STUDIES IN LOGIC

AND

THE FOUNDATIONS OF MATHEMATICS

VOLUME 130

S. ABRAMSKY / J. BARWISE / K. FINE / H.J. KEISLER / A.S. TROELSTRA
EDITORS

Language in Action

Categories, Lambdas and Dynamic Logic

WITH ADDENDA

J. VAN BENTHEM

NORTH-HOLLAND
AMSTERDAM • LONDON • NEW YORK • TOKYO

STUDIES IN LOGIC

AND

THE FOUNDATIONS OF MATHEMATICS

VOLUME 130

NORTH-HOLLAND
AMSTERDAM • LONDON • NEW YORK • TOKYO

LANGUAGE IN ACTION

Categories, Lambdas and
Dynamic Logic

Johan VAN BENTHEM
Faculteit der Wiskunde en Informatica
Universiteit van Amsterdam
Amsterdam, The Netherlands

NORTH-HOLLAND
AMSTERDAM · LONDON · NEW YORK · TOKYO

ELSEVIER SCIENCE B.V.
Sara Burgerhartstraat 25
P.O. Box 211, 1000 AE Amsterdam, The Netherlands

First printing: 1991
Second impression: 1994

ISBN 0 444 89000 9

This book is printed on acid-free paper.

Printed in The Netherlands

LANGUAGE IN ACTION
Categories, Lambdas and Dynamic Logic

to the memory of my father

Abraham Karel van Benthem

PREFACE

This book has arisen out of a series of papers documenting some five years of research into the logical foundations of Categorial Grammar, a grammatical paradigm which has close analogies with Lambda Calculus and Type Theory. The technical theory presented here is a natural outgrowth of an earlier interest in the interface between Logic and Linguistics: in particular, the theory of generalized quantification developed in van Benthem 1986A. Like many of my more linguistic colleagues in The Netherlands, I have gradually found that a categorial framework, provided with a lambda calculus oriented semantics, is a most convenient vehicle for generalizing semantic insights obtained in various corners of natural language into one coherent theory. Moreover, this book is meant to demonstrate to fellow logicians that the resulting applied lambda calculus is not without its intrinsic logical interest either.

Nevertheless, theorizing at this level of abstraction has also led to a general change in my thinking about language. In the final analysis, it seems to me now, the crucial issue is not primarily to 'break the syntactic code' of natural languages, but rather to understand the cognitive functioning of the human mind. And the latter manifests itself not just in the grammatical patterns of our sentences, but also in more global textual arrangement, steered by discourse particles and interpunction, or even in the invisible rules guiding our language games. Fortunately, as it happens, the various formal systems studied here as engines for categorial grammatical analysis also admit of more 'procedural' interpretations, as logics of dynamic interpretation and inference. Thus, motives from Computer Science also make their appearance, resulting in a shift from Categorial Grammar to Dynamic Logic towards the end of the book. For a logician, this is not such a dramatic move as it might seem: for instance, the technical setting for this shift lies in Modal Logic, another earlier research interest of the present author (witness van Benthem 1985). At the moment, I cannot judge the precise significance of this emerging connection between these various logical concerns: perhaps, human beings just cannot help singing the same tune once in a while.

I would like to thank a number of people for their direct help in the actual preparation of this manuscript: in particular, Dirk Roorda, Victor Sanchez and Anne Troelstra have provided many useful comments. As to its incubation period, I have profited for a long time from participation in an active Categorial Grammar community,

both in Holland and abroad, including such inventive colleagues as Wojciech Buszkowski, Michael Moortgat and Frans Zwarts. Finally, for a more general intellectual environment, I should also mention the 'Institute of Language, Logic and Information' at the University of Amsterdam, which has provided a very pleasant environment for someone who finds it hard to stay within the usual academic boundaries, and sometimes even more heretically (rumour has it), to share the received value judgments and intellectual dogmas of the logical profession.

Table of Contents

I INTRODUCTION

In this first Part, the basic ingredients of this book will be presented, whose interaction, and resulting logical theory, will then unfold subsequently.

1 Two Traditions

The idea that the objects of thought form a hierarchy of categories is an old one in Philosophy. That this hierarchy may be based on function-argument relationships was realized by two eminent mathematicians/philosophers in the last century, namely Gottlob Frege and Edmund Husserl. Their influence may be traced in two subsequent streams of logical investigation. One is that of mathematical ontology, where Bertrand Russell developed his *Theory of Types* describing mathematical universes of individual objects, functions over these, functionals over functions, et sursum. Although set theory in the Zermelo style, rather than type theory, became the eventual 'lingua franca' of Mathematics, the type-theoretic tradition in the foundations of mathematics has persisted to this day, inspiring such important special research programs as the Lambda Calculus in its course. A second stream arose in Philosophy, where Lesniewski and Ajdukiewicz developed a technical theory of categories, which eventually became a paradigm of Linguistics known as *Categorial Grammar*.

In the meantime, both streams have found applications within a single new area, namely that of Computer Science. Type theories play a conspicuous role in the semantics of programming languages, while categorial grammars are prominent in computational processing of natural languages. In fact, a convergence may already be observed in the seminal work of Richard Montague, where mathematically inspired type theories fed into the semantics of natural languages. By now, however, a much richer picture is emerging of possible contacts between the two streams: and the main purpose of this book is to set out the basic ideas of logical syntax and semantics relating Categorial Grammar and Lambda Calculus. In the process, it will also become clear that there remains no natural frontier between the fields of mathematical linguistics and mathematical logic.

One pervasive concern in our treatment may be called *fine-structure*. Notably, Montague's original system employed a very powerful type-theoretic machinery, much of which is not actually involved in providing meanings for linguistic expressions. Thus, it may be compared to a syntactic paradigm like unrestricted rewrite grammars, which generates all recursively enumerable languages, but which has to be 'fine-tuned' at various lower levels (e.g., context-free, or regular) in order to get better estimates of the actual syntactic complexity of linguistic constructions. Likewise, we shall be concerned here with a spectrum of categorial calculi, corresponding to a hierarchy of fragments of the full type-theoretic language, in order to get a more sensitive instrument for semantic analysis. That such an interest in fragments and fine-structure, rather than extension and generalization, of existing logical systems can be crucial in applications is also shown by the example of computation. In contemporary Computer Science, the art has been to chip off suitable pieces from such monoliths as 'Turing-computable functions' or 'predicate-logically definable predicates'.

There is also a second theme in this book. Behind a linguistic paradigm such as Categorial Grammar, there lies a more general perspective. In the final analysis, understanding natural language is just one instance of the more general phenomenon of *information processing*. And indeed, there is an interesting convergence to be recorded nowadays between various logical paradigms directed toward the structural and procedural aspects of information, such as Relevance Logic, Modal Logic or Linear Logic. Indeed, much of the theory that we develop applies to this more general setting too, and hence what started out as a mathematical study of a grammatical theory has gradually turned into a kind of 'dynamic logic', independent from the particular encodings found in linguistic syntax. We shall try to map out this broader perspective as a natural extension of our main topic in the final Part of this book, which may in fact be read independently.

As will be clear from the preceding description, this is a book on the interface of a number of disciplines: Mathematics, Linguistics and Computer Science. What we want to present is a unifying logical perspective between these, developed in some mathematical depth, without making a very detailed examination of the various ingredients. In particular, our purpose is not to explain or motivate Categorial Grammar per se. For that, the reader may consult the anthologies Oehrle, Bach & Wheeler 1988 or Buszkowski, Marciszewski & van Benthem 1988, while van Benthem 1986 also provides some relevant background in the theory of generalized quantification. Also, there is already a number of standard references available for the Lambda Calculus as an enterprise in its own right, which need no copying here: see Barendregt 1981 or Hindley & Seldin 1986, or the earlier Gallin 1975 for more linguistically oriented manifestations. A good introduction to the formal machinery of logical semantics for natural languages is Dowty, Wall & Peters 1981. And finally, some independent basic information on various forms of Modal and

Dynamic Logic may be found in Bull & Segerberg 1984 and Harel 1984, while van Benthem 1985 supplies more extensive mathematical background.

This book has been written for an audience with a certain sophistication in mathematical logic, and non-parochial interests in investigating logical structures across various fields of application. What is needed on the part of the reader is a certain acquaintance with basic techniques from model theory and proof theory, both for classical logic and some of its variations (lambda calculus, intensional logic). The necessary 'iron rations' are surveyed in a special Appendix at the end (Part VII of this book).

The above description of contents for the present volume has emphasized a certain *research programme*. What we would like to put on the map is the perspective of a Categorial Hierarchy with an attendant mozaique of fragments of the Lambda Calculus, viewed as a tool for studying both linguistic structures and dynamic procedures. But there are also more concrete logical contributions to be reported here. The exposition to follow involves a survey of a good deal of the relevant literature supporting the preceding picture, while also presenting a number of technical logical results demonstrating its potential. Explicit credits have been given whenever available: all other results presented would seem to be new.

2 Lambda Calculus and Theory of Types

For a start, here is a short review of some basic notions and results in Lambda Calculus and Theory of Types, that will recur in the Chapters to follow.

2.1 Types, Models and Languages

Types consist of some number of *primitive types* and all those which can be formed out of them by means of a certain group of admissible operations. Common primitive types in logical semantics include

e	individual objects	('entities')
t	truth values	
s	intensional indices	('possible worlds', 'computer states', ...)

Common type-forming operations are

(a, b)	function type	('from a-arguments to b-values')
a•b	product type	('ordered pairs')

In these lectures, the above primitive types will often be used for the sake of concrete illustration: but, most of the theory presented is perfectly general. As for operations, we shall work mainly with function types - although product types are very convenient when it comes to practice.

Next, we introduce semantic structures. A *standard model* is a family of domains D_a , one for each type a , such that, for primitive a , D_a is some arbitrary set (pending further stipulations) and, for complex a ,

$$D_{(a,b)} \quad = \quad {D_b}^{D_a} \qquad \text{(function space)}$$

$$D_{a\bullet b} \quad = \quad D_a \times D_b \qquad \text{(Cartesian product)}$$

Now, in order to refer to objects in such structures, we need a suitable *language*. The simplest case is that of a 'lambda calculus', having enough variables (and when desired, also constants) x_a of each type a , as well as the following operations for constructing complex terms:

Application If σ , τ are terms of types (a, b) , a respectively, then $\sigma(\tau)$ is a term of type b

Abstraction If σ is a term of type b , x a variable of type a , then $\lambda x \bullet \sigma$ is a term of type (a, b)

We shall be rather carefree in notation for terms henceforth, suppressing brackets whenever convenient.

Remark. Notation.
Notations for types and lambda terms tend to vary across different research communities. For instance, functional types are sometimes written with arrows:

$(a \to b)$,

and another well-known notation for application of terms is simple juxtaposition:

$\sigma\tau$.

Our particular choice in this book reflects the Montague-Gallin tradition.
The latter will tend to generate large clusters of commas and brackets if pursued consistently, so that the reader is urged to use any ad-hoc abbreviations that might appeal to her. For instance, in many contexts, the basic type (e, (e, t)) of 'two-place predicates of individuals' might be shortened to just

eet .

Likewise, an important type like that of 'determiners' (see Chapter 3), whose official notation reads ((e, t), ((e, t), t)) , might be abbreviated to the more convenient

(et) (et) t .

With product types also present, we may add suitable pairing and projection functions to the language:

Pairing If σ, τ are terms of types a, b respectively,
then $<\sigma , \tau>$ is a term of type $a \bullet b$

Projection If σ is a term of type $a \bullet b$,
then $\pi_L(\sigma)$, $\pi_R(\sigma)$ are terms of types a , b , respectively.

In the language so far, one can form complex descriptions of functions ('procedures'), ascending to a meta-level in order to compare them as to equivalence, etcetera. But, we can also add the basic comparative predicate itself to the object language, being the *identity*:

If σ , τ are terms of the same type,
then $\sigma=\tau$ is a term of the truth value type t .

In the resulting language L_ω , a special primitive type t becomes distinguished.
If enough assumptions are made about the truth value domain, then the usual logical operators can be defined in this 'type theory': notably, the Boolean *connectives* and the

universal and existential *quantifiers* (see Henkin 1963, Gallin 1975 or Lambek & Scott 1986 for full details). For instance, here are some simple equivalences:

T	$\lambda x_t \bullet x_t = \lambda x_t \bullet x_t$	('true')
$\bot$	$\lambda x_t \bullet x_t = \lambda x_t \bullet T$	('false')
$\neg\phi$	$\phi = \bot$	(negation)
$\phi \wedge \psi$	$\lambda x_{(t,(t,t))} \bullet x(\phi)(\psi) = \lambda x_{(t,(t,t))} \bullet x(T)(T)$	(conjunction)
$\forall x_a\, \phi$	$\lambda x_a \bullet \phi = \lambda x_a \bullet T$	(universal quantifier)

Conversely, the new language may also be set up from the start as a higher-order calculus with the usual logical constants

$$\neg\,,\ \wedge\,,\ \vee\,,\ \rightarrow\,,\ \forall x_a\,,\ \exists x_a\,,\ =\,.$$

(Henkin 1950 even adds such further amenities as a *iota-operator* describing objects via definite descriptions.) The resulting system may also be described as a direct generalization of first-order logic, second-order logic and so on through all finite levels.

Interpretation of these languages in the above standard models is straightforward, using auxiliary assignments to free variables in the standard manner. (An ability to supply such formalities when necessary is a typical illustration of the kind of technical facility presupposed on the part of readers of this book.) Thus, we shall assume that, in any model $\mathbb{M}$, given some assignment of suitable objects u to variables x, a term τ with free variables $x_1, .., x_n$ has an appropriate corresponding semantic value

$$[[\tau]]^{\mathbb{M}}\,[x_1, ..., x_n \,/\, u_1, ..., u_n]\,.$$

2.2 Axiomatic Calculi

Now, in an axiomatic approach to these systems, one sets up some reasonable calculus of inference rules. Notably, we have the following key principle relating application and abstraction:

Lambda Conversion

$$(\lambda x \bullet \tau)\,(\sigma) \;=\; [\sigma/x]\tau\,,$$

provided that σ is free for x in τ.

In the literature, this is sometimes called 'β-conversion'. Together with the obvious equality of alphabetic variants (known under the name of 'α-conversion'), as well as the usual rules of inference for Identity - in particular, Replacement of Identicals - one obtains the *Typed Lambda Calculus*.