Institute for Language, Logic and Information

TYPE CHANGE IN SEMANTICS: THE SCOPE OF QUANTIFICATION AND COORDINATION

Herman Hendriks

ITLI Prepublication Series 87-09

University of Amsterdam

TYPE CHANGE IN SEMANTICS: THE SCOPE OF QUANTIFICATION AND COORDINATION

Herman Hendriks
Department of Philosophy
University of Amsterdam

Received December 1987

Research sponsored by Z.W.O.

To be published in
E. Klein and J. van Benthem (eds.),
Categories, Polymorphism and Unification
Edinburgh/Amsterdam 1987

0. Introduction

In this paper we will try to integrate two distinct views on the relationship between syntax and semantics, viz., 'type-driven translation' and 'type ambiguity', into a single theory of flexible Montague grammar.

The integration of both views is a desirable matter, since type-driven translation is an indispensable part of Generalized Phrase Structure Grammar, a highly interesting universal theory of generative grammar, while type ambiguity has shown its merits in the treatment of several phenomena within the field of model-theoretic semantics. Moreover, we claim that the combination of type-driven translation and type ambiguity enables us to account for quantifier scope and *de dicto/de re* ambiguities in a way which allows us to dispense with syntactic rules of quantifying-in (see Montague (1974)) and with Cooper stores (see Cooper (1983)), which have been exposed to syntactic and semantic criticism, respectively.

Type-driven translation was introduced in Gazdar, Klein, Pullum, and Sag (1985), and Klein and Sag (1985). It boils down to the introduction of a general translation procedure instead of the stipulation of a separate semantic rule for each syntactic rule in the grammar.

Type ambiguity has been argued for by Partee and Rooth (1983), Groenendijk and Stokhof (1984, 1987), among others. Within this approach, the expressions of a category are assigned meanings of (infinitely many) different types.

A brief sketch of the ideas involved in these two views will be given in section 1 and 2 of this paper.

In its original formulation, type-driven translation seems to rely on a rigid category-to-type assignment by the grammar (witness the epithet 'type-driven'), a requirement which is definitely undermined by the flexible view on type assignment held by the authors on type ambiguity. So it might be unclear how these approaches could ever be put to use together in a single grammar. In section 3, however, we will see that this incompatibility is only apparent. An outline of a framework unifying both approaches, 'Flexible Montague grammar', will be given there.

Section 4 is concerned with a first attempt to fill in the details of this outline. A non-directional Lambek calculus, together with the semantics given for it by van Benthem (in, e.g., 1986), will be examined with respect to its usefulness as a candidate supplier of derived types and meanings in a flexible Montague grammar. It will appear that the system, in its initial (directional) version devised for syntactic purposes, does not pass the test. The reason for this is that directional subtleties of the calculus are lost as soon as it is applied to logical types, which are non-directional.

That is why section 5 contains an alternative calculus, 'The quantification calculus'. This simple calculus avoids the problems that arise in applying the Lambek-van Benthem calculus in semantics. Nevertheless, it is strong enough to account for scope ambiguities without syntactic quantification rules or Cooper stores. Moreover, it is able to solve problems that remained unsolved in earlier proposals. This will become clear in the sixth section, 'Applications'.

1. Type-driven translation

When Gazdar, Klein, Pullum and Sag (1985, henceforth: GKPS) introduced the idea of type-driven translation, it was in fact necessity made a virtue. In Generalized Phrase Structure Grammar (GPSG), the syntactic rules have been radically impoverished in order to be able to capture interesting syntactic generalizations. As a consequence, a lot of semantically significant properties of trees are absent from the syntactic rules, and only get introduced in the transition from rules to trees.

Within formal approaches to the semantic analysis of natural language, however, it had been commonplace to accept the strategy that for every syntactic rule within the grammar, a corresponding semantic rule must be stated which specifies how structures of the sort analyzed by that rule are to be interpreted. Given the organization of the GPSG framework, this position is no longer tenable.

Therefore, GKPS replaced this 'rule-to-rule hypothesis' by 'shake 'n bake semantics'[1], also known as type-driven translation.

Instead of stipulating a corresponding semantic rule for every syntactic rule, the grammar contains a general translation procedure. The syntax defines a set of trees, and following the procedure from the bottom upward, each tree is compositionally interpreted. Let us consider a simple example: we assume that (i) for every lexical item α of category C, the grammar will admit a local tree[2] consisting of a daughter α, and a mother C plus its translation α', and (ii) every syntactic category is associated with a (single) semantic type. For example: TYP(S) = t, TYP(NP) = ((e,t),t) = T, TYP(VP) = (T,t), etc.

We interpret a non-lexical tree in the following way: given a partly interpreted local tree t with mother C_0 and translated daughters (C_1, α_1'), ..., (C_n, α_n'), it is our task to determine the translation α_0' of the mother node. The procedure is as follows:

[1] Terms due to Emmon Bach.
[2] A local tree is a tree of depth one.

(I) if one of the daughter nodes, C_i, say, immediately dominates *and/or*, then apply generalized conjunction/disjunction to $\{\alpha_1',...,\alpha_n'\} - \{\alpha_i'\}$ in order to obtain a translation α_0' of type $TYP(C_0)$,

(II) else, apply functional application in order to obtain a translation α_0' of type $TYP(C_0)$.

Generalized conjunction and disjunction are defined for expressions of a so-called 'conjoinable' or 'relational type', i.e., a type $(c_1,(...,(c_n,t)...))$: suppose $A = \{X_1,...,X_m\}$ is a finite set of expressions of type $(c_1,(...,(c_n,t),...))$, then GC(A), the *generalized conjunction* of that set, is

(1) $\lambda x_{c_1}...\lambda x_{c_n}.[X_1(x_{c_1})...(x_{c_n}) \,\&\, ... \,\&\, X_m(x_{c_1})...(x_{c_n})]$

and GD(A), the *generalized disjunction,* is

(2) $\lambda x_{c_1}...\lambda x_{c_n}.[X_1(x_{c_1})...(x_{c_n}) \vee ... \vee X_m(x_{c_1})...(x_{c_n})]$.

In the case of functional application, (II), the daughter translations cannot simply be considered to form a set. Sometimes, the order of application is relevant: the 'flat' VP *give Fido Sandy* $([_{VP}[_V...][_{NP1}...][_{NP2}...]])$ cannot translate as both GIVE(f)(s) and GIVE(s)(f). We shall assume that the input of the translation procedure is an n-tuple rather than a set here, the order of the functional applications being determined by additional syntactic or morphological information (case, or word order, say). Suppose V applies first to NP1 and then to NP2, then the application 3-tuple is V, NP1, NP2. The *functional application* of an n-tuple $\alpha_1',...,\alpha_n'$ $(n > 1)$, $FA(\alpha_1',...,\alpha_n')$, is defined inductively for α_1 of type $(TYP(\alpha_2),(...,(TYP(\alpha_n),b)...))$:

(3) $FA(\alpha_1,\alpha_2) = \alpha_1(\alpha_2)$
$FA(\alpha_1,...,\alpha_{n+1}) = FA(\alpha_1,...,\alpha_n)(\alpha_{n+1})$.

Of course, there is more to the semantics of the grammar than just generalized conjunction/disjunction and functional application[3], but for expository purposes we shall restrict ourselves to these clauses in the remainder of the present paper.

[3] Cf., for a more sophisticated example, GKPS (1985), pp.230-231.

2. Type ambiguity

Orthodox Montague grammar requires that every syntactic category be assigned one single semantic type, and every expression only translations of the type assigned to the category the expression belongs to. Partee and Rooth (1983), Groenendijk and Stokhof (1984, 1987), Keenan and Faltz (1984), van Benthem (1986) and Partee (1984) argue that this state of affairs calls for a change and urge the necessity of a more flexible way of type assignment. In this literature on type ambiguity, one can find several interrelated motivations for the introduction of multiple types and translations for expressions. A brief survey will be given below. In this survey (and in the sequel), the type theory will be kept extensional: i.e., s's will be omitted.

(A) *Minimal type assignment..*

Montague grammarians have the habit of generalizing to the 'worst case': if a category contains some expressions that necessitate the assignment of a complex semantic type, the other expressions of that category will have to join them. The category will be assigned the most complex type, and the extensionality of the other expressions is accounted for by means of meaning postulates.

For instance, *John* is assigned the translation λP.P(j) of type ((e,t),t), and not simply j of type e, because the category it belongs to also contains expressions like *a policeman*, and we are able to conjoin them: *John and a policeman.*

Likewise, an extensional expression like *find*, for instance, is not translated as FIND of type (e,(e,t)), but (in effect) as λTλx.T(λy.FIND(y)(x)) of type (((e,t),t),(e,t)), because there are also transitive verbs like *seek,* which do not denote two-place relations between individuals.

The type ambiguity approach suggests that we translate *John* as j and *find* as FIND, and that the more complex translations be derived from the basic translations by rule. Partee and Rooth (1983) show that this reverse strategy (derivation from basic types instead of type-simplification by means of meaning postulates) is not just a nice alternative, but a necessity (cf. example (5), below). There is not always a 'worst case' to generalize to. Groenendijk and Stokhof (1984, 1987) add to this that certain intuitive entailment relations are lost on the 'worst case' level – even if there would be such a level.

(B) *Flexible type assignment.*

Expressions like *and, or* and *not* can occur in a great number of syntactic environments. *Not*, for instance, can occur as sentence negation (*it does not rain*), but also as predicate negation (*everybody does not smoke*). In orthodox Montague grammar this entails that such expressions will have to be considered either as syncategorematic expressions occurring in various syntactic rules, or as multiply ambiguous expressions (each interpretation, moreover, occurring in its own syntactic rule). This predicament can be avoided by defining the semantic operation of generalized negation: if α is of type $(c_1,(\ldots,(c_n,t),\ldots))$, then $GN(\alpha)$, the *generalized negation* of α, is $\lambda x_{c_1}\ldots\lambda x_{c_n}.\neg[\alpha(x_{c_1})\ldots(x_{c_n})]$. Thus we can distinguish, so to speak, a meaning of type (t,t): $\neg$ (used for sentence negation), and other meanings, e.g.: $\lambda P.\neg(P)$ (predicate negation) of type ((e,t),(e,t)).

(C) *Ambiguous type assignment.*
This motivation is dependent on the other two. If we have ambiguous type assignment to lexical expressions, as sketched above, why not exploit this situation for an account of structural ambiguities? Ambiguities that come to mind are, first and foremost, the quantifier scope and *de dicto*/*de re* ambiguities. In earlier work, these called for the introduction of special rules of quantifying-in, which are objectionable from the viewpoint of syntax, since they are purely semantically motivated syntactic rules. The Cooper stores (Cooper 1983) that were meant to replace them, in their turn met with criticism from a semantic point of view (Janssen (1983), Landman and Moerdijk (1983)). So it would be very useful if type ambiguity could help to account for them. For instance, if an expression like *seek* were to have $\lambda T\lambda x.T(\lambda y.\mathrm{SEEK}(\lambda P.P(y)(x))$ of type (((e,t),t),(e,t)) among its derived translations, we would be able to account for the *de re* reading of (4) without quantification rules or Cooper stores:

(4) John seeks a unicorn

But there are also structural ambiguities involving the scope of coordination, which could be treated with the help of type ambiguity. Partee and Rooth (1983) note that it is possible to obtain the *de dicto* 'wide scope *or*'-reading (the reading suggested by the continuation ('...but I don't know which') of (5), once you allow terms like *a fish,* which have as their basic translation A(FISH) of type ((e,t),t), to have a derived translation $\lambda\mathbf{P}.\mathbf{P}(\mathrm{A(FISH)})$ of type ((((e,t),t),t),t).

(5) John is looking for a fish or a bike

In the sequel we will concentrate on (A) and (C). The flexibility required in (B) is restricted to specific lexical items, and can be supplied by adopting generalized conjunction and disjunction and generalized negation among the semantic operations R (cf. section 3) in the grammar.

3. Flexible Montague grammar

As announced in the introductory section, we wish to integrate type-driven translation and type ambiguity into a theory of 'flexible Montague grammar'. Before we start to work out details, however, we will have to tell something about the general organization of the framework (cf. Groenendijk and Stokhof (1984)):

(a) First, each syntactic category C is associated with not just one type, but a set of types: its *type-set* T(C). The type-set consists of a basic type, $t_B(C)$, together with types which are predictable from the basic type by general procedures. An expression α of a category C can be assigned any of the types which belong to the type-set associated with C.

(b) Similarly, every lexical expression α of category C will be associated with a *translation-set* $Tr(\alpha)$, i.e., a set of semantic translations. One of these will be a basic translation $tr_B(\alpha)$ (which need not be of the basic type) and the other members of the translation-set will again be predictable from the basic translation by general procedures. For every $\alpha' \in Tr(\alpha)$ it will hold that for some type $a \in T(C)$, α' is of type a.

(c) Finally, the translation-set of a compound expression α_0 of category C_0, $Tr(\alpha_0)$, is to be determined from the translation-sets of its parts $\alpha_1,...,\alpha_n$: $Tr(\alpha_1),...,Tr(\alpha_n)$. So there will need to be a semantic operation R which yields a member of $Tr_B(\alpha_0)$ (the basic translation set of α_0), $R(\alpha_1',...,\alpha_n')$, whenever it is applicable to $\alpha_1' \in Tr(\alpha_1)$,..., and $\alpha_n' \in Tr(\alpha_n)$. Moreover, the general procedures mentioned in (b) will also apply to the members of Tr_B, yielding the other, predictable, members of $Tr(\alpha_0)$.

The general procedures hinted at above constitute a theory of type ambiguity, whereas the semantic operations R form a (type-driven) translation procedure.

Let us add some characters to this highly abstract story.

(a) As for the basic types associated with syntactic categories, we will be content with the following assignment:

C	$t_B(C)$	C	$t_B(C)$
NP:	e	VP:	(e,t)
S:	t	TV:	(e,(e,t))
CN:	(e,t)	DTV:	(e,(e,(e,t)))
Det:	((e,t),((e,t),t))	PV:	(t,(e,t))

For the time being (i.e., up to section 5), we will assume that the predictable types of the type-sets are supplied by the Lambek calculus. I.e., $T(C) = \{ a : t_B(C) \Rightarrow a$ is a Lambek theorem$\}$.

(b) Some examples of basic translations:

α	category	$tr_B(\alpha)$	type
John	NP	j	e
a	Det	$\lambda Q\lambda P.\exists x[Q(x) \& P(x)]$	((e,t),((e,t),t))
man	CN	MAN	(e,t)
find	TV	FIND	(e,(e,t))
seek	TV	SEEK	(((e,t),t),(e,t))
claim	PV	CLAIM	(t,(e,t))

Note that the basic translation of *seek* is not of the basic type assigned to TV, and that the application of the basic translation of *a* to the basic translation of *man* yields an expression of category NP which is not of the basic type associated with that category.

We will assume (up to section 5) that the predictable translations will be supplied by the van Benthem semantics for the Lambek calculus. I.e., $Tr(\alpha) =$ $\{\tau_c[x_a{:=}tr_B(\alpha)] : tr_B(\alpha)$ is of type a and $a \Rightarrow c$ is a Lambek theorem with semantics $\tau_c<x_a>\}$,
where $\tau_c<x_a>$ is a λ-term of type c with a free variable, x, of type a.

(c) With respect to the semantic operations, we would like to restrict ourselves to the operations of generalized conjunction/disjunction and functional application of the type-driven translation procedure given in section 1. However, the simplified picture we sketched there has one feature which makes it hard to integrate it into a framework with type ambiguity. The clauses there ran:

(I) if (...), then apply generalized conjunction/disjunction (...) in order to obtain a translation α_0' *of type TYP(C_0),*

(II) else, apply functional application in order to obtain a translation α_0' of *type TYP(C_0).*

This feature is, of course, the restriction to a single type, TYP(C_0). Our solution to this problem is straightforward: we simply omit the italicized part of the clauses. More formally:

(III) if one of the daughter nodes, Ci, say, immediately dominates *and/or,* then the set of basic translations of α_0, $Tr_B(\alpha_0)$ = $\{GC/GD(\{\alpha_1',...,\alpha_n'\} - \{\alpha_i'\}) : \alpha_1' \in Tr(\alpha_1),...,\alpha_n' \in Tr(\alpha_n)\}$.

(IV) else, the syntactically or morphologically determined application n-tuple is $\alpha_1,...,\alpha_n$, and then $Tr_B(\alpha_0)$ = $\{FA(\alpha_1',...,\alpha_n') : \alpha_1' \in Tr(\alpha_1),...,\alpha_n' \in Tr(\alpha_n)\}$.

Note that the semantic operations GC, GD and FA have become partial, in a sense, with respect to $Tr(\alpha_1),...,Tr(\alpha_n)$. Not for every $\alpha_1' \in Tr(\alpha_1),...,\alpha_n' \in Tr(\alpha_n)$, GC/GD/FA($\alpha_1',...,\alpha_n'$) is defined. For example, Tr_B(*John and Mary*) does not contain the generalized conjunction of j_e and m_e, since type e fails to be a conjoinable type; neither does Tr_B(*seek John*) contain the functional application of SEEK (type (((e,t),t),(e,t))) and j_e, for such a thing does not exist: the functor SEEK is not of the required type (TYP(j),b). But this is not a problem: since we will also have the translations λP.P(j) and λP.P(m) at our disposal, we can be confident that Tr_B(*John and Mary*) as well as Tr_B(*seek John*) will be nonempty:
GC({λP.P(j),λP.P(m)}) = λP.[P(j)&P(m)]∈ Tr_B(*John and Mary*), and
FA(SEEK,λP.P(j)) = SEEK(λP.P(j))∈ Tr_B(*seek John*).

Finally, $Tr(\alpha_0)$, the set of translations of α_0, is the following set: $Tr(\alpha_0)$ = $\{\tau_c[x_a:=\alpha_a'] : a \Rightarrow c$ is a Lambek theorem with semantics $\tau_c<x_a>$ and $\alpha_a' \in Tr_B(\alpha_0)\}$.

4. The Lambek-van Benthem calculus

Conceived of as a semantic system, the Lambek calculus is a system that gives rules for rewriting sequences of types to types. In its original version (cf. Lambek (1958)), it is a theory of syntactic categories which have a function/argument structure (like semantic types), but a special one. Function categories are directional: they are left- or right-searching. This is a property semantic types lack.

That there is, nevertheless, a strong semantic appeal to the Lambek system, has repeatedly been stressed by Johan van Benthem (cf., e.g., 1986), who has focused on several non-directional versions of the calculus, and provided them with a type-theoretical semantics.

In this section, we shall present yet another version of the non-directional calculus, which has the nice property that its rules correspond neatly with the operations application, abstraction and substitution in the semantics. The exact correspondence of this version (which is equivalent to the calculus given in van Benthem (1987)) enables us to present the van Benthem semantics (which is somewhat implicit in his (1986)) in a completely explicit way.

(6) A non-directional Lambek-van Benthem calculus:

(a) axioms: $a \Rightarrow a$

(b) modus ponens: $\dfrac{X \Rightarrow b \quad Y \Rightarrow (b,c)}{XY \Rightarrow c}$ $\quad \dfrac{Y \Rightarrow (b,c) \quad X \Rightarrow b}{YX \Rightarrow c}$

(c) conditionalization: $\dfrac{XaY \Rightarrow b}{XY \Rightarrow (a,b)}$

(d) cut: $\dfrac{XaZ \Rightarrow b \quad Y \Rightarrow a}{XYZ \Rightarrow b}$

(In the above sequents, a, b, and c stand for types, and X, Y, and Z for sequences of types: $a_1,...,a_n$, where $n \geq 0$).

(7) Semantics: assign a term $\tau_b\langle\vec{x}_X\rangle$ to every sequent $X \Rightarrow b$:

(a) axioms: assign x_a to $a \Rightarrow a$

(b) modus ponens: if $X \Rightarrow b$ and $Y \Rightarrow (b,c)$ have been assigned $\tau_b\langle\vec{x}_X\rangle$ and $\tau_{(b,c)}\langle\vec{y}_Y\rangle$, assign $\tau_{(b,c)}(\tau_b)$ to $XY \Rightarrow c$

(c) conditionalization: if $XaY \Rightarrow b$ has been assigned $\tau_b\langle\vec{x}_{XY},x_a\rangle$, assign $\lambda x_a.\tau_b$ to $XY \Rightarrow (a,b)$

(d) cut: if $XaZ \Rightarrow b$ and $Y \Rightarrow a$ have been assigned $\tau_b\langle\vec{x}_{XZ},x_a\rangle$ and $\tau_a\langle\vec{y}_Y\rangle$, assign $\tau_b[x_a:=\tau_a]$ to $XYZ \Rightarrow b$

($\tau_b\langle\vec{x}_X\rangle$ is a term of type b with free variables $\vec{x} = x_1,...,x_n$ of type $X = a_1,...,a_n$.)

The Lambek-van Benthem calculus is a very rich (i.e., flexible) calculus. Its rewriting *sequences* of types to types is already an abundance from our semantic point of view. We would be satisfied with a calculus which rewrites (single) types to types. It is therefore sufficient for our purposes to concentrate on some important theorems together with their semantics:

(8) commutativity: $(a,(b,c)) \Rightarrow (b,(a,c))$ / $\lambda x_b \lambda y_a . z_{(a,(b,c))}(y)(x)$

(9) Geach: $(b,c) \Rightarrow ((a,b),(a,c))$ / $\lambda x_{(a,b)} \lambda y_a . z_{(b,c)}(x_{(a,b)}(y))$

(10) composition: $(a,b)\ (b,c) \Rightarrow (a,c)$ / $\lambda y_a . z_{(b,c)}(x_{(a,b)}(y))$

(11) Montague: $a \Rightarrow ((a,b),b)$ / $\lambda y_{(a,b)} . y(x_a)$

(12) if $a \Rightarrow b$ $(\tau_b<x_a>)$, then $(\vec{c},a) \Rightarrow (\vec{c},b)$ / $\lambda \vec{y}_{\vec{c}} . \tau_b[x_a{:=}z_{(\vec{c},a)}(\vec{y})]$
(if $\vec{c}$ is a sequence of types: $c_1,\ldots,c_n$, then $(\vec{c},a) = (c_1,(\ldots,(c_n,a)\ldots))$; if $\vec{y}_{\vec{c}}$ is a sequence of variables $y_1,\ldots,y_n$ of type $\vec{c} = c_1,\ldots,c_n$, then $\lambda \vec{y}_{\vec{c}} . \tau$ stands for $\lambda y_{1c1} \ldots \lambda y_{ncn} . \tau$, and $\tau(\vec{y})$ abbreviates $\tau(y_{1c1})\ldots(y_{ncn})$.)

(13) value raising: $(\vec{a},b) \Rightarrow (\vec{a},((b,c),c))$ / $\lambda \vec{x}_{\vec{a}} \lambda y_{(b,c)} . y(z_{(\vec{a},b)}(\vec{x}))$

(14) argument raising: $(a,(\vec{c},b)) \Rightarrow (((a,b),b),(\vec{c},b))$ /
$\lambda v_{((a,b),b)} \vec{x}_{\vec{c}} . v(\lambda z_a . y_{(a,(\vec{c},b))}(z)(\vec{x}))$

(15) generalized argument raising: $(\vec{a},(b,(\vec{c},d))) \Rightarrow (\vec{a},(((b,d),d),(\vec{c},d)))$ /
$\lambda \vec{x}_{\vec{a}} \lambda v_{((b,d),d)} \lambda \vec{y}_{\vec{c}} . v(\lambda z_b . y_{(\vec{a},(b,(\vec{c},d)))}(\vec{x})(z)(\vec{y}))$

(16) if $a \Rightarrow b$ $(\tau_b<x_a>)$, then $(b,c) \Rightarrow (a,c)$ / $\lambda x_a . y_{(b,c)}(\tau_b)$

(17) generalized argument lowering: $(\vec{a},(((b,c),c),d)) \Rightarrow (\vec{a},(b,d))$ /
$\lambda \vec{x}_{\vec{a}} \lambda y_b . z_{(\vec{a},(((b,c),c),d))}(\vec{x})(\lambda v_{(b,c)} . v(y))$

Notwithstanding its usefulness as a tool for *syntactic* category-‘change’, if this calculus is used as a system for *semantic* type- and translation-‘change’ in the way indicated in section 3, it will appear that the application of some theorems leads to dubious results. From a semantic point of view, the calculus simply is too flexible. Here are some examples.

(a) *Commutativity.*
Consider sentence (18).

(18) $[_S[_{NP}\text{John}][_{VP}[_{TV}\text{loves}][_{NP}\text{Paris}]]]$

If we functionally apply the basic translations of *loves*, *Paris*, and *John*, LOVE$_{(e,(e.t))}$, p_e, and j_e in that (syntactically determined) order, the result is LOVE(p)(j).

It is also possible to apply commutativity to LOVE first, which yields λxλy.LOVE(y)(x), also of type (e,(e,t)). If this is applied to p_e and j_e, the result is LOVE(j)(p). One of the two must be wrong.

(b) *Geach.*
Van Benthem (1986, p. 125) notes that a sentence like

(19) Every crook fears some detectives

"may be evaluated as follows:

Every crook *fears* *some detective[s]*
((e,t),t) (e,(e,t)) ((e,t),t)
((e,t),t) $\underline{(e,(e,t))\quad((e,(e,t)),(e,t))}$ [Geach]
$\underline{((e,t),t)\qquad\qquad(e,t)}$ [FA]
t [FA]

Also, additional readings appear." What readings? For example, (iv), which can be derived through (i), (ii), and (iii).

(i) λP.∃x[DETECTIVE(x)&P(x)]
(ii) λRλy.(λP.∃x[DETECTIVE(x)&P(x)](R(y)))
(iii) λRλy.∃x[DETECTIVE(x)&R(y)(x)]
(iv) ∀z[CROOK(z)→∃x[DETECTIVE(x)&FEAR(z)(x)]]

After Geach, the basic translation of *some detectives*, (i) becomes (ii), which reduces to (iii), and after two applications we will obtain (iv), meaning that for every crook there are some detectives who fear him/her, which is wrong.

(c) *Composition.*
On the basis of the same sentence we can show that composition yields undesirable meanings as well:

Every crook *fears* *some detectives*
((e,t),t) $\underline{(e,(e,t))\quad((e,t),t)}$
$\underline{((e,t),t)\qquad\qquad(e,t)}$ [composition]
t [FA]

(v) $\lambda y.(\lambda P.\exists x[\text{DETECTIVE}(x)\&P(x)](\text{FEAR}(y)))$

(vi) $\lambda y.\exists x[\text{DETECTIVE}(x)\&\text{FEAR}(y)(x)]$

(vii) $\forall z[\text{CROOK}(z)\rightarrow\exists x[\text{DETECTIVE}(x)\&\text{FEAR}(z)(x)]]$ (=iii).

(v), which reduces to (vi), is the result of composition applied to the basic translations of *fears* and *some detectives*. After functional application, the whole sentence gets translation (vii) = (iv) again.

Note that the bad results of applying commutativity in (18) and Geach and composition in (19) are due to the non-directionality of the calculus, which might suggest we could fare better using a directional version of the Lambek calculus. In a directional version one would have:

Every crook	*fears*	*some detectives*	
t/(e\t)	(e\t)/e	(t/e)\t	
t/(e\t)	(e\t)/t	(e\(t/e))\(e\t)	[Geach]
t/(e\t)	$\underline{\text{e\\(t/e)}}$	$\underline{\text{(e\\(t/e))\\(e\\t)}}$	[commutativity]
$\underline{\text{t/(e\\t)}}$		$\underline{\text{(e\\t)}}$	[application]
t			[application]

Here the necessary application of commutativity to *fears* gets the 'θ-roles' right, whereas the scope of the quantifiers remains ∀∃, yielding (vii).

(viii) $\forall z[\text{CROOK}(z)\rightarrow\exists x[\text{DETECTIVE}(x)\&\text{FEAR}(x)(z)]]$

(ix) $\exists x[\text{DETECTIVE}(x)\&\forall z[\text{CROOK}(z)\rightarrow\text{FEAR}(x)(z)]]$

Likewise, the ∃∀-reading (ix) can be obtained in the directional calculus (though in a way that does not respect the constituent structure):

Every crook	*fears*	*some detectives*	
$\underline{\text{t/(e\\t)}}$	$\underline{\text{(e\\t)/e}}$	(t/e)\t	
$\underline{\text{t/e}}$		$\underline{\text{(t/e)\\t}}$	[composition]
	t		[application]]

Therefore, the question might arise: why not account for scope ambiguities by using directional semantic types and the directional Lambek calculus?

Gosse Bouma has studied this question, and his conclusion is that "[i]t seems then, that the Lambek calculus will only be able to predict quantifier-scope ambiguities in a restricted number of cases' (p.9).

And even when it is able to do so –(19) is a case in point–, it is dependent on word order. As Bouma observes, in accounting for scope ambiguities in the directional calculus a crucial role appears to be played by commutativity, a rule that can only be used if we have SVO (or OVS) order. Already in the case of (20), for example, we are able to derive only the $\forall\exists$-reading, unless we encode the 'wide scope-object' $\exists\forall$-reading in the lexical translation of *vreest* (fears): $\lambda T_1 \lambda T_2 . T_1(\lambda x. T_2(\lambda y.\text{FEAR}(x)(y))$ (type: TYP(NP)\(TYP(NP))\t)). (For the intensional reading of (20), a second lexical translation of *vreest* will be needed.)

(20) Het is waar dat elke schurk enkele detectives vreest
It is true that every crook some detectives fears
'It is true that every crook fears some detectives'

The conclusion, then, must be that if we want to account for scope ambiguities, we will have to use a calculus which operates on non-directional types and does not contain commutativity, Geach, and composition as theorems.

5. The quantification calculus

Given the undesirability of commutativity (8), Geach (9) and composition (10), you could try and build a calculus[4] for type change on the basis of the Lambek calculus, e.g.,

(21) axioms:
(I) $a \Rightarrow ((a,b),b) \,/\, \lambda y_{(a,b)}.y(x_a)$ (=11)
(II) $(a,(\vec{c},b)) \Rightarrow (((a,b),b),(\vec{c},b)) \,/\, \lambda v_{((a,b),b)}\vec{x}\vec{c}.v(\lambda z_a.y_{(a,(\vec{c},b))}(z)(\vec{x}))$ (=14)
rules:
(III) if $a \Rightarrow b \,/\, \tau_b\langle x_a\rangle$, then $(\vec{c},a) \Rightarrow (\vec{c},b) \,/\, \lambda\vec{y}\vec{c}.\tau_b[x_a{:=}z_{(\vec{c},a)}(\vec{y})]$ (= 12),
(IV) if $a \Rightarrow b \,/\, \tau_b\langle x_a\rangle$, then $(b,c) \Rightarrow (a,c) \,/\, \lambda x_a.y_{(b,c)}(\tau_b)$ (= 16), and
(V) if $a \Rightarrow b \,/\, \tau_b\langle x_a\rangle$, and $b \Rightarrow c \,/\, \tau_c\langle y_b\rangle$, then $a \Rightarrow c \,/\, \tau_c[x_b{:=}\tau_b]$
(cf. 6(d) and 7(d))

It is easy to see that (21) preserves the theorems (11), (14) (viz., as axioms), (12), (16) (as rules), (13), (15) and (17). (21) is a calculus which rewrites types to types, and thus significantly poorer than the full Lambek-van Benthem calculus. Nonetheless, it is not a very perspicuous calculus: e.g., it is far from obvious whether we have really got rid of (8), (9) and (10). Besides, it is 'slow': if we want to lower, say, the third argument of a type, we must apply (I) and (IV) first, then shortcut (I) and (IV) with the help of (V), then apply (III), and finally (V) again.

In both respects, (22), though weaker than (21), is an improvement on it:

[4] (21) is the calculus I presented in my talk at the Amsterdam workshop.

(22) axioms:

(I) $(\vec{a},b) \Rightarrow (\vec{a},((b,c),c))\ /\ \lambda\vec{x}_{\vec{a}}\lambda y_{(b,c)}.y(z_{(\vec{a},b)}(\vec{x}))$ (=13)

(II) $(\vec{a},(b,(\vec{c},d))) \Rightarrow (\vec{a},(((b,d),d),(\vec{c},d)))\ /$
$\lambda\vec{x}_{\vec{a}}\lambda v_{((b,d),d)}\lambda\vec{y}_{\vec{c}}.v(\lambda z_b.y_{(\vec{a},(b,(\vec{c},d)))}(\vec{x})(z)(\vec{y}))$ (=15)

(III) $(\vec{a},(((b,c),c),d)) \Rightarrow (\vec{a},(b,d))\ /\ \lambda\vec{x}_{\vec{a}}\lambda y_b.z_{(\vec{a},(((b,c),c),d))}(\vec{x})(\lambda v_{(b,c)}.v(y))$
(=17)

rule:

(IV) if $a \Rightarrow b\ /\ \tau_b\langle x_a\rangle$, and $b \Rightarrow c\ /\ \tau_c\langle y_b\rangle$, then $a \Rightarrow c\ /\ \tau_c[x_b{:=}\tau_b]$

But from a semantic point of view, (22) still contains anomalous superfluities. For instance, there are more possibilities for argument-raising the n-th argument of a type, than for argument-raising its (n+1)-th argument. We can (e,t)-argument-raise the first e-argument of (e,(e,t)): (e,(e,t)) $\Rightarrow$ (((e,(e,t)),(e,t)),(e,t)), whereas this is impossible for the second e-argument: (e,(e,t)) $\Rightarrow$ (e,(((e,t),(e,t)),(e,t)),t) is not a theorem. This extra freedom of the first argument appears to serve no useful semantic purpose.

Instead of trying to devise a calculus which preserves as many theorems from the Lambek calculus as possible (while avoiding the harmful (8), (9) and (10)), it is maybe better to develop one which is just strong enough to perform the tasks set in section 2, viz. (A) provide a general framework for the minimal type assignment-approach, and (C) account for structural ambiguities which can be explicated in terms of scope. We claim that the type-changing calculus needed for that purpose only contains t-argument-raisings and t-argument-lowerings:

(23) *The quantification calculus*:

axioms:

(I) value raising
$(\vec{a},b) \Rightarrow (\vec{a},((b,c),c))\ /\ \lambda\vec{x}_{\vec{a}}\lambda y_{(b,c)}.y(z_{(\vec{a},b)}(\vec{x}))$

(II) argument raising
$(\vec{a},(b,(\vec{c},t))) \Rightarrow (\vec{a},(((b,t),t),(\vec{c},t)))\ /$
$\lambda\vec{x}_{\vec{a}}\lambda v_{((b,t),t)}\lambda\vec{y}_{\vec{c}}.v(\lambda z_b.y_{(\vec{a},(b,(\vec{c},t)))}(\vec{x})(z)(\vec{y}))$

(III) argument lowering
$(\vec{a},(((b,t),t),d)) \Rightarrow (\vec{a},(b,d))\ /\ \lambda\vec{x}_{\vec{a}}\lambda y_b.z_{(\vec{a},(((b,t),t),d))}(\vec{x})(\lambda v_{(b,t)}.v(y))$

rule:

(IV) cut:
if $a \Rightarrow b\ /\ \tau_b\langle x_a\rangle$, and $b \Rightarrow c\ /\ \tau_c\langle y_b\rangle$, then $a \Rightarrow c\ /\ \tau_c[x_b{:=}\tau_b]$

This calculus does not have the argument-permuting effects of the Lambek calculus.

Each of its axioms performs some specific semantic task(s).

Value raising can be used to raise (meanings of) argument expressions as a whole: it predicts $\lambda P.P(j)_{((e,t),t)}$ from j_e as a translation for *John*, for example. But it will also be shown to be extremely valuable in the derivation of complex structural ambiguities.

Argument raising and argument lowering provide (meanings of) functor expressions with an amount of flexibility: you can, e.g., raise the first argument of $\text{FIND}_{(e,(e,t))}$, and obtain $\lambda T \lambda y.T(\lambda x.\text{FIND}(x)(y))$, an expression which can be functionally applied to ((e,t),t)-type translations of quantified NPs. On the other hand, you can argument-lower $\text{SEEK}_{(((e,t),t),(e,t))}$, and obtain $\lambda x.\text{SEEK}(\lambda P.P(x))$, an expression which can be functionally applied to the basic translations of proper names.

6. Applications

In this section we will show that the quantification calculus has a number of useful applications. 6.1 deals with the scope of quantification. We will see that the calculus is capable of handling quantifier scope ambiguities, not only in 'flat' functional application structures (6.1.1), but also in embedded functional application structures (6.1.2). In 6.2, we will consider ambiguities involving the scope of coordination.

6.1 The scope of quantification

6.1.1 Flat structures

Consider sentence (24).

(24) John finds Sandy

Let us make the uncontroversial assumptions that the semantic operations to be used in (24) are all functional applications; that the application 2-tuple for the VP is TV,NP; and that the application 2-tuple for the whole sentence is VP,NP. If we functionally apply the basic translations of the lexical expressions, the sentence translates as $\text{FA}(\text{FA}(\text{FIND},s),j) = \text{FIND}(s)(j)$. It can be proven that this is the only reading of type t (henceforth: 'reading') we obtain using the quantification calculus: if arbitrary non-basic translations had been used instead, the result would have been equivalent. (The proof, and proofs of similar claims in the remainder of this paper, can be found in my (in preparation (i))).

If a higher order, 'intensional' TV occurs with two proper names, the quantification calculus also predicts one reading, whatever the translations used. Cf. (25):

(25) John seeks Sandy

We can get this reading, $\text{SEEK}(\lambda P.P(s))(j)$, in two ways. By applying value raising to s_e: $\lambda P.P(s)_{((e,t),t)}$, and taking the basic translation of both *seek*, $\text{SEEK}_{(((e,t),t),(e.t))}$, and *John*, j_e; but also by taking the basic translations of *Sandy* and *John*, and lowering the first argument of SEEK: $\lambda x.\text{SEEK}(\lambda P.P(x))$.

However, if the arguments are quantified NPs, SEEK and FIND behave differently with respect to the number of readings predicted by the calculus.

(26) Every boy finds some girl
- (i) $\forall y[\text{BOY}(y)\rightarrow\exists x[\text{GIRL}(x)\&\text{FIND}(x)(y)]]$
- (ii) $\exists x[\text{GIRL}(x)\&\forall y[\text{BOY}(y)\rightarrow\text{FIND}(x)(y)]]$

(27) Every boy seeks some girl
- (i) $\forall y[\text{BOY}(y)\rightarrow\exists x[\text{GIRL}(x)\&\text{SEEK}(\lambda P.P(x))(y)]]$
- (ii) $\exists x[\text{GIRL}(x)\&\forall y[\text{BOY}(y)\rightarrow\text{SEEK}(\lambda P.P(x))(y)]]$
- (iii) $\forall y[\text{BOY}(y)\rightarrow\text{SEEK}(\lambda P.\exists x[\text{GIRL}(x)\&P(x)])(y)]$

For (26), the calculus has no more than two readings in petto, whereas (27) is assigned three readings: FIND is of type (e,(e,t)), and there are two nonequivalent ways of raising it to type (((e,t),t),((e,t),t),t)) = (T,(T,t)):

(a) $(e,(e,t)) \Rightarrow (T,(e,t))$ / $\lambda T_1'\lambda y'.T_1'(\lambda x.\text{FIND}(x)(y')) = \alpha$,
$(T,(e,t)) \Rightarrow (T,(T,t))$ / $\lambda T_1\lambda T_2.T_2(\lambda y.\alpha(T_1)(y)) \Leftrightarrow$
$\lambda T_1\lambda T_2.T_2(\lambda y.T_1(\lambda x.\text{FIND}(x)(y)))$

(b) $(e,(e,t)) \Rightarrow (e,(T,t))$ / $\lambda x'\lambda T_2'.T_2'(\lambda y.\text{FIND}(x')(y)) = \alpha$,
$(e,(T,t)) \Rightarrow (T,(T,t))$ / $\lambda T_1\lambda T_2.T_1(\lambda x.\alpha(x)(T_2)) \Leftrightarrow$
$\lambda T_1\lambda T_2.T_1(\lambda x.T_2(\lambda y.\text{FIND}(x)(y)))$

If (a) is used, (i) is obtained. (ii) results from using (b). Additional type change provably does not increase the number of readings. In the case of (27), we can let the lowered translation of *seek*, $\lambda x.\text{SEEK}(\lambda P.P(x))$, undergo (a) and (b) as well, obtaining (27) (i) and (ii), respectively. But in addition, it is possible to use the basic translation SEEK, and raise its second argument:$\lambda T_1 \lambda T_2.T_2(\lambda y.\text{SEEK}(T_1)(y))$, which yields the 'intensional' reading (iii).

In general, an n-place extensional relation R of type $(e_1,(\ldots(e_n,t)\ldots))$ plus n quantified (i.e., ((e.t),t)-type) arguments has maximally n! readings of type t (notice that n! is a maximum: the quantified arguments which are raised e-type expressions, for example, do not multiply the number of readings), which is exactly what we need.

6.1.2 Embedded structures

The quantification calculus, poor though it may seem in comparison with the full Lambek-van Benthem calculus, is nonetheless capable of accounting for scope ambiguities in arbitrarily complex functional application structures as well: consider the parse tree of *some man claims that every boy knew that John kissed one girl*:

(28) $[_S[_{NP}\text{some man}][_{VP}[_{PV}\text{claims that}][_S[_{NP}\text{every boy}][_{VP}[_{PV}\text{knew that}]$
$[_S[_{NP}\text{John}]\ [_{VP}[_{TV}\text{kissed}][_{NP}\text{one girl}]]]]]]]$

Assuming the following basic translations:

some man	$\lambda P.\exists x[M(x)\&P(x)]$ of type ((e,t),t),
every boy	$\lambda P.\forall y[B(y)\rightarrow P(x)]$ of type ((e,t),t),
one girl	$\lambda P.\exists! z[G(z)\&P(z)]$ of type ((e,t),t),
kissed	KISS of type (e,(e,t))
claims that	CLAIM of type (t,(e,t)),
knew that	KNOW of type (t,(e,t)), and
John	j of type e,

the quantification calculus assigns (28) exactly the 13 readings we might wish to assign to it:

(29) (a) $\exists x[M(x)\&CLAIM(x,\forall y[B(y)\rightarrow KNOW(y,\exists!z[G(z)\&KISS(j,z)])])]$
(b) $\exists x[M(x)\&\forall y[B(y)\rightarrow CLAIM(x,KNOW(y,\exists!z[G(z)\&KISS(j,z)]))]]$
(c) $\forall y[B(y)\rightarrow\exists x[M(x)\&CLAIM(x,KNOW(y,\exists!z[G(z)\&KISS(j,z)]))]]$
(d) $\exists x[M(x)\&CLAIM(x,\forall y[B(y)\rightarrow\exists!z[G(z)\&KNOW(y,KISS(j,z))]])]$
(e) $\exists x[M(x)\&CLAIM(x,\exists!z[G(z)\&\forall y[B(y)\rightarrow KNOW(y,KISS(j,z))]])]$
(f) $\exists x[M(x)\&\forall y[B(y)\rightarrow CLAIM(x,\exists!z[G(z)\&KNOW(y,KISS(j,z))])]]$
(g) $\forall y[B(y)\rightarrow\exists x[M(x)\&CLAIM(x,\exists!z[G(z)\&KNOW(y,KISS(j,z))])]]$
(h) $\exists x[M(x)\&\forall y[B(y)\rightarrow\exists!z[G(z)\&CLAIM(x,KNOW(y,KISS(j,z)))]]]$
(j) $\forall y[B(y)\rightarrow\exists x[M(x)\&\exists!z[G(z)\&CLAIM(x,KNOW(y,KISS(j,z)))]]]$
(k) $\exists x[M(x)\&\exists!z[G(z)\&\forall y[B(y)\rightarrow CLAIM(x,KNOW(y,KISS(j,z)))]]]$
(l) $\forall y[B(y)\rightarrow\exists!z[G(z)\&\exists x[M(x)\&CLAIM(x,KNOW(y,KISS(j,z)))]]]$
(m) $\exists!z[G(z)\&\exists x[M(x)\&\forall y[B(y)\rightarrow CLAIM(x,KNOW(y,KISS(j,z)))]]]$
(n) $\exists!z[G(z)\&\forall y[B(y)\rightarrow\exists x[M(x)\&CLAIM(x,KNOW(y,KISS(j,z)))]]]$

In order to get a feel for what the calculus does, let us take reading (k) as an example and work it out in full, be it perhaps somewhat complex, detail:

(30) $\exists x[\text{MAN}(x)\&\exists!z[\text{GIRL}(z)\&\forall y[\text{BOY}(y)\rightarrow \text{CLAIM}(x,\text{KNOW}(y,\text{KISS}(j,z)))]]]$

Some man must have the widest scope, therefore we postpone its 'treatment' to the last moment. We start with the quantifier which gets the narrowest scope: *every boy*. The NP *every boy* must have wide scope with respect to *claims that*, so we change the type (t,(e,t)) of *knew that*, its 'own' functor, to (t,(e,((t,(e,t)),(e,t))) with the help of value raising. Translation:

(i) $\lambda t_t\lambda x_e\lambda G_{(t,(e,t))}.G(\text{KNOW}(x,t))$

We then apply argument raising to the second argument, e, of (i): (t,(T,((t,(e,t)),(e,t)))) and get the reduced translation (ii):

(ii) $\lambda t_t\lambda T\lambda G_{(t,(e,t))}\lambda v_e.T(\lambda u.G(v,(\text{KNOW}(u,t)))$

That is essentially enough. The NP *one girl* must have wider scope than *claims that* and *knew that.* We can effectuate this by applying value raising to *kissed*, its 'own' functor, making use of the type (t,(T,((t,(e,t)),(e,t)))) *knew that* has been raised to in (ii): (e,(e,((t,(T,((t,(e,t)),(e,t)))),(T,((t,(e,t)),(e,t)))))) Translation:

(iii) $\lambda x\lambda y\lambda Q_{(t,(T,((t,(e,t)),(e,t))))}.Q(\text{KISS}(y,x))$

(T,(e,((t,(T,((t,(e,t)),(e,t)))),(T,((t,(e,t)),(e,t)))))) is, finally, the result of applying argument raising to the first argument. The translation reduces to (iv):

(iv) $\lambda T_1\lambda y\lambda Q_{(t,(T,((t,(e,t)),(e,t))))}\lambda T_2\lambda G\lambda x.T_1(\lambda w.Q(\text{KISS}(y,w))(T_2)(G)(x))$

This completes the 'treatment' of *one girl*. Apply (iv) to $\lambda P.\exists!z[\text{GIRL}(z)\&P(z)]$ and j, respectively, and reduce:

(v) $\lambda Q_{(t,(T,((t,(e,t)),(e,t))))}\lambda T_2\lambda G\lambda x.\exists!z[\text{GIRL}(z)\&Q(\text{KISS}(j,z))(T_2)(G)(x)]$

This is of type ((t,(T,((t,(e,t)),(e,t)))),(T,((t,(e,t)),(e,t)))). Now, we could apply (v) to (ii), our derived translation of *knew that*, but since the syntactically determined 2-tuple is PV,S, PV has to be the functor. Therefore, we raise (ii) as a whole to the type ((TYP(v),(T,((t,(e,t)),(e,t)))),(T,((t,(e,t)),(e,t)))), obtaining (vi):

(vi) $\lambda W_{(\text{TYP}(v),(T,((t,(e,t)),(e,t))))}.W(\lambda t_t\lambda T\lambda G\lambda v_e.T(\lambda u.G(v,\text{KNOW}(u,t))))$

(vi) applied to (v) yields (vii) (slightly reduced), which reduces further to (viii):

(vii) $\lambda Q\lambda T_2\lambda G\lambda x.\exists!z[\text{GIRL}(z)\&Q(\text{KISS}(j,z))(T_2)(G)(x)]$
$(\lambda t\lambda T\lambda G\lambda v.T(\lambda u.G(v,\text{KNOW}(u,t))))$

(viii) $\lambda T_2\lambda G\lambda x.\exists!z[\text{GIRL}(z)\&T_2(\lambda u.G(x,\text{KNOW}(u,\text{KISS}(j,z))))]$

(viii) applied to $\lambda P.\forall y[\text{BOY}(y)\rightarrow P(y)]$ reduces to (ix):

(ix) $\lambda G\lambda x.\exists!z[\text{GIRL}(z)\&\forall y[\text{BOY}(y)\rightarrow G(x,\text{KNOW}(y,\text{KISS}(j,z)))]]$

(ix) is of type ((t,(e.t)),(e,t)), so if CLAIM of type (t,(e,t)) is to apply to it, it has to be blown up to functor-size:

(x) $\lambda B_{((t,(e,t)),(e,t))}.B(\text{CLAIM})$

Functional application of (x) to (ix) reduces to (xi):

(xi) $\lambda x.\exists!z[\text{GIRL}(z)\&\forall y[\text{BOY}(y)\rightarrow\text{CLAIM}(x,\text{KNOW}(y,\text{KISS}(j,z)))]]$

(xi) is of type (e.t). We blow it up to (xii) of type (((e,t),t),t), and apply this to $\lambda P.\exists x[\text{MAN}(x)\&P(x)]$ of type ((e,t),t), with the reduced result (xiii):

(xii) $\lambda T.T(\lambda x.\exists!z[\text{GIRL}(z)\&\forall y[\text{BOY}(y)\rightarrow\text{CLAIM}(x,\text{KNOW}(y,\text{KNOW}(z)(j)))]])$

(xiii) $\exists x[\mathrm{MAN}(x)\&\exists!z[\mathrm{GIRL}(z)\&\forall y[\mathrm{BOY}(y)\rightarrow \mathrm{CLAIM}(x,\mathrm{KNOW}(y,\mathrm{KISS}(z)(j)))]]$

This completes our derivation of reading (k).

6.2 The scope of coordination

The structural ambiguities involving the scope of coordination that were studied in Partee and Rooth (1983) can also be accounted for with the quantification calculus. For instance, cf. (31):

(31) $[_S[_{NP}\text{John}][_{VP}[_{TV}\text{caught and ate}][_{NP}\text{a fish}]]]$
  (i) $\exists x[\mathrm{FISH}(x)\&\mathrm{CATCH}(j,x)\&\mathrm{EAT}(j,x)]$
  (ii) $\exists x[\mathrm{FISH}(x)\&\mathrm{CATCH}(j.x)]\&\exists x[\mathrm{FISH}(x)\&\mathrm{EAT}(x)]$

The calculus assigns two readings to (31). We can apply generalized conjunction to CATCH and EAT: $\lambda y\lambda x.[\mathrm{CATCH}(x,y)\&\mathrm{EAT}(x,y)]_{(e,(e.t))}$, and then raise the first argument of the conjunction: $\lambda T\lambda x.T(\lambda y.[\mathrm{CATCH}(x,y)\&\mathrm{EAT}(x,y)])$. This leads to (i). But we are also able to start with raising the first argument of both verbs separately, $\lambda T\lambda x.T(\lambda y.\mathrm{CATCH}(x,y))$ and $\lambda T\lambda x.T(\lambda y.\mathrm{EAT}(x,y))$, and apply generalized conjunction afterwards: in that event, the result reduces to the formula $\lambda T\lambda x.[T(\lambda y.\mathrm{CATCH}(x,y))\&T(\lambda y.\mathrm{EAT}(x,y))]$, and combination with *a fish* and *John* yields (ii).

Partee and Rooth in fact only accept (i), "unless the sentence(...) [is] given a very marked intonation or the context is heavily loaded" (p.365), but this is something which I fail to be able to agree with, considering the completely natural continuation in (32):

(32) John caught and ate a fish. The fish he caught was inedible, and the fish he ate caught his eye.

Be this as it may, less probable but possible readings should be predicted by the grammar too, preferably together with an explanation of their lesser probability. The explanation of this case will be given in footnote 5, below.

The *de dicto* 'wide scope *or*'-reading of (5), repeated below as (33), can be obtained in the quantification calculus:

(33) John is looking for a fish or a bike

We can apply (e,t)-value raising to $\lambda P.\exists x[\text{FISH}(x)\&P(x)]$ and $\lambda P.\exists x[\text{BIKE}(x)\&P(x)]$: which results in $\lambda Q_{(((e,t),t),(e,t))}.Q(\lambda P.\exists x[\text{FISH}(x)\&P(x)])$ and $\lambda Q.Q(\lambda P.\exists x[\text{BIKE}(x)\&P(x)])$, and subsequently take the (reduced) generalized disjunction of the raised translations:

(34) $\lambda Q\lambda y.[Q(\lambda P.\exists x[\text{FISH}(x)\&P(x)]\ )(y)\vee Q(\lambda P.\exists x[\text{BIKE}(x)\&P(x)]\ )(y)]$

The functional application of the raised $\lambda W_{((((e,t),t),(e,t)),(e,t))}W(\text{LOOKF})$ to (34) reduces to (35):

(35) $\lambda y.[\text{LOOKF}(\lambda P.\exists x[\text{FISH}(x)\&P(x)]\ )(y)\vee \text{LOOKF}(\lambda P.\exists x[\text{BIKE}(x)\&P(x)]\ )(y)]$

And (35) applied to j yields (36),

(36) $\text{LOOKF}(j,\lambda P.\exists x[\text{FISH}(x)\&P(x)])\vee\text{LOOKF}(j,\lambda P.\exists x[\text{BIKE}(x)\&P(x)])$

which is what we are looking for.

More interestingly, the calculus also supplies a desired reading, viz., (38), for (37), a problematic case for which Partee and Rooth, quite surprisingly, suggest "a rule quantifying in terms of a higher type" (p.376):

(37) John believes that a fish or a bike swam in the canal

(38) $\text{BELIEVE}(j,\exists x[\text{FISH}(x)\&\text{SWIM}(x)])\vee\text{BELIEVE}(j,\exists x[\text{BIKE}(x)\&\text{SWIM}(x)])$

To get (38), we must first raise the NPs to $\alpha = ((e,t),((t,(e,t)),(e,t))$, and then apply generalized disjunction:

(39) $\lambda P\lambda G\lambda y.[G(\exists x[\text{FISH}(x)\&P(x)])(y)\vee G(\exists x[\text{BIKE}(x)\&P(x)])(y)]$

We also raise $\text{SWIM}_{(e,t)}$, to $(\alpha,((t,(e,t)),(e,t)))$: $\lambda Y_{\alpha}.Y(\text{SWIM})$, and apply this to (39):

(40) $\lambda G\lambda y.[G(\exists x[\text{FISH}(x)\&\text{SWIM}(x)])(y)\vee G(\exists x[\text{BIKE}(x)\&\text{SWIM}(x)])(y)]$

Finally, raise BELIEVE to $(((t,(e,t)),(e,t)),(e,t))$, and apply the result to (40) and j.

In our system we get all possible coordination scopes. For example, (41) is assigned the readings (i), (ii) and (iii):

(41) Mary said that every student lost or won

(i) $\text{SAY}(m,\forall x[\text{STUD}(x)\rightarrow[\text{LOOSE}(x)\vee\text{WIN}(x)]])$

(ii) $\text{SAY}(m,\forall x[\text{STUD}(x)\rightarrow\text{LOOSE}(x)]\vee\forall x[\text{STUD}(x)\rightarrow\text{WIN}(x)])$

(iii) $\text{SAY}(m,\forall x[\text{STUD}(x)\rightarrow\text{LOOSE}(x)])\vee\ \text{SAY}(m,\forall x[\text{STUD}(x)\rightarrow\text{WIN}(x)])$

"While intuitions are far from clear" (p.376), only (i) and (iii) are considered okay by Partee and Rooth. A similar case is (42):

(42) Every student failed or got a D

This sentence does not seem to have the reading 'every student failed or every student got a D', a reading that is predicted by all available theories of type ambiguity, including ours: apply t-raising and generalized disjunction to the verbs, and the result is $\lambda T.[T(\text{FAIL})\vee T(\text{GET-D})]$, which after application to $\lambda P.\forall x[\text{STUDENT}(x)\rightarrow P(x)]$ gets us the undesirable reading.

Groenendijk and Stokhof (1987) suggest that we exclude this reading by requiring that only argument raisings be applied to FAIL and GET-D (motivating this by stipulating that the syntactic function/argument structure should be respected). But this does not work, since in the situation we are considering, argument raising and value raising yield equivalent translations: $\lambda T.T(\lambda x.\text{FAIL}(x))$ (argument raising), and $\lambda T.T(\text{FAIL})$ (value raising).

However, it is obvious that the data are far from clear. Consider sentence (43):

(43) Every player of our team is wearing a red shirt or a green shirt

That this sentence does have the reading 'every player of our team is wearing a red shirt or every player of our team is wearing a green shirt', can be supported by the possibility (if we add a 'heavily loaded' context: the speaker is colour-blind) of the continuation '...but I can't tell you what the exact colour is'.

That the context must be heavily loaded is what we should expect. For, under normal circumstances the Gricean maxim 'avoid ambiguity' requires us not to utter (41), (42) or (43) to convey the information expressed by the 'every ... or every ...'-readings: there are unambiguous sentences that express them[5] :

(44) Mary said that every student lost or every student won

[5] The same pragmatic explanation applies to (31) (ii): there is an unambiguous sentence available for expressing this reading: *John caught a fish and ate a fish*

(45) Every student failed or every student got a D

(46) Every player of our team is wearing a red shirt or every player of our team is wearing a green shirt

Only a colour-blind person entangled in the cumbersomeness of his or her disjunctive world-view may be expected to let being short prevail over being unambiguous in cases like these.

7. Conclusion

We have seen that the combination of a (type-driven) translation procedure and a theory of type ambiguity based on the quantification calculus yields an adequate semantic theory of quantification and coordination.

Moreover, the quantification calculus is a perspicuous system. As a consequence, it is not difficult to prove that it is not only complete with respect to the wished-for scope ambiguities in the examples we have been considering, but also correct: it can be shown that these readings are the only ones given by the calculus. This is done in my (in preparation (i)). Besides, from the proofs given there, it emerges that within the quantification calculus, one of the disastrous aspects of type-shifting –the chaotic ensemble of possible ways to obtain a reading– seems to be open to systematization by means of an orderly procedure which yields the required reading with a predictable amount of effort.

Finally, we have kept the type theory extensional for practical reasons. First, it was done for expository purposes: extensionality helped to prevent the presentation from being more complicated than necessary. Second, omitting s's facilitated comparison with the Lambek calculus. Van Benthem (1986, pp.147-150) has discussed the options available for intensionalizing this calculus, and in view of the extremely tentative nature of this discussion, a straightforward introduction of intensional types into the calculus seems to be a problematic matter. However, this does not necessarily hold for the quantification calculus. Since the axioms of this calculus each perform some specific semantic task(s), it is quite obvious what a quantification calculus for intensional type change should look like (cf. my (in preparation (ii))).

Acknowledgements

I would like to thank Renate Bartsch, Johan van Benthem, Gosse Bouma, Wojciech Buszkowski, Jeroen Groenendijk and Martin Stokhof for corrections, stimulation, discussion, comment, criticism and elucidation, respectively. The research for this paper was supported by the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

References

Benthem, J. van: 1986, *Essays on logical semantics*, Dordrecht: Reidel.

Benthem, J. van: 1987, 'Semantic type change and syntactic recognition', University of Amsterdam: Department of Mathematics report 87-05.

Bouma, G.: 1986, 'Lambek semantics', ms.

Cooper, R.: 1983, *Quantification and syntactic theory*, Dordrecht: Reidel.

Gazdar, G.: 1980, 'A cross-categorial semantics for coordination', *Linguistics and Philosophy* 3, 407-410.

Gazdar, G., E. Klein, G. Pullum, and I. Sag: 1985, *Generalized phrase structure grammar*, Oxford Basil Blackwell.

Groenendijk, J., and M. Stokhof: 1984, *Studies on the semantics of questions and the pragmatics of answers*, University of Amsterdam disssertation.

Groenendijk, J., and M. Stokhof: 1987, 'Type-shifting principles and the semantics of interrogatives', University of Amsterdam: ITLI Prepublication Series 87-01.

Hendriks, H.: in preparation, 'Flexibilization, so far so good? (i) Semantic properties of the quantification calculus; (ii) A calculus for intensional type change' (working title).

Janssen, T.: 1983, *Foundations and applications of Montague grammar*, University of Amsterdam dissertation.

Keenan, E., and L. Faltz: 1985, *Boolean semantics for natural language*, Dordrecht: Reidel.

Klein, E., and I. Sag: 1984, 'Type-driven translation', *Linguistics and Philosophy* 8, 163-202.

Lambek, J.: 1958, 'The mathematics of sentence structure', *American Mathematical Monthly* 65, 154-169.

Landman, F., and I. Moerdijk: 1983, 'Compositionality and the analysis of anaphora', *Linguistics and Philosophy* 6, 89-114.

Montague, R.: 1974, 'The proper treatment of quantification in ordinary English', in R. Thomason (ed.), *Formal philosophy*. New Haven: Yale University Press.

Partee, B.: 1986, 'Noun phrase interpretation and type-shifting principles', in J. Groenendijk, D. de Jongh, and M. Stokhof (eds.), *Studies in discourse representation theory and the theory of generalized quantifiers*, Dordrecht: Foris.

Partee, B., and M. Rooth: 1983, 'Generalized conjunction and type ambiguity', in R. Bäuerle, C. Schwarze, and A. von Stechow (eds.), *Meaning, use and interpretation of language*, Berlin: de Gruyter.