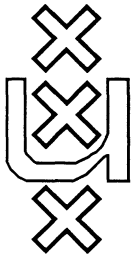
Institute for Language, Logic and Information

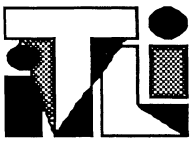
CATEGORIAL GRAMMAR AND TYPE THEORY

Johan van Benthem

ITLI Prepublication Series 87-07



University of Amsterdam



Institute for Language, Logic and Information
Instituut voor Taal, Logica en Informatie

CATEGORIAL GRAMMAR AND TYPE THEORY

Johan van Benthem
Department of Mathematics and Computer Science
University of Amsterdam

Received October 1987

to appear in *Linguistics and Philosophy*

Correspondence to:

Faculteit der Wiskunde en Informatica
(Department of Mathematics and Computer Science)
Roetersstraat 15
1018WB Amsterdam

or

Faculteit der Wijsbegeerte
(Department of Philosophy)
Grimburgwal 10
1012GA Amsterdam

1. Reunion of Relatives

Categorial Grammar and the Theory of Types are two channels of the same stream, originating in Frege's and Husserl's ideas on categorial structure in human thought. One has become a linguistic paradigm in the hands of Ajdukiewicz, Bar-Hillel and later authors, the other has been a standard logical approach ever since Russell's work in the foundations of mathematics. The two can still be viewed with profit as being complementary, as will be illustrated in this paper. In mathematical logic, 'theory of types' denotes a certain kind of approach, rather than one single monolithic system: our perspective on 'categorial grammar' in linguistics is in the same spirit.

The basic idea in Categorial Grammar has been to associate syntactic *categories* of expression with semantic *types*, in such a way that syntactic construction mirrors semantic combination. In this paper, we shall use Montagovian types, generated as follows:

e and t are basic types (*e*ntity, *t*ruth value);
if a and b are types, then so is (a,b).

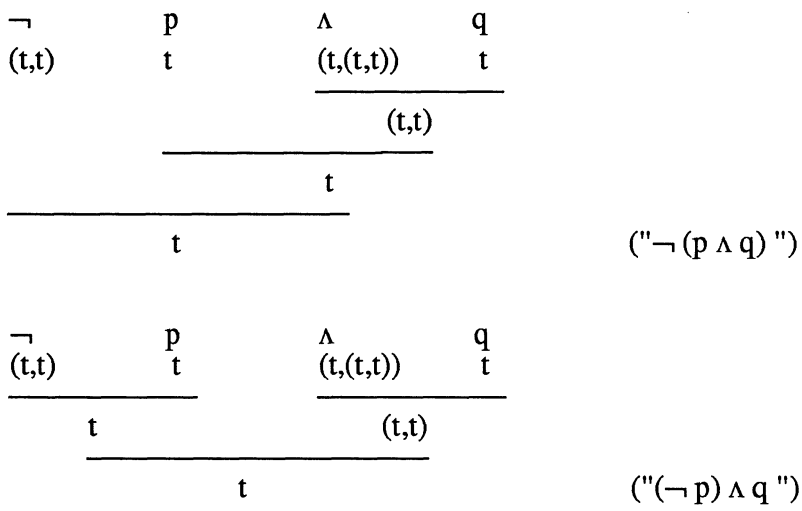
Later on, another basic type s (for *s*ense, or *s*ituation) will be considered too. Semantically, e stands for the individual objects, t for the truth values, and (a,b) for *functions* taking a-objects to b-objects. This gives us such familiar correspondence as:

category	*type*
proper name	e
intransitive verb	(e,t)
transitive verb	(e,(e,t))
(complex) noun phrase	((e,t),t)
adverb	((e,t),(e,t))
binary connective	(t,(t,t))

Categories need not correspond one-to-one with types. This slack enables us to model certain semantic similarities between different categories of expression; witness

common noun	(e,t)
adjective	((e,t),(e,t))

The mechanism of categorial combination may be illustrated by the following standard example, which gives two scope readings for a propositional formula:



Note how meanings are built up through successive functional application.

To get a closer fit with the syntax of natural language, where restrictions on order are (usually) important, categorial grammarians have also introduced *directed* slashes:

a\b (a 'left-searching' functor)
b/a (a 'right-searching' functor)

Despite its descriptive, combinatorial importance, this refinement will play only a marginal rôle in what follows.

In the initial phase of Categorial Grammar, its envisaged use consisted in assigning one or more types to basic expressions ('words'), so as to make the computed combinations match actual grammatical strings. Over the last decade, however, various proposals have been made for adding a mechanism of *type changing*, or alternatively, more liberal modes of *type combination*. There are many kinds of motivation for this move. For instance, Geach 1972 used it to account for the *polymorphism* of negation, introducing his recursive rule:

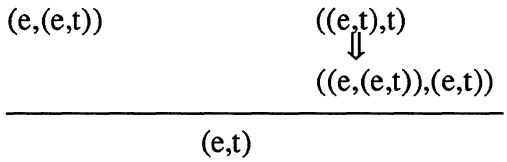
an expression occurring in type (a,b)
may also occur in type ((c,a), (c,b)) (for any c).

This rule raises the basic type (t,t) for negation to ((e,t), (e,t)) (intransitive verb negation), ((e,(e,t)),(e,(e,t))) (transitive verb negation), etcetera. But the same move also accounted for quite different facts, such as the notorious difficulty in getting transitive verbs to accept complex noun phrase objects:

(e,(e,t)) ((e,t),t)
?

[no function application is possible]

Geach's solution is to let the direct object expression 'adapt to context':



Alternatively, this transition may be described as allowing *Composition* (in addition to Application) of functions:

(e,(e,t)) ((e,t),t) ⇒ (e,t)
---- ----

Many other rules of type change have been proposed since in syntax and morphology. These concern such diverse purposes as describing coordination, morphological argument inheritance, parasitic gaps, etcetera - with important contributions by Bach 1984, Steedman 1985, Moortgat, 1984, and many others.Here is a sample:

e ⇒ ((e,t),t) ('type raising'; Montague 1974)
and more generally
a ⇒ ((a,b),b)

(((e,t),t),t) ⇒ (e,t) ('argument lowering'; Partee & Rooth1983)
and more generally
(((a,b),b)),c) ⇒ (a,c)

For a fuller survey of recent developments, see Bach, Oehrle & Wheeler, eds, 1987; Buszkowski, Marciszewski & van Benthem, eds, 1987, and Klein & van Benthem, eds, 1987.

To illustrate the flexibility of these approaches, here is one more example. Let a prepositional phrase ("to Palo Alto") have the adverbial type ((e,t),(e,t)). (In general, prepositions by themselves will have type (((e,t),t),((e,t),(e,t))) to accomplish this.) This gives us a classical analysis for combination with intransitive verbs:

"walk to Palo Alto"
(e,(e,t)) ((e,t),(e,t)) ⇒ (e,t)

But, how to *drive* to Palo Alto? For this purpose, the Geach rule may be used, adapting adverbials to transitive verbs:

"drive to Palo Alto"
(e,(e,t)) ((e,t),(e,t)) ⇒ (e,(e,t))

And finally, to take another well-known issue in standard Categorial Grammar, stacking of prepositional phrases is easy too:

"from Los Altos to Palo Alto"
((e,t),(e,t)) ((e,t),(e,t)) ⇒ ((e,t),(e,t))

In view of all this permissiveness, it may be appropriate to point at some type changes that are definitely not admissible. One example is a would-be converse to the Partee & Rooth rule:

(a,c) ⇒ (((a,b),b),c) ('argument raising').

Some special cases of this rule are in fact admissible, for various reasons. (E.g., (e,t) ⇒ (((e,t),t),t) by the Montague Rule. Cf. also van Benthem 1987c on a proposal by Groenendijk & Stokhof 1987b.) But the general schema is invalid; in a sense to be explained now.

2. A Logical Perspective

A more systematic perspective upon the various proposals made for adopting type changes may be found in Logic. As it happens, there is a strong formal analogy between *function* types and logical *implications:*

(a,b) a→b

This analogy was first exploited for linguistic purposes in Lambek 1958, who set up a calculus of sequents

X ⇒ b
('type sequence X combines to/derives type b')

The idea here is that type combination is very much like derivation in an *implicational logic*. ('Parsing as Deduction'!) Thus, one can set up a system of logical axioms and rules for categorial combination. Here, we shall present one of the many possible formats:

Axiom : $a \Rightarrow a$

Rules :

$$\frac{X\ b\ Y \Rightarrow c}{X\ a\ (a,b)\ Y \Rightarrow c} \qquad \textit{(Modus Ponens)}$$

$$\frac{X\ a \Rightarrow b}{X \Rightarrow (a,b)} \qquad \textit{(Conditionalization)}$$

$$\frac{X\ a \Rightarrow b \qquad Y \Rightarrow a}{X\ Y \Rightarrow b} \qquad \textit{(Cut)}$$

These rules produce derivations for the earlier proposals.

Example ('Geach'):

$$\cfrac{\cfrac{\cfrac{\cfrac{t \Rightarrow t}{(t,t)\ t \Rightarrow t}}{(t,t)\ (e,t)\ e \Rightarrow t}}{(t,t)\ (e,t) \Rightarrow (e,t)}}{(t,t) \Rightarrow ((e,t),(e,t))}$$

In practice, a *Natural Deduction* format is much easier to use (see van Benthem 1986a, chapter 7). But, we retain the present approach here for its theoretical perspicuity.

These were the *logical rules* of the Lambek system. But, there is another degree of freedom in setting up such logical calculi, being the so-called *structural rules* for handling premises. For instance, the following conventions would turn the above set of rules into the standard Intuitionistic Implicational Logic:

Permutation (P) :
if $X \Rightarrow b$, then $\pi[X] \Rightarrow b$,
for any permutation π of the premises X

Contraction (C) :
if $X\ a\ a \Rightarrow b$, then $X\ a \Rightarrow b$

Monotonicity (M) :
if $X \Rightarrow b$, then $X\ a \Rightarrow b$.

Note: the full formulation of C and M should be more complex in the absence of Permutation.

Now, what distinguishes the calculus of type change from implicational logic is precisely which of these structural rules (if any) are admissible. Without them, the calculus records full information about which *occurrences of* which premises were used in what order. With them, certain identifications become possible (which may be described alternatively as 'transformations' on proof trees : copying, deletion, etcetera).

Example (Conjunction). To derive polymorphic types for "and", Contraction and Permutation are to be invoked. An ordinary deduction produces

$(t,(t,t))\ e\ (e,t)\ e\ (e,t) \Rightarrow t$
and then,
$(t,(t,t))\ (e,t)\ e\ e \Rightarrow t$ (Permutation)
$(t,(t,t))\ (e,t)\ (e,t)\ e \Rightarrow t$ (Contraction)
and by successive uses of Conditionalization,
$(t,(t,t)) \Rightarrow ((e,t),((e,t),(e,t)))$.

Of course, some structural rules would produce rather implausible kinds of type change; such as Monotonicity:

$$\frac{t \Rightarrow t}{t\ e \Rightarrow t}$$
$t \Rightarrow (e,t)$: can sentences become intransitive verbs?

In any case, the point here is not to take a stand on such cases, but rather to show how they become amenable to systematic discussion. Another example of an inadmissible transition is the earlier-mentioned case $(a,c) \Rightarrow (((a,b),b),c)$. The reason is simply that $a \rightarrow c / ((a \rightarrow b) \rightarrow b) \rightarrow c$ is not a valid law of implication, neither intuitionistically nor even classically.

Thus, below what is often regarded as the weakest natural implicational logic, there lies a whole new wonderland; with a *Categorial Hierarchy* extending from the basic Ajdukiewicz calculus to that of Lambek, and then, via various systems of 'relevant logic' to the full intuitionistic system.

There are various possible linguistic uses for this hierarchy. One can experiment with weaker and stronger systems for syntactic purposes, admitting ever further constituent structures. (Some of the stronger calculi might also be used, precisely because they 'over-generate', to explain our ability to still make sense of slightly defective syntactic material - as with local permutations in children's discourse.) One can also exploit the earlier-mentioned slack between categories and types here, being stricter as to syntactic combination and having a more liberal calculus producing semantic readings. And of course, as has been observed by Oehrle, Zwarts, Szabolcsi and others, the present vista offers a lot of parametric variation for describing natural languages, and formulating significant *linguistic universals* based on human logical-combinatorial ability.

To conclude, here are a few additional remarks.

- Lambek also considered *concatenation* of expressions, with the following logical analogy:

product type	$a \bullet b$	*conjunction*	$a \wedge b$

The relevant derivation rules for sequents become:

$$\frac{X \Rightarrow a \quad Y \Rightarrow b}{X\ Y \Rightarrow a \bullet b} \qquad \frac{X\ a\ b\ Y \Rightarrow c}{X\ a \bullet b\ Y \Rightarrow c}$$

- There are also *directed* versions of all these calculi, with the earlier slashes, involving rules such as:

$$\frac{X\ b\ Y \Rightarrow c}{X\ a\ a\backslash b\ Y \Rightarrow c} \qquad \frac{X\ a \Rightarrow b}{X \Rightarrow b/a}$$

$$\frac{X\ b\ Y \Rightarrow c}{X\ b/a\ a\ Y \Rightarrow c} \qquad \frac{a\ X \Rightarrow b}{X \Rightarrow a\backslash b}$$

Again, the latter will be more fundamental in detailed syntactic description.

- These 'subterranean' implicational logics may have an interest for Logic itself after all. For, many of the above structural rules have been challenged for logics too: notably Monotonicity, but also Permutation (cf. Groenendijk & Stokhof 1987a). In such a setting, even two 'directed implications' ("if A,B" versus "B, if A") start making sense (cf. van Benthem 1988).

3. Syntax and Proof Theory

Bringing together logical and linguistic perspectives as in the preceding Sections has interesting consequences. Notably, questions of theoretical Linguistics merge with questions of Logic; and tools from both sides may find mutual application. For instance, calculi of sequents have been studied extensively in logical *Proof Theory*, and grammars in *Mathematical Linguistics*. In the realm of Categorial Grammar, these two research traditions meet.

One early example of this phenomenon is Lambek's proof that derivability in his basic directional calculus is *decidable*; for which he used a Gentzen-type Cut Elimination argument from Proof Theory. But, linguistic applications also suggest many aspects of proof structure not yet considered by proof theorists. For instance, the following simple question arises in the study of coordination of linguistic expressions:

given any two type sequences X,Y,
when is there a single type a such that
$X \Rightarrow a$ and $Y \Rightarrow a$ are L-derivable?

It is an open question if this problem is decidable for the basic Lambek calculus.

Much research has been devoted to questions of *recognizing power* for various more flexible categorial calculi. See Buszkowski 1982 for an extensive study, as well as Friedman and Venkatesan 1986 for some results on weak fragments of the Lambek system. The main question has been around since the sixties already, being 'Chomsky's Conjecture':

Does L recognize only the context-free languages?

With one single slash, the answer is positive (Buszkowski). For non-directed systems, some results have been obtained by standard proof-theoretic techniques (van Benthem 1987c, van Benthem 1987e), analyzing 'normal forms' of derivations:

- L + P recognizes all permutation closures of context-free languages (these include non-context-free ones!)
- L + C recognizes only regular languages.

A new impetus in syntactic studies derives from the current interest in *parsing* with categorial grammars (see Moortgat 1987, Pareschi and Steedman 1987). Parsers have to build in various devices for reducing the search spaces encountered in searching for logical proofs (cf. Moortgat's use of 'count' invariants, from van Benthem 1986a), which themselves generate interesting proof-theoretical questions. For instance, the Moortgat parser uses a modicum of 'variable polymorphism': conjunction gets introduced with a variable type (x,(x,x)) , which is instantiated to some specific type a in a suitable environment a a. Again, the *decidability* of a Lambek calculus allowing such polymorphic typing is still an open question.

Another interesting question in this connection concerns the relation between directed and non-directed Lambek calculi. Moortgat 1985 uses the directed L-calculus, with two slashes \ and /. But, at certain places, he needs to recognize 'local permutations', which calls for type changes such as

$((a\backslash b)/c) \Rightarrow (a\backslash(b/c))$.

It turns out that allowing this, or apparently more modest principles of permutation tends to collapse the directed calculus into the non-directed one presented above. Thus, one central question right now is:

Are there any useful but principled bounds on
permutation in categorial calculi?

The variety of questions raised here may have shown how Categorial Grammar can serve as a source of problems enriching the traditional fund of Proof Theory.

4. Semantics and Type Theory

In order to find one's way in the Categorial Hierarchy, it would be useful to have a more *semantic* view of admissible type changes, supplementing the earlier proof-theoretic considerations. One way of approaching this issue is by establishing completeness theorems for various categorial calculi with respect to some suitable generalization of intuitionistic semantics. Results of this kind have been obtained by Došen 1986, Buszkowski 1986. Another way, more relevant here, is to study meanings of specific categorial derivations ('readings'). For this purpose, a general logical analogy may be invoked: there exists an effective correspondence between *categorial derivations* and *type-theoretical terms* describing denotations.

- *The Basic Semantical Correspondence*

In the Ajdukiewicz calculus, the only semantic operation involved is *functional application*, as in the following example (cf. Section 1):

$$\dfrac{\dfrac{\dfrac{(t,t) \quad t}{t}}{t} \qquad \dfrac{\dfrac{(t,(t,t) \quad t}{(t,t)}}{t}$$

$$\dfrac{\dfrac{x_{(t,t)} \quad y_t}{x(y)} \qquad \dfrac{z_{(t,(t,t))} \quad u_t}{z(u)}}{z(u)\,(x(y))}$$

In general, a derivation for a Lambek sequent $X \Rightarrow b$ is encoded in a term

$$\tau_b[\{u_x \mid x \in X\}] .$$

τ_b can be viewed as a 'recipe' for computing an object of type b from given objects of the types x occurring in X. The new feature is now that instances of Conditionalization are encoded by means of *lambda abstraction*..

Example. The 'canonical meaning' of the Geach rule may be read off from the earlier derivation (Section 2):

$$\dfrac{\dfrac{\dfrac{\dfrac{x_t}{y_{(t,t)}\,(z_t)}}{y_{(t,t)}\,(u_{(e,t)}\,(v_e))}}{\lambda v_e \bullet\ y\,(u\,(v))}}{\lambda u_{(e,t)} \bullet \lambda v_e \bullet\ y\,(u\,(v))}$$

Thus, we are giving a precise logical procedure for effecting so-called 'type-driven translation'.

One telling illustration of this perspective is its use in 'deflating' complex semantic truth conditions. A classical example is Montague's analysis of the verb "be", which has been a sign-post to formalism for so many students. In its extensional version, "be" denotes

$$\lambda x_{((e,t),t)} \bullet \lambda y_{e} \bullet \text{'}\{y\} \in x\text{'}.$$

But, this is precisely what would be obtained by letting an ordinary transitive verb in type (e,(e,t)) (here, simple identity between individuals) undergo the earlier-mentioned Geach change for direct objects:

$$\frac{(e,(e,t)) \qquad ((e,t),t) \quad \Rightarrow \quad (e,t)}{(e,(e,t)) \quad \Rightarrow \quad (((e,t),t),\ (e,t))}$$

$$\frac{\lambda y_{e} \bullet x_{((e,t),t)}\ (v_{(e,(e,t))}\ (y))}{\lambda x_{((e,t),t)} \bullet \lambda y_{e} \bullet \ x\ (v_{(e,(e,t))}\ (y))\ \text{('x holds of (being y)')}}$$

Thus, Montague's celebrated uniform treatment of "be" for proper names and complex noun phrases is an automatic result of our semantics for type change applied to ordinary identity.

The precise correspondence employed above may be found in van Benthem 1986a (chapter 7). Various categorial calculi will have derivations encoded in different *fragments* of the full lambda/application language. Notably, terms for the basic Lambek calculus have the following two characteristic properties:

- each operator λ binds exactly one free variable occurrence,
- each subterm has at least one free variable.

With further structural rules, other semantic patterns become admissible. For instance, the term associated with the earlier LPC-derivation for raising sentence conjunction to verb conjunction, will have one 'double bind':

$$\lambda x_{(e,t)} \bullet \lambda y_{(e,t)} \bullet \lambda z_{e} \bullet u_{(t,(t,t))}\ (x(\underline{z}))\ (y(\underline{z}))$$

Another way of viewing what is going on here emphasizes its 'dynamic' potential. By providing lambda terms for various non-sentential expressions, we can build up sentence meanings progressively in some processing order (say, from left to right), thus providing an *incremental semantics.*

Finally, the basic semantic correspondence is easily extended to *product categories.* It suffices to enrich the type-theoretic language with operations of product on types (a•b describes the Cartesian product of 'a-objects cross b-objects') and *pairing* plus *projections* in the language.

Example. Two derivations with product rules:

$$\frac{(a,(b,c)) \quad a \quad b \ \Rightarrow \ c}{(a,(b,c)) \quad a \bullet b \ \Rightarrow \ c}$$
$$\frac{(a,(b,c)) \ \Rightarrow \ (a \bullet b,\ c)}{}$$

$$\frac{x_{(a,(b,c))}\ (y_{a})\ (z_{b})}{x_{(a,(b,c))}\ (\pi_{L}(v_{a \bullet b}))\ (\pi_{R}(v_{a \bullet b}))}$$
$$\frac{}{\lambda v_{a \bullet b} \bullet x_{(a,(b,c))}\ (\pi_{L}(v))\ (\pi_{R}(v))}$$

$$\frac{e \Rightarrow ((e,t),t) \quad ((e,t),t) \Rightarrow ((e,t),t)}{e\ ((e,t),t) \Rightarrow ((e,t),t) \bullet ((e,t),t)} \qquad \frac{\lambda x_{(e,t)} \bullet x(u_e) \qquad y_{((e,t),t)}}{< \lambda x_{(e,t)} \bullet x(u_e),\ y_{((e,t),t)} >}$$

• *Practical Questions*

Even with the above general semantics, many questions remain for practical application. For instance, there may be an over-abundance of readings which is to be curbed. ("Every boy loves a girl" gets four different readings in van Benthem 1986a. See Hendriks 1987 for systematic pruning.) Also, special care is needed as to argument order conventions. E.g., "John loves" should end up denoting the property of being loved by John, rather than loving John. The latter phenomena are handled in Bouma 1987 using type-theoretical terms assigned to derivations in the directed Lambek calculus. For instance, the earlier-mentioned argument-switching rule $((a\backslash b)/c) \Rightarrow (a\backslash(b/c))$ would then denote *conversion* of the relevant two-place predicate.

Another wide-spread temptation in the syntactic literature is to equate different constituent structures with presumably different readings, without exploring the precise correspondence. For instance, the otherwise quite interesting paper Houtman 1987 notes an ambiguity in the phrase "radio and television fan" (a slightly adapted example); which might mean "fan of both radios and televions" or "fan of joint radio-television sets". He then produces two distinct categorial analyses; which may be simplified for present purposes to the following [with N for common nouns, (N,N) for adjectival uses - conjunction being treated syncategorematically] :

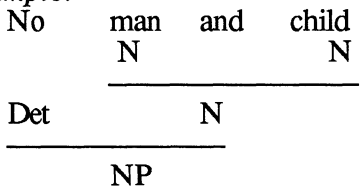
$$\frac{\dfrac{\text{radio}\ (N,N) \quad \text{and} \quad \dfrac{\text{television}\ (N,N) \quad \text{fan}\ N}{(N,N)}}{N}$$

versus

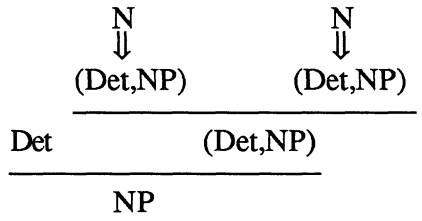
$$\frac{\dfrac{(N,N) \quad \dfrac{(N,N) \quad N}{\Downarrow}}{(N,N)}}{N} \qquad \text{(Montague raising)}$$

But, calculation of the corresponding terms will give the *same* autcome in both cases, being the first reading mentioned above. (To obtain the second reading, one would have to conjoin "radio" and "television" in type N, and only then use the result as an adjective.) On the other hand, with proper care, ambiguities in conjunction *can* often be handled in our framework:

Example.



The calculation for this standard analysis produces the reading "no male child". To obtain the other possibility, being 'no man and no child', one must proceed as follows:



A final practical question concerns the computational cost of generating all the required lambda-terms. Once the basic principle is understood, of course, one can devise various more efficient *implementations*, suppressing lambdas whenever convenient, by performing appropriate substitutions of arguments on the spot.

• *Theoretical Questions*

With our systematic semantic perspective, one can now start studying peculiarities of various categorial calculi as systems of possible meanings for natural language expressions. For instance, *how many* readings are assigned to one given valid transition $X \Rightarrow b$ (i.e., how many essentially different derivations exist?) It is proved in van Benthem 1986a (chapter 7) that the Lambek calculus produces only *finitely many* readings, up to logical equivalence, for any given expression of a type b. By contrast, the full intuitionistic calculus may generate *infinitely many* different ones; thus violating our intuitions about what is reasonable for natural language. In other words, one can try to locate the appropriate carriers of linguistic meanings in certain fragments of the type-theoretical language.

One general question again emerges in this perspective:

What are 'admissible' meaning changes for linguistic expressions?

We shall return to this issue in due course.

Nothing in the above, however, precludes consideration of richer notions of type change, going beyond the lambda/application language. For instance, one might add *identity* statements, allowing such type changes as

$$e \Rightarrow (e,t) \quad : \quad \lambda x_{(e,t)} \bullet x \equiv u_e \ .$$

This will send individuals to their singletons; as has been proposed by various authors in the treatment of *plurals* (cf. van Eyck 1985.) Nevertheless, as a methodological strategy, it is always wise to look for an explicit syntactic locus, rather than a hidden mechanism, producing the identity - such as plural markers or other particles of 'collectivization'.

• *Denotational Constraints*

Perhaps the most important virtue of our type-theoretic semantics is that it links up Categorial Grammar with ongoing research in logical semantics, where Type Theory is still a 'lingua franca' providing a suitable setting for asking general questions about natural language.

One such general question concerns *denotational constraints* governing various classes of expressions (cf. Keenan and Faltz 1985, van Benthem 1986a.) It is convenient to view these matters in the present perspective; because type changes and special denotational behaviour are often related. For instance, as is well-known, individuals behave like Boolean *homomorphisms* in the noun phrase domain of type ((e,t),t). But, this is entirely predictable from the type-theoretic term effecting their meaning change; being $\lambda x_{(e,t)} \bullet x(y_e)$. All terms of this form denote homomorphisms. Thus, 'plain' objects in one type may become embellished when changing to another.

There is also a converse to this observation. As a rule of thumb, nice mathematical behaviour in one type may really be a sign of belonging to another, simpler one. For instance, according to Keenan and Faltz, all prepositions are homomorphisms with respect to their Noun Phrase arguments:

"to Palo Alto and Los Altos"	↔	"to Palo Alto and to Los Altos"
"to Palo Alto or Los Altos"	↔	"to Palo Alto or to Los Altos"
"to every town"	↔	"for every town: to it"

But, as is shown in van Benthem 1986a (chapter 3), the homomorphisms in type (((e,t),t), ((e,t), (e,t))) are in canonical one-to-one correspondence with all ordinary functions in type (e, ((e,t), (e,t))). Thus, an alternative would be simply to demote prepositions to ordinary citizens of the latter type.

For a somewhat different example, consider the well-known distinction between *functors* and *adjoints*. Is not Categorial Grammar committed to treating all these on a par? Here, denotational constraints may make the difference. Adjoints are special functors, satisfying additional conditions; for instance, in the form of special *inference* patterns supported by them.

Example. An adjoint A in type ((e,t), (e,t)) will support the inferences

A (X)	implies	X
A (X∪Y)	is equivalent with	A(X) ∪ A(Y) .

It may be shown (van Benthem 1986a, chapter 3), that these two conditions suffice for representing A by means of some fixed set A* of individuals:

$$A(X) = X \cap A^* .$$

A case in point would be absolute adjectives, which reduce to unary predicates in a natural sense.

There are other aspects to the difference between functors and adjoints, such as their relative difference in *scopal* behaviour, which could also be studied in this setting. But for now, it will be sufficient to have established the possibility of this line of investigation.

• *General Semantic Mechanisms*

As was stated before, the Theory of Types provides a convenient setting for studying general semantic phenomena in natural language. But then, one criterion for admissibility of type changes, or categorial combination generally, might be how they *interact* with such phenomena. We shall consider two prominent examples.

In the type-theoretic perspective, there is one feature which links expressions across various categories, that we might want to call *logical.* (These comprise *verbs* like "be", *Boolean connectives* like "and", or *determiners* like "all", "some".) They are all *permutation-invariant* in the following general sense.

Definition. Let $\{D_a \mid a \in \text{TYPE}\}$ be a system of domains. Let π be a permutation of the individual domain D_e. The *canonical lifting* $\bar{\pi}$ turns π into a permutation on all domains:

$$\bar{\pi}_e = \pi \; ; \qquad \bar{\pi}_t \text{ is the identity on } D_t;$$

$$\bar{\pi}_{(a,b)}(f) = \{ (\bar{\pi}_a(x), \bar{\pi}_b(y)) \mid (x,y) \in f \}, \quad \text{for } f \in D_{(a,b)}.$$

An item $x \in D_a$ is *permutation-invariant* if $\bar{\pi}_a(x) = x$ for all individual permutations π.

Now, one general constraint on admissible type-changes might be this:

'Logical items should remain logical in their new guise'.

But in fact, this is guaranteed for *all* type changes definable in the lambda/application language ; even if we were to throw in identity as well (van Benthem 1986a, chapter 7).

There is also a converse, which may be worth stating.

Proposition. With a finite domain D_e of individuals , all permutation-invariant objects in any type can be defined uniquely by a term of the type-theoretic language having lambdas, application and identity.

Proof: See Appendix 7. ∎

Another plausible candidate for constraining type changes concerns logical *inference.* As is well-known, our type domains support a general notion of Boolean inclusion , defined as follows:

$$\sqsubseteq_t \text{ is } \leq \; ; \qquad \sqsubseteq_e \text{ is the identity on } D_e;$$
$$f \sqsubseteq_{(a,b)} g \quad \text{iff} \quad \forall x \in D_a \; f(x) \sqsubseteq_b g(x)$$

Then, one might require general preservation of inferential relationships for type changes $a \Rightarrow b$:

$$\text{If } u_a \sqsubseteq_a u^*_a \; ,$$
$$\text{then } \tau_b[u_a] \sqsubseteq_b \tau_b[u^*_a] \; .$$

In other words, the new semantic object τ_b should be *monotone* in its a-parameter.

This requirement does have a bite (cf. van Benthem 1986a, chapter 7). It holds for the Geach rule, but not for Montague raising in general. Thus, one might devise principled reasons for preferring more Geach-like, composition-based fragments of the Lambek calculus over its full variants with different forms of type raising. [Warning. We do not have 'full monotonicity', however, for composition with two arguments. "Not" + "moving" combines to "not moving", with types (t,t) (e,t) ⇒ (e,t). But, although "moving" implies "awake" in type (e,t), and "not" naturally implies "not" itself, "not moving" does not imply "not awake".]

There is a syntactic background to the previous observations. In the lambda term for the Geach rule, being (in general)

$$\lambda x_{(c,a)} \bullet \lambda y_{c} \bullet \underline{u}_{(a,b)} \ (x(y)) ,$$

the parameter $u_{(a,b)}$ occurs *positively*. And this fact alone explains the monotonic behaviour. By contrast, there is no such positive occurrence for the Montague rule, being (in general)

$$\lambda x_{(a,b)} \bullet x \ (\underline{u}_{a}) \quad .$$

This brings in a general logical issue. Is there a so-called *Preservation Theorem* for the type-theoretical language, characterizing the monotone type changes as being those definable by a term in which the relevant parameter has only positive occurrences? The answer depends on the particular fragment considered (van Benthem 1987b). The equivalence does not hold for the full type-theoretical language; but it does hold for its Lambek-fragment.

Thus, the present semantic setting also offers new employment for the traditional concerns of logical *Model Theory*.

Appendix.

The preceding topic raises a very practical issue. How are we to set up useful calculi of inference in the presence of type change?

As it happens, one can enrich the earlier categorial calculi with a mechanism of *monotonicity marking*, which keeps track of inferential potential. In general, there will be *two* sources then of monotone behaviour. One arises out of the general categorial mechanism: *functors* occupy monotone positions. And also, monotone behaviour may be encoded in arguments through Conditionalization.

Example. + - marking on the left-hand side of derivations:

$$\frac{t \Rightarrow t}{(t,t)\ t \Rightarrow t} \qquad \frac{\overset{+}{t}}{\overset{+}{(t,t)}\quad t}$$

$$\frac{(t,t)\ e\ (e,t) \Rightarrow t}{\overset{+}{(t,t)}\ e\ \overset{+}{(e,t)}}$$

Only those items having an unbroken string of + to the top occur in positive position. This reflects an example like "not John comes", where only the expression "not" admits replacement by a logically weaker one.

Now, continue as follows:

e (e,t) $\Rightarrow$ ((t,t),t)

The information that the withdrawn premise (t,t) occurred positively, can be encoded in the resulting type:

$((t,t)^{+},t)$

A second source of monotone behaviour are special *lexical items*, such as monotone determiners, which grant (some of) their arguments special status, whether "upward" or "downward" monotone. This information, too, can be encoded - and it then interacts with the above process in an obvious way. (See van Benthem 1986a, chapter 6.)

Thus, categorial combination viewed as implicational inference, and ordinary inference can co-exist in one natural calculus of deduction.

5. Variations

The account so far has given the 'standard picture' of Categorial Grammar and Type Theory. Once this has been understood, of course, many modifications are possible - and indeed even suggested by this formalism. To show the wide range of possibilities, here is a brief survey of reasonable alternatives.

5.1 *Syntactic Variations*

• Various authors have preferred an approach based on *Combinatory Logic*, rather than *Lambda Calculus*, as in the above. (Cf.Steedman 1985, Szabolcsi 1987.) In principle, the two frame-works are equivalent, as is known from general logic. The dominant tendency among logicians has been to employ the type-theoretic lambda approach, if only for reasons of intelligibility in reading the notation.

Nevertheless, extensionally equivalent frameworks may have interesting intensional differences. For instance, in the Combinatory Logic approach, it is natural to look for reasonable finite clusters of combinators. These would correspond, in the earlier Categorial Hierarchy, to those systems which admit of a finite Hilbert-style axiomatization. This question has been considered by Zielonka and Buszkowski, with outcomes such as the following (cf. Buszkowski 1987) :

> Neither the directed Lambek calculus L nor its undirected variant LP has a corresponding finite set of combinators.

On the other hand, there are cases of overlap too. Here is one semantically interesting case.

Proposition. The categorial calculus LPC has its derivations encoded by the following finite set of combinators:

I x	=	x	(Identity)
P xy	=	yx	(Permutation)
S xyz	=	xy (xz)	(Curry's S-combinator)
C xy	=	yz (xz)	(Composition)

Proof. See Appendix 7. ∎

• Another possible variant would be to change from *functional* types to *relational* ones (as advocated in Muskens 1987). Again, there is an extensional equivalence here between the two approaches (cf. Doets and van Benthem 1983). But this time, no intensional advantages are discernible at present. Even so, stating the framework may be of independent interest.

Types are constructed from e, using finite sequences. In general, e denotes the individual objects, and a sequence $(a_1,...,a_n)$ all n-ary relations on $D_{a1},...,D_{an}$ (respectively) ; i.e.,

$$D_{(a1,...,an)} = \text{POW}\ (D_{a1} \times ... \times D_{an}).$$

Some specific types are as follows:

e	: entities	
( )	: truth values	(the empty sequence)
(e)	: 1-place predicates	
(e,e)	: 2-place predicates	
((e))	: noun phrases	
(( ))	: 1-place connectives	
((),(()))	: 2-place connectives	

This perspective suggests *composition* of relational types as a natural operation. Also, *application* is definable in a natural sense, using 'projection' with respect to the relevant argument position.

In this format, rules of type combination will look as follows:

$$(X,a,Y) \quad a \quad \Rightarrow \quad (X,Y)$$

$$\frac{X\ ,\ a \quad \Rightarrow\ b}{X \Rightarrow (a) \frown b}$$

Example ('(t,t) ⇒ ((e,t),(e,t))'):

(e) e	⇒ ( )	
(( )) ( )	⇒ ( )	, so
(( )) (e) e	⇒ ( )	, and hence
(( )) (e)	⇒ (e)	, and also
(( ))	⇒ ((e),e)	

Unfortunately, this framework seems to offer exactly the same choice-points in setting up a Categorial Hierarchy as the earlier functional one. So, it does not appear to be of novel interest for present purposes.

• Current versions of categorial grammar often employ a measure of *polymorphism* in their basic type assignment, using types containing *variables* whose values can still be determined. For instance, negation might be assigned the polymorphic type ((x,t),(x,t)). (Cf. Zeevat, Klein and Calder 1987, Uszkoreit 1986.) There is a trade-off here, between polymorphism right in the initial assignment, and polymorphism as *induced* by our type change rules. (E.g., (t,t) ⇒ ((x,t),(x,t)) is derivable in the Lambek system, for arbitrary x.) The exact details of this trade-off remain to be understood.

The mechanism for combining polymorphic types is a form of *Resolution* in the simplest case, when added to the Ajdukiewicz system:

a (a*,b) $\Rightarrow$ σ(b),
where σ is a most general unifier of a and a*.

For the Lambek system in general, a similar generalization of its rules is possible (cf. Van Benthem 1987a). Thus, systems in the Categorial Hierarchy acquire a polymorphic 'second-order' version too. Many of the earlier questions are still open in this new setting. For instance,

Is the polymorphic Lambek calculus still decidable?

5.2 *Semantic variations*

As is usual in semantics, having one formal framework at once stimulates phantasy in setting up new ones. Although our type-theoretical semantics was couched in terms of standard function hierarchies on base domains, this is by no means the only possible interpretation for Categorial Grammar. It is important to emphasize this point, because many authors seem to feel that frameworks in formal semantics are 'package deals', where one has to buy all the components.

For instance, instead of having full function hierarchies, one can take smaller ones, using Henkin's *general models* for the Theory of Types (cf. Gallin 1975). Or, as Gordon Plotkin has suggested (private communication), the limited expressive power of the Lambek system invites a restriction to only *linear functionals* over some base lattice. Such sparser models can also come with a different view of what functions are: witness the *category -theoretic models* of Lambek 1987 which interpret type theory in suitable Cartesian-closed categories.

A category-theoretic approach might be useful at another level too. Many denotational constraints, as considered in Section 4, are formulated 'locally', within single models. But in general, meanings of linguistic expressions are functors picking an object in each model - and the resulting family should obey certain *uniformities*. For instance, if $D_e \subseteq D_e'$, then denotations in the hierarchy on D_e should be the natural D_e - *restrictions* of those in the model on D_e'. Again, the most elegant way of stating such uniformity may involve the mathematical apparatus of Category Theory.

But, one can equally well go by a less standard route, and provide type-theoretical formulas with associated *discourse representations* (cf. Klein 1987), or other recent 'dynamic' model structures. In this process, the use of *functions* itself is not sacrosanct. For many purposes, functional types can be viewed just as well as standing for *parametrized objects*. (In fact, a function itself *is* a good mathematical model for what one might mean by a 'parametrized object'. Cf. van Benthem 1986b.)

Other semantic variations concern such aspects as how to treat Booleans, or other specific kinds of expression. For instance, our treatment of predicate *negation* (type ((e,t),(e,t))) as being derived from sentence negation, gives it a standard bivalent semantics induced by the usual truth tables. But undoubtedly, there is a use of predicate negation which is at least trivalent: *true, false, undefined* (cf. Horn 1987). There is no difficulty in principle, however, in interpreting our type-theoretical language in three-valued, or other kinds of *partial models*, to accommodate such observations.

Finally, one task which is still to be done concerns the additional structure present in actual linguistic uses of Categorial Grammar. For instance, the phenomenon of subcategorization demands the use of *subtypes*, with a corresponding enrichment of our model structures and logics. We may need a general logic of relations among types, encompassing Lambek-type derivability, type inclusion, as well as substitutional specification ('subsumption'). Likewise, the general use of syntactic *features* suggests an eventual link up between the above semantics and current logics of feature structures and unification (cf. Kasper & Rounds 1987).

6. Intensionality

Among the many tasks in our program one stands out as being of particular importance. Since Montague Grammar already covered *intensional* phenomena, Categorial Grammar can do no less. Can the previous approach be generalized to an intensional Type Theory, having an additional base domain D_s for 'indices', 'possible worlds' or 'situations' ? Actually, there is a series of questions here. Of course, the Categorial Hierarchy was already defined in such a way that it does not depend on any particular choice of primitive types. But, many of our specific questions were couched in terms of the specific base set {e,t}.What will become of of these now?

- *The Mechanism of Intensional Interpretation*

As a general strategy of intensionalization, one can *reinterpret* the former type t as standing for, not truth values, but *propositions*. By the familiar reduction of the latter to sets of indices, then, these may be indentified with the objects of type (s,t) (with 't' now taken again in its old sense). Thus, types formerly assigned to expressions undergo a uniform innnntensionalization, via the replacement

$$t^* = (s,t).$$

This process is studied in van Benthem 1987d; which proves, amongst others, that no new syntactic structures are created in this way:

For all extensional {e,t}-types X,a,
$X \Rightarrow_L a$ if and only if $X^* \Rightarrow_L a^*$

In a sense, the *-operation is a form of type change itself. Is there any uniform meaning associated with it? This time, the picture seems diverse. Some extensional items indeed become intensional by mere type changing. An example is Boolean negation. (t,t)* is ((s,t),(s,t)), which gets its meaning automatically via the ordinary categorial transition $(t,t) \Rightarrow ((s,t))$.

In general, however, there may be several *options* for items in intensional contexts. For instance, the determiner "every" has its type ((e,t),((e,t),t)) intensionalized to

((e,(s,t)), ((e,(s,t)), (s,t)).

One 'conservative' option here is to employ again a derivation from the extensional case:

((e,t), ((e,t), t)) ⇒ ((e,(s,t)), ((e, (s,t)), (s,t)) is derivable, not in the base system L, but in the calculus LPC (with Contraction on index parameters).The associated meaning will be the natural lifting:

$$\lambda x_{(e,(s,t))} \bullet \lambda y_{(e,(s,t))} \bullet \lambda z_s \bullet \mathrm{EVERY}_{((e,t),((e,t),t))}\ ((x(z))(y(z)).$$

(In general, type change calculi for s may be more liberal than those for just e,t.) But, there could also be a truly intensional 'law-like' "every", quantifying over more situations in D_s than just the actual one. An interesting general issue here is to define a good notion of *extensionality* for items in such intensional models (as being those which have remained pure {e,t}-objects at heart).

• *Invariance and Monotonicity*

Another immediate question is what becomes of earlier general semantic themes attached to Type Theory and hence to Categorial Grammar. Two prominent examples are the earlier notions of *logicality/permutation invariance* and *monotonicity*. We shall consider this matter for some special cases, rather than in complete generality. The general outcome is this. Matters become more complicated, as notions split up into several variants. They also become more interesting, as we gain more power of discrimination.

One example (treated in van Benthem 1986a, chapter 5) is that of *time*. Here, indices of type s may be taken to be points in time, and propositions can vary their truth value along some temporal order. Propositional operators obtain type ((s,t),(s,t)) in this setting, and one can investigate special classes of denotations in this intensional type. Applying the old notion without any changes has the following result:

the permutation-invariant operations on sets of points are precisely the Boolean ones.

A larger class of truly temporal operators arises by requiring invariance only with respect to *automorphisms* of the temporal order. Various forms of monotonicity can then be used to classify the resulting 'tenses', or operators of temporal perspective.

Another example of a basic intensional setting is that of *situations*. Again, these do not come as a bare set, but rather as a structure of partial objects, ordered by *inclusion* : $(S,\sqsubseteq)$. Propositional operators will again be functions from sets of situations to sets of situations, which may satisfy certain constraints. For instance, a reasonable general constraint might be *invariance for inclusion automorphisms* π, in the following sense:

$$\pi\,[f(X)] = f(\pi[X]), \quad \text{for all } X \subseteq S.$$

Again, the Boolean operations pass this test; but so do various 'modalities', such as 'definite truth':

$$\Box\,(X) = \{s \in S \mid \forall s' \in S\ (s \sqsubseteq s' \Rightarrow s' \in X)\}.$$

It would be of interest to delimit a natural class of modalities in this setting, using automorphism invariance plus some strong forms of monotonicity.

For the moment, we just point out some new distinction which arise here. In particular, some care is needed with common notions like 'monotonicity' or 'persistence'. As they were used before, these referred to Boolean inclusion within one single model (or world). E.g. "every" is (upward) *monotone* in its right-hand argument:

EVERY (A,B), $B \subseteq B^+$ implies EVERY (A,B^+).

What this does not mean, however, is that "every" is *persistent* in the sense that universal statements true in one situation are also true in larger situations. Thus, having the additional -structure on the domain S really adds new semantic notions.

For instance, propositions themselves may come in different classes of closure behaviour with respect to inclusion. Some propositions are persistent, in the above sense of being upward preserved under inclusion. It has even been proposed that *all* propositions might be *convex*, in the sense of containing all situations in between situations belonging to them. One general question here is how propositions acquire or lose such properties by their construction. For instance, on the first reading presented above for "every", propositions of the form "every AB" are not persistent. They would be persistent, however, with a more law-like "every" quantifying over all extensions of a given situation.

Actually, the best perspective for studying these matters would be a *three-valued* semantics, as mentioned in Section 5.2. A thorough logical study of first-order predicate logic in this setting may be found in Langholm 1987; which characterizes, e.g., exactly those first-order-formulas defining persistent propositions. Can the same be done for arbitrary formulas of type (s,t) in Type Theory?

The intensional perspective does not just bring a proliferation of semantic notions. It also brings the potential for new applications. For instance, this new setting is required if one wants to extend our earlier comparison between *natural languages* (as described in Categorial Grammar) and *formal languages* (as employed in Type Theory), to the realm of *programming languages* :

- *Programs and Operational Semantics*

The preceding analysis can also be applied to 'dynamic logic', where programs are naturally viewed as being of type (s,s) : i.e., functions from *states* to *states* of some computer. Or, more generally, allowing undefined or multi-valued (indeterministic) cases, one may use the relational type (s,(s,t)) eventually. (Cf. Harel 1984.) On such programs, there are certain natural operations, such as *Composition* (;) , *Boolean Choice* (IF THEN ELSE) or *Iteration* (WHILE DO) . For instance, Composition has the following type

((s,s), ((s,s), (s,s))),

and Iteration has

((s,t), ((s,s), (s,s))).

(Note the propositional type (s,t), needed for testing a controlling assertion in different states.)

Interestingly, basic operations in this setting are again *logical*, in the earlier sense of permutation invariance. Conversely, Logicality provides a uniformity in programming notions across different categories. Classifying all permutation-invariant items in given types is a problem of already considerable complexity here; witness Plotkin 1980. We shall consider some examples.

Starting with pure s-types, there is only one logical item in type (s,s), viz. the identity function. (One supposes the ground domain D_s to be large enough to exclude 'accidental' candidates, such as 'reversal' on $D_s = \{1,2\}$.) This is

the familiar instruction SKIP. As for operations on programs, it can still be shown that the only logical candidates are *Composition* and its iterations. (For a check, one can see at least that all items in, e.g., ((s,s), ((s,s), (s,s))) which are definable in our earlier lambda/application language must be of this kind. See Appendix 7.) Thus, composition deserved its central rôle in the above.

When control is exercised, types will also contain the Boolean t. For instance, a type like ((s,s),t) will already have many logical items, including all classes of programs definable by some condition expressible in the lambda/application/identity language (cf. Section 4). An example is the formula $\exists!x\ f(x) \equiv x$ ('f has exactly one fixed point'). The basic operation of control in the above, however, is the conditional choice IF THEN ELSE. Its meaning may be expressed as follows:

$$\lambda x_{(s,t)} \bullet \lambda y_{(s,s)} \bullet \lambda z_{(s,s)} \bullet \lambda u_s \bullet \iota v \bullet ((x(u) \wedge v \equiv y(u)) \vee (\neg x(u) \wedge v \equiv z(u)))$$

As in an earlier passage, there is an 'inflated' intensional meaning here, derived from a simpler 'local ' type

$$(t, (s, (s,s)))\ .$$

I.e., given a truth value, being the outcome of a test on the current situation, and two possible goal states, one resulting state is to be selected. The logical items in this type are easily classified (see Appendix 7), with IF THEN ELSE being the only non-trivial candidate.

Another reason for introducing mixed s,t-types arises with programs viewed as *relations*. Then, even basic operations on programs will involve Boolean structure, and a richer picture arises, where more of our earlier notions make sense. For instance, logical binary operations on relations in type (s,(s,t)) include both *Boolean* ones (compare Section 4) and relational *Composition*, as well as more complex cases. Moreover, such operations can be *monotone* in our earlier sense. For instance, composition satisfies double monotonicity:

$$R \subseteq R',\ S \subseteq S' \text{ imply } R \circ S \subseteq R' \circ S'.$$

And stronger constraints may be used too, in describing important special classes of operations. For instance, composition is also *continuous* in both of its arguments:

$$\cup \{R_i \mid i \in I\} \circ \cup \{S_j \mid j \in J\} = \cup \{R_i \circ S_j \mid i \in I, j \in J\}.$$

As an illustration, the continuous logical items in the simpler type of operations on programs (being ((s,(s,t)), (s,(s,t)))) may be classified:

Proposition. The continuous logical operations on programs are those definable in the following form:

$$\lambda R_{(s,(s,t))} \bullet \lambda x_s \bullet \lambda y_s \bullet \exists u_s \bullet \exists v_s \bullet \langle \text{Boolean combination of identities in } x, y, u, v \rangle$$

Proof. See Appendix 7. ∎

These operations include as major examples: Identity, Converse, Diagonal and Projection. (Further applications, and generalizations, of logical invariance in the semantics of programming languages are found in Trakhtenbrot 1987.)

We conclude with another analogy to previous themes in this paper. The phenomenon of *type change* occurs naturally in programming languages too, for reasons similar to those encountered in natural language. For instance, ordinary composition can extend its action through type change. When a binary function composes with two unary ones (as in notations for Recursion Theory), the underlying type change is as follows:

$$((s,s), ((s,s), (s,s))) \Rightarrow ((s, (s,s)), ((s,s), ((s,s), (s,s)))).$$

This transition is not derivable in Lambek's L; but it is derivable in LPC. (Recall an earlier remark about greater freedom for manipulating intensional types s.) The associated meaning of Section 4 will match the intended interpretation:

$$\lambda x_s \bullet f^2(g^1(x))\,(k^1(x))$$

(See Appendix 7 for details.)

This analogy brings us to the general comparison of natural languages and programming languages in this framework. We have found similarities in general semantic questions. Are there also similarities in the structure of specific types? This question is not easy to answer, as the intensional type s has a rather different motivation in both cases. Nevertheless, in recent 'dynamic' theories of meaning, propositions themselves are viewed as transformations on (knowledge) states. In that light, one might study the natural language counterpart of the programming notion of *control*, as well as other dynamic features in computation. Again, there may be limits here. Does natural language have any built-in counterpart to, say, *iteration* or *recursion* ?

In the other direction, many questions may be formulated too. We saw how semantic analyses in terms of *denotational constraints* can be applied to programming notions too. But perhaps, the same holds for syntax. For instance, how does type change affect *recognition* and complexity of programming languages? Is the frequent occurrence of LPC (with its threat of collapse to regular languages; cf. Section 3) significant in this respect?

The very formulation of these various questions shows how Categorial Grammar and Type Theory provide a suitable framework for discussing common features of natural languages and programming languages.

7. Appendix

I. *Proof of the Proposition in Section 4*

The language with application, lambda abstraction and identity can define all the usual logical operators $\neg, \wedge, \vee, \forall, \exists$ (cf. Gallin 1975). This will make the following construction possible.

First, here is a general result about interpretation in type-theoretic models M.

Lemma : For any term τ, and any assignment A to its free variables, the following identity holds for all permutations π:

$$\bar{\pi}\,[\![\tau]\!]_A = [\![\tau]\!]_{\bar{\pi} \circ A}\,.$$

The proof is by induction on the construction of τ.

Now, if $\varphi(x_a)$ defines an object f of type a uniquely, then - since $\bar{\pi}(f)$ satisfies φ in M when f does (by the Lemma) - $\bar{\pi}(f) = f$ for all π. So, all type-theoretic terms define invariant denotations, on all models, finite or infinite.

As for the converse, we can use a standard model-theoretic argument, which works on finite, but also on suitably saturated infinite models. The argument depends on the following

Lemma : If $f,g \in D_a$, and $(M,f) \equiv (M,g)$, then there exists a permutation π of D_e with $\bar{\pi}(f) = g$.

Then, a logical $f \in D_a$ can be defined uniquely as follows. For every $g \neq f \in D_a$, there exists some formula $\alpha_g(x_a)$ with $(M,f) \models \alpha_g$, $(M,g) \not\models \alpha_g$. (Otherwise, $(M,f) \equiv (M,g)$; and hence, by the Lemma, some $\bar{\pi}$ would map f onto g; which contradicts the permutation invariance of f.) Now, f is uniquely defined by the conjunction

$$\bigwedge \{\alpha_g \mid g \in D_a, g \neq f\} .$$

Finally, the second Lemma is proved by a standard zigzag-argument, finding increasing sequences $\vec{a}$, $\vec{b}$ such that $(M,\vec{a})$, $(M,\vec{b})$ always remain elementarily equivalent. When the relevant portion of the hierarchy has been exhausted (i.e., the transitive closure of D_a), one can read off the desired permutation on the ground domain D_e, and show that its canonical lifting respects the correspondence between all items in the matching sequences; in particular, that between f and g. ∎

II. *Proof of the Proposition in Section 5.1*

First, one shows that the following axioms H give an alternative Hilbert-style axiomatization of derivability in LPC:

$x \rightarrow x$	1
$(x \rightarrow (y \rightarrow z)) \rightarrow (y \rightarrow (x \rightarrow z))$	2
$(x \rightarrow (y \rightarrow z)) \rightarrow ((x \rightarrow y) \rightarrow (x \rightarrow z))$	3
$(x \rightarrow y) \rightarrow ((y \rightarrow z) \rightarrow (x \rightarrow z))$	4

Lemma: A sequent $X \Rightarrow b$ is derivable in LPC if and only if there exists a derivation of b from X using axioms from H and the rule of Modus Ponens only, in which each premise from X is used at least once.

Proof. The main task is to show that H admits of Conditionalization. This calls for a somewhat closer inspection of the usual proof: with the above axioms needed to push through the various cases for Modus Ponens. ∎

Next, there is a well-known correspondence between such Hilbert-style proofs and terms with combinators. For the typed case considered here, we can think, either of *typed* combinators, or of *polymorphic* ones adapting to context. Some illustrations of the principle are as follows:

- $e \Rightarrow ((e,t),t)$

H-proof: i ((e,t), (e,t)) ,axiom 1
ii (e,((e,t),t)) ,Modus Ponens (i,axiom 2)
iii e ,assumption
iv ((e,t),t) ,Modus Ponens (iii,ii)

Combinatory term: $P(I)(v_e) :=$

$\lambda x \bullet \lambda yz \bullet x(z)y \ (\lambda u \bullet u) \ (v_e) =$
$\lambda yz \bullet z(y) \ (v_e) =$
$\lambda z \bullet z(v_e)$

- $(e,(e,t)) \Rightarrow (e,t)$

H-proof: i (e,(e,t)) ,assumption
ii ((e,(e,t)),((e,e),(e,t))) ,axiom 3
iii ((e,e),(e,t)) ,Modus Ponens (i,ii)
iv (e,e) ,axiom 1
v (e,t) ,Modus Ponens (iv,iii)

Combinatory term: $S(v_{(e,(e,t))}) \ (I) :=$

$\lambda x \bullet \lambda yz \bullet x(z)(y(z)) \ (v_{(e,(e,t))}) \ (\lambda u \bullet u) =$

$\lambda yz \bullet v_{(e,(e,t))} \ (z) \ (y(z)) \ \ (\lambda u \bullet u) =$

$\lambda z \bullet v_{(e,(e,t))} \ (z) \ (z).$

III. *Proofs of some Assertions in Section 6*

- Functions in ((s,s),((s,s),(s,s))) which are lambda/application definable have a definition in *normal form*, in which all possible lambda conversions have been performed. In particular also, all types occurring as subscripts of variables in the definition must be proper subtypes of the above. Some argument about the shape of this normal form will show that it must look like this:

$$\lambda x_{(s,s)} \bullet \lambda y_{(s,s)} \bullet \lambda z_s \bullet \ < \text{applications of } x,y,z > .$$

- Logical terms in type (t, (s,(s,s))) consist of two functions in (s,(s,s)) (one for t = 0, the other for t = 1), both of which are permutation-invariant themselves (as D_t cannot be permuted). And of the latter items, there are only two, being

$$\lambda x_s \bullet \lambda y_s \bullet x_s \text{ and } \lambda x_s \bullet \lambda y_s \bullet y_s .$$

To see this, consider two objects a,b in D_s, together with f(a,b), where f is the invariant function. Logicality has two kinds of effect. One is *Locality* :

$$f(a,b) \in \{a,b\}.$$

(Otherwise, we could keep a,b fixed, while permuting f(a,b) to some other object - thus disturbing f by a permutation). The other effect is *Uniformity* :

If $f(a,b) = a$ for some a,b (with $a \neq b$)
then $f(x,y) = x$ for all x,y.

The reason is this. If $x = y$, then $f(x,y) = x = y$, by Locality. If $x \neq y$, then some permutation π sends a to x, b to y, and by the invariance of f, it follows that $f(x,y) = f(\pi(a),\pi(b)) = \pi(f(a,b)) = \pi(a) = x$. So, the choice of projection is made uniformly.

• Continuous operations f in type $((s,(s,t)), (s,(s,t)))$ have the following property. Since any argument R is the union of its *singleton* subsets, $f(R)$ equals the union of all $f(\{< x,y >\})$ (with Rxy). This explains the form of the definition

$$\lambda R_{(s,(s,t))} \bullet \lambda x_s \bullet \lambda y_s \bullet \exists u_s \bullet \exists v_s \bullet \ ....$$

Next, it suffices to consider the admissible patterns of the form

$$\{< u,v >\} \ x \ y \ .$$

Using permutation invariance, the only distinguishable cases are those definable by Boolean combination of identities among these objects. (E.g., if x is distinct from u, v and y, *any* such distinct x' will be admissible in $f(R)$: using a permutation leaving u,v,y fixed, while sending x to x'.)

• A derivation for binary composition in LPC which produces the intended meaning. A natural deduction format will be used for ease of presentation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{((s,s),((s,s),(s,s))}\quad \dfrac{}{(s,(s,s))}}{((s,s),(s,s))}\quad \dfrac{}{(s,s)}}{(s,s)}\quad \dfrac{}{(s,s)}}{s}\quad *}{s}}{(s,s)}\quad \text{withdraw } *$$

Acknowledgement

I would like to thank Wojciech Buszkowski, Joyce Friedman, Aravind Joshi, Michael Moortgat, Gordon Plotkin, Mark Steedman and Anna Szabolcsi for stimulating conversations.

8. References

A. Ades and M. Steedman, 1982, 'On the Order of Words', *Linguistics and Philosophy* 4, 517-558.

E. Bach, 1984, 'Some Generalizations of Categorial Grammars', in F. Landman and F. Veltman, eds, 1984, 1-23.

E. Bach, R. Oehrle and D. Wheeler, eds, 1987, *Categorial Grammars and Natural Language Structures*, Reidel, Dordrecht and Boston.

R. Bäuerle, C. Schwarze and A. von Stechow, eds, 1983, *Meaning, Use and Interpretation of Language*, De Gruyter, Berlin.

J. van Benthem, 1986a, *Essays in Logical Semantics*, Reidel, Dordrecht and Boston.

J. van Benthem, 1986b, 'The Relational Theory of Meaning', *Logique et Analyse* 29, 251-273.

J. van Benthem, 1987a, 'Categorial Equations', Faculteit Wiskunde en Informatica, University of Amsterdam . (To appear in Klein and van Benthem, eds, 1987.)

J. van Benthem, 1987b, 'Categorial Grammar and Lambda Calculus', to appear in D. Skordev, ed, 1987.

J. van Benthem, 1987c, 'Semantic Type Change and Syntactic Recognition', to appear in Chierchia, Partee and Turner, eds, 1987.

J. van Benthem, 1987d, 'Strategies of Intensionalisation', to appear in L. Pólos, ed, *Festschrift for Imre Ruzsa*, 1987, Filozófiai Figyelö, L. Eötvös University, Budapest.

J. van Benthem, 1987e, 'The Lambek Calculus', to appear in Bach, Oehrle and Wheeler, eds, 1987.

J. van Benthem, 1988, 'Parallels in the Semantics of Natural and Programming Languages', to appear in M. Garrido et al., eds, 1988.

G. Bouma, 1987, 'Flexible Phrase Structure Grammars and Categorial Unification Grammars', Institut für Romanistik/Linguistik, University of Stuttgart. (To appear in Klein and van Benthem, eds, 1987.)

W. Buszkowski, 1982, *Lambek's Categorial Grammars,* dissertation, Mathematical Institute, Adam Mickiewicz University, Poznan.

W. Buszkowski, 1986, 'Completeness Results for Lambek Syntactic Calculus', *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* 32, 13-28.

W. Buszkowski, 1987, 'Hilbert-style Axiomatization for the Lambek-van Benthem Calculus', Mathematical Institute, Adam Mickiewicz University, Poznán.

W. Buszkowski, W. Marciszewski and J. van Benthem, eds, 1987, *Categorial Grammar* , John Benjamin, Amsterdam and Philadelphia.

G. Chierchia, B. Partee and R. Turner, eds, 1987, *Categories, Types and Semantics*, Reidel, Dordrecht and Boston.

D. Davidson and G. Harman, eds, 1972, *Semantics of Natural Language*, Reidel, Dordrecht and Boston.

H. Doets and J. van Benthem, 1983, 'Higher-Order Logic', in D. Gabbay and F. Guenther, eds, 1983, 275-329.

K. Došen, 1986, *Sequent Systems and Groupoid Models*, Mathematical Institute, Serbian Academy of Sciences, Belgrade.

J. van Eyck, 1985, *Aspects of Quantification in Natural Language*, dissertation, Filosofisch Instituut, Rijksuniversiteit Groningen. (To appear with Reidel, Dordrecht and Boston.)

J. Friedman and R. Venkatesan, 1986, *Categorial and Non-Categorial Languages*, Technical Report 86/005, Computer Science Department, Boston University.

D. Gabbay and F. Guenther, eds, 1983, *Handbook of Philosophical Logic*, vol. I, Reidel, Dordrecht and Boston.

D. Gabbay and F. Guenther, eds, 1984, *Handbook of Philosophical Logic*, vol. II, Reidel, Dordrecht and Boston.

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

M. Garrido et al., eds, 1988, *Logic Colloquium. Granada 1987*, North-Holland, Amsterdam.

P. Geach, 1972, 'A Program for Syntax', in D. Davidson and G. Harman, eds, 1972, 483-497.

J. Groenendijk and M. Stokhof, 1987a, 'Dynamic Predicate Logic', Filosofisch Instituut, University of Amsterdam.

J. Groenendijk and M. Stokhof, 1987b, *Type-shifting Rules and the Semantics of Interrogatives*, report 87-01, Institute for Language, Logic and Information, University of Amsterdam. (To appear in Chierchia, Partee and Turner, eds, 1987.)

N. Haddock et al., eds, 1987, *Categorial Grammar, Unification Grammar and Parsing*, Centre for Cognitive Science, University of Edinburgh.

D. Harel, 1984, 'Dynamic Logic', in D. Gabbay and F. Guenthner, eds, 1984, 497-604.

H. Hendriks, 1987, 'Type-driven Translation, Type-Ambiguity, and the Lambek Calculus', Filosofisch Instituut, Universiteit van Amsterdam . (To appear in Klein and van Benthem, eds, 1987.)

L. Horn, 1987, 'Aristotle as a Montague Grammarian',
talk presented at the *ASL/LSA Colloquium on Logic and Linguistics*,
Stanford, July 1987.

J. Houtman, 1987, 'Coordination in Dutch',
Nederlands Instituut, Rijksuniversiteit Groningen.
(To appear in Klein and van Benthem, eds, 1987.)

R. Kasper and W. Rounds, 1987, 'The Logic of Unification in Grammar', to appear in *Linguistics and Philosophy*.

E. Keenan & L. Faltz, 1985, *Boolean Semantics for Natural Language*,
Reidel, Dordrecht and Boston.

E. Klein, 1987, 'DRT in Unification Categorial Grammar',
Centre for Cognitive Science, University of Edinburgh.

E. Klein and J. van Benthem, eds, 1987, *Categories, Polymorphism and Unification*, Centre for Cognitive Science (University of Edinburgh)
and Institute for Language, Logic and Information (University of Amsterdam).

J. Lambek, 1958, 'The Mathematics of Sentence Structure',
American Mathematical Monthly 65, 154-170.

J. Lambek, 1987, 'Categorial and Categorical Grammars',
to appear in Bach, Oehrle and Wheeler, eds, 1987.

F. Landman and F. Veltman, eds, 1984, *Varieties of Formal Semantics*,
Foris, Dordrecht and Cinnaminson, (GRASS series, vol. 3).

T. Langholm, 1987, *Partiality, Truth and Persistence*,
dissertation, Department of Philosophy, Stanford University.
(To appear in CSLI Lecture Notes, Chicago University Press.)

R. Montague, 1974, *Formal Philosophy*,
Yale University Press, New Haven, (R. Thomason, ed.).

M. Moortgat, 1984, 'Functional Composition and Complement Inheritance', in *Proceedings of Conference on Meaning and the Lexicon. Cleves 1983*, Nijmegen.

M. Moortgat, 1985, 'Mixed Composition and Discontinuous Dependencies', Instituut voor Nederlandse Lexicologie, Leiden.
(To appear in Bach, Oehrle and Wheeler, eds, 1987.)

M. Moortgat, 1987, 'Lambek Theorem Proving',
to appear in Klein and van Benthem, eds, 1987.

R. Muskens, 1986, 'A Relational Formulation of the Theory of Types',
Report 86-04, Institute for Language, Logic and Information,
University of Amsterdam. (To appear in *Linguistics and Philosophy*.)

R. Pareschi and M. Steedman, 1987, 'A Lazy Way to Chart-Parse with Categorial Grammars', to appear in *Proccedings 25th Annual Meeting of the Association for Computational Linguistics, Stanford, July 1987*.

B. Partee and M. Rooth, 1983, 'Generalized Conjunction and Type Ambiguity', in Bäuerle, Schwarze and von Stechow, eds, 1983, 361-383.

G. Plotkin, 1980, 'Lambda Definability in the full Type Hierarchy', in J. Seldin and J. Hindley, eds, 1980, 363-373.

J. Seldin and J. Hindley, eds, 1980, *To H.B. Curry. Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, New York.

D. Skordev, ed, 1987, *Druzhba Summer School in Applied Logic*, Plenum Press, New York.

M. Steedman, 1985, 'Dependency and Coordination in the Grammar of Dutch and English', *Language* 61, 523-568.

A. Szabolcsi, 1987, 'Bound Variables in Syntax (Are there Any?)', Research Institute for Linguistics, Hungarian Academy of Sciences.

B. Trakhtenbrot, 1986, 'Using Logical Relations in Programming Semantics', to appear in D. Skordev, ed, 1987.

H. Uszkoreit, 1986, 'Categorial Unification Grammars', in *Proceedings of the 11th International Conference on Computational Linguistics, Bonn, August 1986*, 187-194.

H. Zeevat, E. Klein and J. Calder, 'An Introduction to Unification Categorial Grammar', in N. Haddock et al., eds, 1987.

F. Zwarts, 1986, *Categoriale Grammatica en Algebraische Semantiek*, dissertation, Nederlands Instituut, Rijksuniversiteit Groningen. (To appear with Reidel or Foris.)