_0 \mathsf{willen}_{\mathsf{vc}}), \lambda P \lambda x \lambda i. \forall j [W x j i \rightarrow P x j] \rangle$
denkt	CP(NP IP)	$\langle \lambda t t'. (t' \bullet (\mathsf{denkt}_{\mathsf{vc},\mathsf{fin}} \bullet \Diamond_y t)), \lambda p \lambda x \lambda i. \forall j [B x j i \rightarrow p j] \rangle$
denken	CP INF	$\langle \lambda t. (\mathsf{denken}_{\mathsf{vc}} \bullet \Diamond_y t), \lambda p \lambda x \lambda i. \forall j [B x j i \rightarrow p j] \rangle$
dat	IP CP	$\langle \lambda t. (\mathsf{dat} \bullet t), \lambda p. p \rangle$
assert	IP S	$\langle \lambda t. \Diamond_y \Diamond_2 t, \lambda p. p w_0 \rangle$
?	IP QP	$\langle \lambda t. \Diamond_y \Diamond_1 t, \lambda p \lambda i. p i \leftrightarrow p w_0 \rangle$

We write $A_{B_1,\ldots,B_n}$ for $A \sqcap B_1 \sqcap \ldots \sqcap B_n$ if $B_1, \ldots, B_n$ are such features, so that $\mathsf{mag}_{\mathsf{vc},\mathsf{fin}}$ is short for $\mathsf{mag} \sqcap \mathsf{vc} \sqcap \mathsf{fin}$. Other modal operators that can be found in the syntactic dimension of the lexicon are $\Diamond_2$, which is related to placement of the verb in second position, and the set $\Box_{sc}$ and $\Diamond_{sc}$, used for checking scope boundaries, as will be explained in 3.2.4 below.

3.2.1. *The Semantic Dimension*

Semantics is not the primary focus of this paper, but we have endeavored to provide the signs generated by our lexicon with a reasonable second dimension. The set-up largely follows that of Muskens (1995). In Table IV

predicates such as *student* (of type $e(st)$) and *kiss* (of type $e(e(st))$) have an argument place of type s in addition to the usual type e argument places they need. A term such as *student* m therefore is of type st and intuitively denotes a set of possible worlds, the set of worlds in which Marie is a student. It is only when this term is applied to a term of type s, e.g. the constant w_0, which stands for the actual world, that we get a term of type t; *student* mw_0 intuitively meaning that Marie is a student (in the actual world). Quantifiers also take the extra argument place into account. The modal verbs in Table IV get a semantics that can be read as a transcription of Kripke style modalities, much in the way in which syntactic operators $\bullet_m$ and $\Diamond_m$ were defined using transcription of Kripke modalities. For example, the semantics of *mag*, $\lambda P\lambda x\lambda i.\exists j[Mji \wedge Pxj]$, leads to a translation of *Aad mag slapen* (*'Aad may sleep'*) of the form $\exists j[Mjw_0 \wedge \textit{sleep}\ aj]$, which expresses that in some world j, M-accessible from the actual world w_0, Aad is sleeping. Other modal verbs are provided with a similar semantics.⋆ The attitude verb *denken* (*'think'*) gets a classical Hintikka-like analysis that was already encountered in (18) and (19); the analysis of *willen* (*'want'*) is comparable, with a buletic accessibility relation W instead of the doxastic B.⋆⋆

Note that in our lexicon the actual world w_0 only comes into play in the semantics of the special operators `assert` and `?`. The first of these turns an IP into a declarative sentence, the second turns an IP into a (yes/no) question. While the semantics of `assert` just applies the semantics of its argument to w_0, `?` is more interesting and embodies (the simplest part of) the theory of questions developed in Groenendijk and Stokhof (1984). In short, this theory holds that the extension of a yes/no question such as *Is Marie asleep?* is the set of worlds in which Marie is asleep if she is indeed asleep and the complement of this set otherwise. Similarly, if w_0 is such that some student may kiss some teacher, then (21b) will denote the set of worlds in which that is also the case, otherwise it will denote the complement of this set. In both cases the denotation is the intension of the correct (and complete) answer.⋆

⋆ In the present limited set-up we have ignored the context-dependency of modals such as *mag, moet* and *kan*. See Kratzer (1977) for an argument why such context-dependency is important and how it can be taken into account.

⋆⋆ This is a *very* rough approximation of the semantics of *willen*. For reasons why it is less than adequate, see Heim (1992). Heim gives a nice account of buletic modalities in terms of conditionals, basing herself upon insights in Stalnaker (1984). Incorporating such an account here would take us too far afield, however.

⋆ One place in the lexicon were we have resorted to an obvious stop-gap is in the semantics of `helpen` in Table IV. The meaning of this verb is just given as a constant of the right type, $(e(st))(e(e(st)))$, and no attempt at a more fine-grained account of its lexical semantics has been made. This is because a reasonable account would certainly involve the introduction of eventualities, which is no doubt feasible but would complicate the theory in a way that is not compatible with its illustrative purpose.

3.2.2. *Verb Clusters*

Let us move to the syntactic, phenogrammatical, component and give an account of verb clustering in Dutch that is very much inspired by Oehrle (1998) and Moortgat (1999). A verb cluster 'package' of inclusion postulates as in (22) is adopted.

(22) $(A \bullet B) \bullet_0 C \sqsubseteq A \bullet (B \bullet_0 C)$ VC1
$(A \bullet B) \bullet_0 C \sqsubseteq (A \bullet_0 C) \bullet B$ VC2
$A_{\mathsf{vc}} \bullet_0 B_{\mathsf{vc}} \sqsubseteq (B \bullet A)_{\mathsf{vc}}$ VC3

The idea here is that a node marked vc always dominates a phrasal tree with only verbs at its leaves and that the only way to get rid of $\bullet_0$ involves clustering verbs with the help of VC3. Postulates VC1 and VC2 can be used to rearrange the bracketing so that VC3 may apply. As an example, consider the IP (23a), whose syntactic dimension is given in (23b).

(23) a. `((wil((helpen(kussen marie))ben))aad)`

b. $(\mathsf{Aad} \bullet ((\mathsf{Ben} \bullet ((\mathsf{Marie} \bullet \mathsf{kussen}_{\mathsf{vc}}) \bullet_0 \mathsf{helpen}_{\mathsf{vc}})) \bullet_0 \mathsf{wil}_{\mathsf{vc,fin}}))$

The only way to get rid of the two occurrences of $\bullet_0$ in (23b) is to rearrange brackets so that the three verbs form a subtree and then to percolate vc using VC3. This is carried out in (24). (In the last line of (24) the feature vc gets dropped by the Boolean property $A \sqcap B \sqsubseteq A$.[⋆])

(24) $(\mathsf{Aad} \bullet ((\mathsf{Ben} \bullet ((\mathsf{Marie} \bullet \mathsf{kussen}_{\mathsf{vc}}) \bullet_0 \mathsf{helpen}_{\mathsf{vc}})) \bullet_0 \mathsf{wil}_{\mathsf{vc,fin}}))$
$(\mathsf{Aad} \bullet ((\mathsf{Ben} \bullet (\mathsf{Marie} \bullet (\mathsf{kussen}_{\mathsf{vc}} \bullet_0 \mathsf{helpen}_{\mathsf{vc}}))) \bullet_0 \mathsf{wil}_{\mathsf{vc,fin}}))$ VC1
$(\mathsf{Aad} \bullet (\mathsf{Ben} \bullet ((\mathsf{Marie} \bullet (\mathsf{kussen}_{\mathsf{vc}} \bullet_0 \mathsf{helpen}_{\mathsf{vc}})) \bullet_0 \mathsf{wil}_{\mathsf{vc,fin}})))$ VC1
$(\mathsf{Aad} \bullet (\mathsf{Ben} \bullet (\mathsf{Marie} \bullet ((\mathsf{kussen}_{\mathsf{vc}} \bullet_0 \mathsf{helpen}_{\mathsf{vc}}) \bullet_0 \mathsf{wil}_{\mathsf{vc,fin}}))))$ VC1
$(\mathsf{Aad} \bullet (\mathsf{Ben} \bullet (\mathsf{Marie} \bullet ((\mathsf{helpen} \bullet \mathsf{kussen})_{\mathsf{vc}} \bullet_0 \mathsf{wil}_{\mathsf{vc,fin}}))))$ VC3
$(\mathsf{Aad} \bullet (\mathsf{Ben} \bullet (\mathsf{Marie} \bullet (\mathsf{wil}_{\mathsf{fin}} \bullet (\mathsf{helpen} \bullet \mathsf{kussen}))_{\mathsf{vc}})))$ VC3
$(\mathsf{Aad} \bullet (\mathsf{Ben} \bullet (\mathsf{Marie} \bullet (\mathsf{wil}_{\mathsf{fin}} \bullet (\mathsf{helpen} \bullet \mathsf{kussen})))))$ Boole

Since VC3 also reorders verbs, the typical cross-serial dependency pattern of verbal complexes in Dutch subordinate clauses results (compare (23a) with the last line of (24)). For an example of the use of VC2, in which complements are 'extraposed' to the right, consider (25a) and its syntactic dimension (25b). The short derivation (26) extraposes the sentential complement of *denken* and the subordinate clause *Aad mag denken dat Ben slaapt* (*'Aad may think that Ben is sleeping'*) appears.

[⋆] Note that the Boolean property allows us to get rid of the features vc and fin, but that using a similar property to discard elements of Lex in general will not lead to rewriting to a $\circ$-term. Remember that the latter were defined as those terms that could be obtained from terms in Lex with the help of $\circ$ only and that vc and fin (and other feature terms in a suitable extension of the fragment) are not in Lex.

(25) a. `((mag(denken(dat(slaapt ben))))aad)`

b. $(\mathsf{Aad} \bullet ((\mathsf{denken}_{\mathsf{vc}} \bullet \Diamond_y(\mathsf{dat} \bullet (\mathsf{Ben} \bullet \mathsf{slaapt}_{\mathsf{vc,fin}}))) \bullet_0 \mathsf{mag}_{\mathsf{vc,fin}}))$

(26)
$$\begin{array}{ll}
(\mathsf{Aad} \bullet ((\mathsf{denken}_{\mathsf{vc}} \bullet \Diamond_y(\mathsf{dat} \bullet (\mathsf{Ben} \bullet \mathsf{slaapt}_{\mathsf{vc,fin}}))) \bullet_0 \mathsf{mag}_{\mathsf{vc,fin}})) & \\
(\mathsf{Aad} \bullet ((\mathsf{denken}_{\mathsf{vc}} \bullet_0 \mathsf{mag}_{\mathsf{vc,fin}}) \bullet \Diamond_y(\mathsf{dat} \bullet (\mathsf{Ben} \bullet \mathsf{slaapt}_{\mathsf{vc,fin}})))) & \text{VC2} \\
(\mathsf{Aad} \bullet ((\mathsf{mag}_{\mathsf{fin}} \bullet \mathsf{denken})_{\mathsf{vc}} \bullet \Diamond_y(\mathsf{dat} \bullet (\mathsf{Ben} \bullet \mathsf{slaapt}_{\mathsf{vc,fin}})))) & \text{VC3} \\
(\mathsf{Aad} \bullet ((\mathsf{mag}_{\mathsf{fin}} \bullet \mathsf{denken}) \bullet \Diamond_y(\mathsf{dat} \bullet (\mathsf{Ben} \bullet \mathsf{slaapt})))) & \text{Boole}
\end{array}$$

The last lines in (24) and (26) are not in the required $\circ$-term form yet, but it will be possible to obtain such forms if these structures are embedded into questions or declarative sentences. Let us move to a treatment of the latter.

3.2.3. *Verb Initial, Verb Second*

It is well-known that Dutch places the finite verb in second position in declarative sentences while it places it in initial position in questions. This can be modeled if finite verbs are allowed to leave their place and raise, as in the following 'raise' package.

(27)
$$\begin{array}{ll}
A_{\mathsf{fin}} \sqsubseteq \mathsf{e} \bullet_\uparrow A_{\mathsf{fin}} & \uparrow 1 \\
A \bullet (B \bullet_\uparrow C) \sqsubseteq (A \bullet B) \bullet_\uparrow C & \uparrow 2 \\
(A \bullet_\uparrow B) \bullet C \sqsubseteq (A \bullet C) \bullet_\uparrow B & \uparrow 3
\end{array}$$

Postulate $\uparrow 1$ allows a finite verb to go into a 'raise mode' $\bullet_\uparrow$, leaving an empty element behind, while the postulates $\uparrow 2$ and $\uparrow 3$ implement the idea of raising. Clearly, the raising must also be brought to a halt again, for which the following postulates can be used. V1 realizes the verb initial position, V2 verb second.

(28)
$$\begin{array}{ll}
\Diamond_1(A \bullet_\uparrow B_{\mathsf{fin}}) \sqsubseteq B \bullet A & \text{V1} \\
\Diamond_2((A \bullet B) \bullet_\uparrow C_{\mathsf{fin}}) \sqsubseteq A \bullet (C \bullet B) & \text{V2}
\end{array}$$

Let us see how this works. In (30) the syntactic dimension (29b) of (29a) is taken and a term with the finite verb in second position is derived from it.

(29) a. `(assert((wil((helpen(kussen marie))ben))aad))`

b. $\Diamond_y\Diamond_2(\mathsf{Aad} \bullet ((\mathsf{Ben} \bullet ((\mathsf{Marie} \bullet \mathsf{kussen}_{\mathsf{vc}}) \bullet_0 \mathsf{helpen}_{\mathsf{vc}})) \bullet_0 \mathsf{wil}_{\mathsf{vc,fin}}))$

(30) $\Diamond_y\Diamond_2(\mathsf{Aad} \bullet ((\mathsf{Ben} \bullet ((\mathsf{Marie} \bullet \mathsf{kussen}_{\mathsf{vc}}) \bullet_0 \mathsf{helpen}_{\mathsf{vc}})) \bullet_0 \mathsf{wil}_{\mathsf{vc,fin}}))$
$\Diamond_y\Diamond_2(\mathsf{Aad} \bullet (\mathsf{Ben} \bullet (\mathsf{Marie} \bullet (\mathsf{wil}_{\mathsf{fin}} \bullet (\mathsf{helpen} \bullet \mathsf{kussen})))))$ (24)
$\Diamond_y\Diamond_2(\mathsf{Aad} \bullet (\mathsf{Ben} \bullet (\mathsf{Marie} \bullet ((\mathsf{e} \bullet_\uparrow \mathsf{wil}_{\mathsf{fin}}) \bullet (\mathsf{helpen} \bullet \mathsf{kussen})))))$ $\uparrow 1$
$\Diamond_y\Diamond_2(\mathsf{Aad} \bullet (\mathsf{Ben} \bullet (\mathsf{Marie} \bullet ((\mathsf{e} \bullet (\mathsf{helpen} \bullet \mathsf{kussen})) \bullet_\uparrow \mathsf{wil}_{\mathsf{fin}}))))$ $\uparrow 3$
$\Diamond_y\Diamond_2(\mathsf{Aad} \bullet (\mathsf{Ben} \bullet ((\mathsf{Marie} \bullet (\mathsf{e} \bullet (\mathsf{helpen} \bullet \mathsf{kussen}))) \bullet_\uparrow \mathsf{wil}_{\mathsf{fin}})))$ $\uparrow 2$
$\Diamond_y\Diamond_2(\mathsf{Aad} \bullet ((\mathsf{Ben} \bullet (\mathsf{Marie} \bullet (\mathsf{e} \bullet (\mathsf{helpen} \bullet \mathsf{kussen})))) \bullet_\uparrow \mathsf{wil}_{\mathsf{fin}}))$ $\uparrow 2$
$\Diamond_y\Diamond_2((\mathsf{Aad} \bullet (\mathsf{Ben} \bullet (\mathsf{Marie} \bullet (\mathsf{e} \bullet (\mathsf{helpen} \bullet \mathsf{kussen}))))) \bullet_\uparrow \mathsf{wil}_{\mathsf{fin}})$ $\uparrow 2$
$\Diamond_y(\mathsf{Aad} \bullet (\mathsf{wil} \bullet (\mathsf{Ben} \bullet (\mathsf{Marie} \bullet (\mathsf{e} \bullet (\mathsf{helpen} \bullet \mathsf{kussen}))))))$ V2

In the example a $\Diamond_2$ operator was placed by the `assert` operator. Since this operator must be gotten rid of and since the only way in which we can get rid of it is by the use of V2, the finite verb wil must enter 'raising mode'. This is done by the application of $\uparrow 1$. A series of applications of $\uparrow 2$ and $\uparrow 3$ then percolates the verb upwards until the structural conditions for application of V2 are met. After application of V2 the resulting term can easily be brought into $\circ$-term form using the TS package. In (31) we also use the fact that e is a unit for $\circ$ in order to remove this element.

(31) $\Diamond_y(\mathsf{Aad} \bullet (\mathsf{wil} \bullet (\mathsf{Ben} \bullet (\mathsf{Marie} \bullet (\mathsf{e} \bullet (\mathsf{helpen} \bullet \mathsf{kussen}))))))$
$\mathsf{Aad} \circ \mathsf{wil} \circ \mathsf{Ben} \circ \mathsf{Marie} \circ \mathsf{e} \circ \mathsf{helpen} \circ \mathsf{kussen}$ TS
$\mathsf{Aad} \circ \mathsf{wil} \circ \mathsf{Ben} \circ \mathsf{Marie} \circ \mathsf{helpen} \circ \mathsf{kussen}$ (13)

For an illustration of the V1 rule, a $\circ$-term could be derived from (21a), a task we leave to the reader.

3.2.4. *Reining in Quantifier Scope*

The tectogrammatical part of Lambda Grammars is extremely flexible and initially allows, for example, quantification into arbitrary contexts. One way of reining this in might be to put extra requirements on tectogrammatic combination, to recognize that signs can be classified according to their syntactic and semantic properties and to use such classifications to constrain pointwise application and abstraction. We shall not follow this path here but wish to point out that for blocking certain scopings there are also possibilities in the syntactic dimension. This is illustrated in the lexical entry for *elke* (*'each'*), which comes with a combination of a diamond $\Diamond_{sc}$ and a box $\Box_{sc}$ that will act as a 'lock and key' pair, much as in other forms of type logical grammar (Moortgat, 1997). In the syntactic term the box will be placed immediately before the noun phrase, while the position of the diamond corresponds to the place where 'quantifying-in' has taken place.

(32) a. $\mathtt{assert}((\mathtt{elke}\,\mathtt{student})\lambda\zeta.((\mathtt{een}\,\mathtt{docent})(\mathtt{kust}\,\zeta)))$

b. $\Diamond_y\Diamond_2\Diamond_{sc}((\mathsf{een} \bullet \mathsf{docent}) \bullet (\Box_{sc}(\mathsf{elke} \bullet \mathsf{student}) \bullet \mathsf{kust}_{\mathsf{vc,fin}}))$

c. $\forall x[\mathit{student}\, xw_0 \rightarrow \exists y[\mathit{teacher}\, yw_0 \wedge \mathit{kiss}\, yxw_0]]$

We see this illustrated in (32), where (32b) and (32c) are the syntactic and the semantic dimensions of (32a) respectively. The $\Diamond_{sc}$ diamond in (32b) indicates the scope of the noun phrase (elke • student), which is marked with a corresponding $\Box_{sc}$. As things stand, (32b) is not reducible to a $\circ$-term, but this will change as soon as the $\Box_{sc}$ box is allowed to travel upward. The following scope package makes such upward travel possible.

(33) $\Box_{sc}A \bullet B \sqsubseteq \Box_{sc}(A \bullet B)$ SC1
$A \bullet \Box_{sc}B \sqsubseteq \Box_{sc}(A \bullet B)$ SC2

In (34) it is shown how two applications of SC bring the $\Box_{sc}$ box adjacent to its corresponding diamond, after which the general $\Diamond_m\Box_m A \sqsubseteq A$ rule may apply and both box and diamond can be gotten rid of. The rest of the derivation can proceed in a way now familiar.

(34)
$\Diamond_y\Diamond_2\Diamond_{sc}((\mathsf{een} \bullet \mathsf{docent}) \bullet (\Box_{sc}(\mathsf{elke} \bullet \mathsf{student}) \bullet \mathsf{kust}_{\mathsf{vc,fin}}))$
$\Diamond_y\Diamond_2\Diamond_{sc}((\mathsf{een} \bullet \mathsf{docent}) \bullet \Box_{sc}((\mathsf{elke} \bullet \mathsf{student}) \bullet \mathsf{kust}_{\mathsf{vc,fin}}))$ SC1
$\Diamond_y\Diamond_2\Diamond_{sc}\Box_{sc}((\mathsf{een} \bullet \mathsf{docent}) \bullet ((\mathsf{elke} \bullet \mathsf{student}) \bullet \mathsf{kust}_{\mathsf{vc,fin}}))$ SC2
$\Diamond_y\Diamond_2((\mathsf{een} \bullet \mathsf{docent}) \bullet ((\mathsf{elke} \bullet \mathsf{student}) \bullet \mathsf{kust}_{\mathsf{vc,fin}}))$ (9)
$\vdots$
$\mathsf{een} \circ \mathsf{docent} \circ \mathsf{kust} \circ \mathsf{elke} \circ \mathsf{student}$

Note that the package in (33) only permits a $\Box_{sc}$ box to percolate upwards through an uninterrupted sequence of •s. This means that for a $\Box_{sc}$ to reach a $\Diamond_{sc}$ there must be an uninterrupted path of conventional phrasal combinations between them and the mechanism in fact functions as a check as to whether such a path exists.

Such paths are not always in existence. In fact, we have used the $\Diamond_y$ operator to obstruct communication across clause boundaries. In (35a) an attempt is made to universally quantify into a subordinate clause. (35a) has a perfectly acceptable semantic dimension, but in phenogrammar any attempt at deriving a $\circ$-term must fail. The $\Diamond_y$ that was put on the subordinate clause by the verb *denkt* intervenes between $\Box_{sc}$ and $\Diamond_{sc}$ and neither can be dropped from the derivation. The treatment is reminiscent of the blocking procedure in Morrill (1994).

(35) a. $\mathtt{assert}((\mathtt{elke}\,\mathtt{docent})\lambda\zeta.(\mathtt{een}\,\mathtt{student})(\mathtt{denkt}(\mathtt{dat}(\mathtt{kust}\,\mathtt{marie}\,\zeta))))$

b. $\Diamond_y\Diamond_2\Diamond_{sc}((\mathsf{een} \bullet \mathsf{student}) \bullet$
$(\mathsf{denkt}_{\mathsf{vc,fin}} \bullet \Diamond_y(\mathsf{dat} \bullet (\Box_{sc}(\mathsf{elke} \bullet \mathsf{docent}) \bullet (\mathsf{Marie} \bullet \mathsf{kust}_{\mathsf{vc,fin}})))))$

4. Conclusion

In this paper we have argued that a separation between combinatorics and syntax, or, in Curry's words, tectogrammar and phenogrammar, can considerably simplify the architecture of modern categorial grammar, and in particular its multimodal variant. Multimodal categorial grammar can be split into a categorial part, whose logic we have assumed to be the logic of the linear combinators here, and a multimodal part, whose logic depends on the properties that are stipulated to hold for a collection of underlying accessibility relations. An important difference between the two levels is that semantics is dependent upon tectogrammar and not upon the phenogrammatic level. All re-ordering and re-grouping of syntactic material can take place in the dimension of phenogrammar and will be independent from what happens in the semantic dimension.

How should the division of labour between the two levels be organized? A reasonable rule of thumb seems to be that, since the abstract level is what form and meaning have in common, in order to obtain greatest modularity it should *only* contain that which is common to form and meaning. This rules out the option of dealing with word order at the tectolevel, for example, and in general this modularity assumption will drive us towards a rather abstract, universal, and minimalistic conception of this level. But some meaningful questions about what should go where remain. For example, while we have assumed here that the *resource sensitivity* of language is common to form and meaning and therefore tecto (modeled with the help of a linearity constraint on lambda terms), there is some room to doubt whether this is correct. Conceivably resource sensitivity should not go the same way as word order and be modeled in the pheno dimension. Sharing of variables is very common in semantics, after all. Other considerations may go in the other direction and may enrich the tectolevel somewhat. Ultimately the deciding factor is empirical, of course, and hinges on the question what will give the simplest theory of the form-meaning relation in language.

Acknowledgements

I would like to thank the two anonymous referees for providing me with sets of careful and detailed comments. Glyn Morrill also sent me some highly useful remarks.

References

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.

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

van Benthem, J.: 1986, *Essays in Logical Semantics*. Dordrecht: Reidel.
van Benthem, J.: 1991, *Language in Action*. Amsterdam: North-Holland.
Curry, H.: 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*. Providence: American Mathematical Society, pp. 56–68.
Dalrymple, M., J. Lamping, and V. Saraswat: 1993, 'LFG Semantics via Constraints'. In: *Proceedings of the Sixth Meeting of the European ACL*. European Chapter of the Association for Computational Linguistics.
de Groote, P.: 2001, 'Towards Abstract Categorial Grammars'. In: *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*. Toulouse, France, pp. 148–155, ACL.
de Groote, P.: 2002, 'Tree-Adjoining Grammars as Abstract Categorial Grammars'. In: *TAG+6, Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*. pp. 145–150.
Dowty, D.: 1982, 'Grammatical Relations and Montague Grammar'. In: P. Jacobson and G. Pullum (eds.): *The Nature of Syntactic Representation*. Dordrecht: Reidel, pp. 79–130.
Dowty, D.: 1995, 'Toward a Minimalist Theory of Syntactic Structure'. In: H. Bunt and A. van Horck (eds.): *Syntactic Discontinuity*. The Hague: Mouton, pp. 11–62. (Paper originally presented at a 1989 conference).
Groenendijk, J. and M. Stokhof: 1984, 'Studies on the Semantics of Questions and the Pragmatics of Answers'. Ph.D. thesis, University of Amsterdam.
Heim, I.: 1992, 'Presupposition Projection and the Semantics of Attitude Verbs'. *Journal of Semantics* **9**, 183–221.
Johnson, M.: 1991, 'Logic and Feature Structures'. In: *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*. Sydney, Australia.
Kaplan, R. and J. Bresnan: 1982, 'Lexical-Functional Grammar: a Formal System for Grammatical Representation'. In: J. Bresnan (ed.): *The Mental Representation of Grammatical Relations*. Cambridge, MA: The MIT Press, pp. 173–281.
Kratzer, A.: 1977, 'What "must" and "can" must and can mean'. *Linguistics and Philosophy* **1**, 337–355.
Kurtonina, N.: 1995, 'Frames and Labels: A Modal Analysis of Categorial Inference'. Ph.D. thesis, Institute for Logic, Language and Computation, Amsterdam.
Lambek, J.: 1958, 'The Mathematics of Sentence Structure'. *American Mathematical Monthly* **65**, 154–170.
Marcus, M., D. Hindle, and M. Fleck: 1983, 'D-theory: Talking about Talking about Trees'. In: *Proceedings of the 21st ACL*. pp. 129–136.
Moortgat, M.: 1997, 'Categorial Type Logics'. In: J. v. Benthem and A. t. Meulen (eds.): *Handbook of Logic and Language*. Elsevier, pp. 93–177.
Moortgat, M.: 1999, 'Meaningful Patterns'. In: *JFAK, Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday*. Vossiuspers/Amsterdam University Press. CD-Rom.
Moortgat, M. and R. Oehrle: 1993, 'Adjacency, Dependency and Order'. In: P. Dekker and M. Stokhof (eds.): *Proceedings of the 9th Amsterdam Colloquium*. Amsterdam: ILLC/Department of Philosophy, University of Amsterdam, pp. 447–466.
Morrill, G.: 1994, *Type Logical Grammar: Categorial Logic of Signs*. Dordrecht: Kluwer.
Muskens, R.: 1995, *Meaning and Partiality*. Stanford: CSLI.
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*. Stanford CA, pp. 259–279, CSLI Publications. `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.: 2003, 'Language, Lambdas, and Logic'. In: G.-J. Kruijff and R. Oehrle (eds.): *Resource Sensitivity in Binding and Anaphora*, Studies in Linguistics and Philosophy. Kluwer, pp. 23–54.

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*. Dordrecht: Reidel, 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.: 1998, 'Multi-Modal Type-Logical Grammar'. In: R. Borsley and K. Borjars (eds.): *Non-transformational Syntax*. Blackwell. to appear.

Stalnaker, R.: 1984, *Inquiry*. Cambridge, MA: MIT Press.