Institute for Language, Logic and Information

INTUITIONISTIC CATEGORIAL GRAMMAR

Aarne Ranta

ITLI Prepublication Series
for Logic, Semantics and Philosophy of Language LP-90-04

University of Amsterdam

INTUITIONISTIC CATEGORIAL GRAMMAR

Aarne Ranta
Department of Mathematics and Computer Science
University of Amsterdam
Academy of Finland, and
Department of Philosophy
University of Helsinki
Unionkatu 40 B, 00170 Helsinki, Finland

Received April 1990

INTUITIONISTIC CATEGORIAL GRAMMAR

Contents

Acknowledgments. I am grateful to Per Martin-Löf for discussions, to Petri Mäenpää for cooperation in computer implementation, and to Johan van Benthem for comments and for his hospitality during a stay in Amsterdam in the spring of 1990, which provided me excellent conditions for finishing this paper.

1. INTRODUCTION

In a categorial grammar, expressions of a language are assigned categories, to account for their modes of combination. The category assignments also have a semantic significance: expressions of any category denote objects of a definite type. Thus the categorial grammar does not merely give the syntax of the language, but its meaning structure as well.

In the study of natural languages, the categorial grammars most widely employed are based on the simple type theory with two basic types, the type e of individuals and the type t of truth values. Proper names are, most straightforwardly, assigned the category e, i.e. they are explained as names of individuals. Sentences are assigned the category t. Common nouns, adjectives, and intransitive verbs are all assigned the category (e,t) of expressions denoting functions from individuals to truth values.

A grammar of this kind assigns to expressions of English the same categories as to certain expressions of classical predicate calculus. For predicate calculus, such assignments are mathematically well-established, because predicate calculus is an artificial language that can be defined by the categorial grammar. For English, categorial grammar is a research program: to the degree to which a categorial grammar can be imposed on English, the logical structure of English becomes understood.

The adequacy of the simple type theory in the study of English has been challenged for various reasons. Richer type theories have been suggested accordingly, e.g. Montague's theory with one more basic type, the type s of possible worlds. There are also approaches stepping out of categorial grammars, e.g. Kamp's discourse representation theory. It might be claimed that in the explanation of the dynamicity of natural languages, exemplified by pronominal reference, one must give up the ideal of parallelism between syntax and semantics. This doubt has recently been challenged by Groenendijk and Stokhof (1989, 1990), in attempts to see categorial structure even in dynamic phenomena.

What has been employed in the categorial grammar of natural languages is, essentially, the simple type theory and a series of additions to it, additions made in the purpose of coping with problems that have arisen. These additions have also resulted in new formalisms in which meanings of English sentences can be expressed better than in predicate calculus. But the basic ontology of entities and truth values has always been retained.

In mathematics, classical logic and its explanations of meaning have been challenged by intuitionists. One of the crucial criticisms runs as follows. We have introduced the basic type of truth values, which has exactly the two objects True and False. We have explained logical formulae as names of truth values, and this is unquestionable as long as we can tell which truth value each formula denotes. But there are formulae for which we cannot tell this, such as many of those formed by universal quantification over an infinite domain. In what sense does such an expression denote a truth value?

Intuitionistic logic does not give up universal quantification or any other mode of expression, but it explains them in a new way. We do not know how to determine the truth value of $(\forall x)A$, but we do know what is required to establish it as true: we ought to introduce a function which to each individual d establishes the truth of $A(d/x)$. In other words, we can prescribe what the _proofs_ of the proposition are like, even if we do not know any such proof.

Intuitionistic explanations often stop at this point, without assigning any denotations to formulae, in the way classical logic assigns truth values to them. But proof prescriptions can be understood as definitions of _sets_, in the usual intuitionistic sense in which sets are defined by prescriptions of elements. A formula denoting such a set is _true_ if the set has an element.

This interpretation, called the _propositions as types_ principle, was developed by Curry (Curry and Feys 1958), Howard (1980), de Bruijn (1970), and Martin-Löf (1973). A categorial grammar implementing this principle was devised by Martin-Löf as the "higher level notation" mentioned in the preface of Martin-Löf 1984, but it was only published in detail by Nordström & al. (1990). The grammar, as well as related systems by Harper & al. (1987), and by Coquand and Huet (1988) are currently employed in computer implementations of logical formalisms.

The intuitionistic type theory yields the simple type theory as a special case. As a whole, it has an expressive power exceeding the power of the simple type theory. In this paper, the theory will be used for studying the functioning of natural language. Due to the interpretation of formulae as sets of proofs rather than as truth values, the theory can make intensional distinctions. But we shall concentrate on the treatment of logical constants as set-theoretical operators, which are dynamical in a sense corresponding operators of predicate calculus are not. Taking this general view on the logical constants from the start, we need not add anything to the formalizations of single sentences to be able to capture pronominal references by them.

What is more, the English quantifier words "every", "any", "some", and the indefinite article will be unambiguously interpreted as universal or existential, which is not possible in predicate calculus, nor in Montague's intensional logic.

Intuitionistic logic has been used very little in the grammar of natural languages. An observation about Geach's donkey sentences was made by Sundholm (1986, pp. 502-503). Ahn and Kolb (1989) have interpreted Kamp's discourse representations in the type theory of Coquand and Huet. Hintikka's game-theoretical semantics (Hintikka and Kulas 1985) can be interpreted as an intuitionistic theory, by only taking effective winning strategies into account and by interpreting games as propositions and strategies as proofs (see Ranta 1988, 1990). Game-theoretical semantics might have given the first systematic account of donkey sentences (Hintikka and Carlson 1979), and since then it has made advance on various fields of semantics. Thas work has remained relatively unknown among linguists, partly because it has not been implemented in any kind of a generative grammar. But the work at hand does make some use of it in a grammar that generates English sentences from propositions of intuitionistic type theory.

One strong reason for formalizing natural language in intuitionistic logic is the direct computational content of the latter. The intuitionistic notion of proof resembles the notion of computer program in a way that has been made formally precise, so that Martin-Löf's type theory can be used as a program and specification language. The formalization of an English sentence as a proposition in Martin-Löf's theory is as such a program specification. (See Martin-Löf 1982 and Nordström & al. 1990.)

This paper presents a grammar of a fragment of English based on Martin-Löf's type theory. The structure of the grammar resembles, in a way to be described in detail, the structure of Montagues's PTQ grammar ("The proper treatment of quantification in ordinary English", Montaque 1974, chapter 8). The fragment of English we deal with does not contain modal operators, but it goes beyond the PTQ fragment in the direction of quantifier words and anaphoric expressions. An earlier version of the system of rules transforming expressions of type theory into expressions of English has been implemented on a personal computer by Mäenpää and Ranta (1989).

2. A review of logical grammars

2.1. Artificial and formal languages. To give a grammar to a language is to impose a structure on it, a grammatical structure. There are languages created by grammars, such that every expression has an intrinsic structure reflecting its creation. Such languages will be called artificial. Some artificial languages are formal in the sense that every expression has a unique grammatical structure that can be effectively read from the form of the expression.

Natural languages, such as English, are not created by grammars. The task of a grammarian of English is to impose a structure on a language that exists independently of his grammatical activity. If he is giving a generative grammar, he is in fact creating an artificial language all of whose expressions are also recognizable as expressions of English. This artificial language is called a fragment of English. Large enough fragments are not, in general, formal languages, as they contain expressions that can be generated in different ways. Such expressions are called _ambiguous_, with respect to the grammar in question. But to every artificial language corresponds a formal language, consisting of expressions of the artificial language endowed with information about their generation. For instance, to an artificial language generated by a phrase structure grammar corresponds the formal language whose expressions are phrase structure trees.

A procedure reverse to generative grammar is the one in which English is taken as it stands, with no effort to delimit a fragment of it. The grammarian gives _parsing_ rules, which assign grammatical structures to expressions of English. A given set of parsing rules cannot, in general, assign a structure to every expression. It functions, after all, only on a fragment of English, the fragment defined in the indirect way as consisting of those expressions of English that can be parsed by means of the set of rules at hand.

There is a current terminology in which the word "parsing" is reserved for artificial languages, meaning a process in which the structures of their expressions are revealed. For natural languages, structures are not, in this sense, revealed, as no structure is presupposed to exist independently of the grammarian's activity. Assignment of grammatical forms to expressions of a natural language is _formalization_ rather than parsing: forms are imposed rather than revealed.

Outputs of formalization rules are _grammatical representations_ of expressions of English. We will be concerned with cases where grammatical representations are expressions of a formal language. The formal language whose expressions the grammatical representations are will be called a _representation formalism_. The grammarian who chooses to work in the direction of formalization rather than generative grammar must of course rely on a generative grammar of the representation formalism. If the formalization procedure can be reversed, to turn grammatical representations into expressions of English, the grammar generating the formalism and the reverse procedure of formalization together constitute a generative grammar of a fragment of English.

In _logical grammar_, logical formalisms are used for grammatical representation. If we have a representation of an English sentence in a logical formalism, we have imposed on it a grammatical structure of a peculiar kind, a _logical form_. What is peculiar to a logical form or, correspondingly, what makes a formalism logical, need not be precisely defined. We shall just provide a number of examples. But subsection 4.1 will suggest a characterization of logical formalisms: they can be defined by pure categorial grammars. In other words, all of their expressions have definite type-theoretical meanings.

2.2. _Predicate calculus as a representation formalism_.

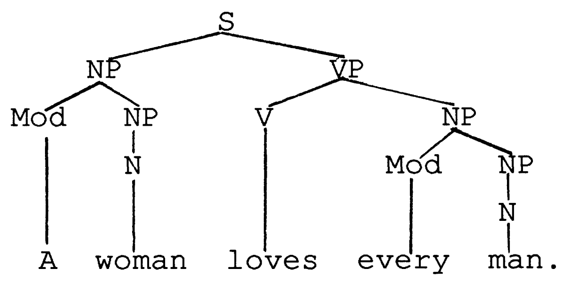
We are now considering a first order language, defined by introducing a number of predicates and individual constants, from which expressions can be formed by substitution of terms for argument places and by the logical constants $\&$, $\vee$, $\neg$, $\supset$, $\forall$, $\exists$. We assume that the language has been interpreted in a standard way. Sentences of English are now customarily represented by well-formed formulae of the formalism. Knowing the interpretation of the formalism, we know the _proposition_ expressed by each formula, i.e. its _truth condition_, and, derivatively, the truth conditions of those English sentences we manage to represent. Truth conditions open the way to other logical properties we are interested in, such as the validity of inferences made by using English sentences.

Thus we know something about the _meanings_ of those English sentences we manage to give a logical form. We know something that is not revealed by "purely syntactic" grammatical representations. For instance, the sentence

(1) A woman loves every man.

is in the formalism of phrase structure trees represented unambiguously as the tree

(2)



In predicate calculus, there are two representations, imposing two different structures on the sentence - two different propositions:

(3) $(\exists x)(\text{woman}(x)\ \&\ (\forall y)(\text{man}(y) \supset \text{love}(x,y)))$,

(4) $(\forall y)(\text{man}(y) \supset (\exists x)(\text{woman}(x)\ \&\ \text{love}(x,y)))$.

Assignment of logical formulae to English sentences can be called <u>semantics</u> of English, in a perfectly good sense, provided that the formulae themselves have been given meaning.

2.3. <u>Montague's formalisms</u>. All truth conditions cannot be expressed by formulae of first order predicate calculus. The sentence

(5) John seeks a unicorn.

has only one representation,

(6) $(\exists x)(\text{unicorn}(x)\ \&\ \text{seek}\ (\text{John},x))$.

Montague extended predicate calculus to <u>intensional logic</u>, in which reference can be made to possible worlds and, moreover, variables of arbitrary types can be bound. In addition to a formula corresponding to (6), the sentence (5) has the formalization

(7) $\text{seek}({}^\wedge\text{John},\ \hat{P}(\exists x)(\text{unicorn}_*(x)\ \&\ P\{{}^\wedge x\}))$

(Montague 1974, p. 266).

Besides intensional logic, Montague had another formalism with sufficient expressive power, where English sentences were represented by _analysis trees_. In the formalism of analysis trees the sentence (5) is formalized in the following two ways, corresponding to (6) and (7) in the same order:

(8) $F_{10,0}(F_2(\text{unicorn},F_4(\text{John},F_5(\text{seek},\text{he}_0))))$,

(9) $F_4(\text{John},F_5(\text{seek},F_2(\text{unicorn})))$.

Montague's grammar of English is generative. He first defines analysis trees, and then tells how they are transformed into English sentences whereby their unambiguous structures are destroyed. A process of the latter kind, which is the reverse of parsing, will be called _sugaring_, in accordance to current terminology of computer science. The analysis trees constitute a formal language, and the fragment of English is an artificial language obtained from analysis trees by sugaring. The language of analysis trees is, moreover, a logical formalism, in the sense that analysis trees can be effectively interpreted in a model by reference to their forms. In "The proper treatment of quantification in ordinary English", the interpretation is made via a translation to intensional logic, and in "English as a formal language"(Montague 1974, chapter 6), directly.

2.4. _Compositionality_. The importance of Montague's grammar is not only in the great expressive power of his formalisms. It is even more in the detailed account he gave of the relation between his fragment of English and the formalisms. He almost indicated that he had given formalization rules from English into intensional logic:

It is understood that each English sentence listed below translates into some formula logically equivalent to each of the one or more formulas of intensional logic listed with it, and that every formula into which the

English sentence translates is logically equivalent to one of those formulas. It should be emphasized that this is not a matter of vague intuition, as in elementary logic courses, but an assertion to which we have assigned exact significance in preceding sections and which can be rigorously proved.
(Montague 1974, pp. 265-266.)

But he did not give any formalization rules for plain English, nor any rules for parsing expressions of the fragment into analysis trees, but only rules for sugaring analysis trees into English and rules for translating analysis trees into intensional logic. Analysis trees constitute the formal language from whose expressions the fragment is obtained by sugaring. The following figure shows the structure of Montague's PTQ grammar.

$$\begin{tikzcd}
\text{analysis trees} \arrow[r, "\text{sugaring}"] \arrow[d, "\text{translation}"] & \text{English} \\
\text{intensional logic} &
\end{tikzcd}$$

The language of analysis trees is designed in such a way that it can be straightforwardly sugared into plain English. In a sense, one could say that analysis trees are built from English words by using parentheses - although the word "every" is replaced by "F_0", etc, and although there are different signs for concatenation, e.g. F_5, F_6, F_7. These signs are necessary in order to make analysis trees reveal their logical forms. But each sign has a straightforward English reading, unlike the signs of intensional logic.

It is often said that Montague's grammar is compositional, which is explained as meaning something like that each expression of English has a definite translation in intensional logic, and that the translation of each complex expression is determined by the translations of its parts and the way they are composed into the complex. But this explanation - "its parts", "the way they are composed" - refers to analysis trees instead of plain English, and makes compositionality trivial. It does not hold of composition by simple concatenation of words, nor of composition formalized by ordinary phrase structure trees, as shown by the example sentence (1).

Compositionality in the above sense is not always trivial if two distinct formalisms are compared, one "syntactic" and the other "logical". But he crucial invention of Montague was of course the proper design of the analysis trees, so that they were both close to plain English and explicit in logical form.

2.5. Dynamic representation formalisms. Consider the extension of the PTQ fragment to conditional sentences, represented by means of the implication of intensional logic. The sentence

(10) If a man walks he talks.

cannot be treated satisfactorily. For there is no analysis tree that is both sugared into (10) and translated into a formula of intensional logic equivalent to

(11) $(\forall x)(man(x)\ \&\ walk(x) \supset talk(x))$.

The sentence (10) contains the indefinite article "a", which is the plain English counterpart of "F_2". But when analysis trees are translated into intensional logic, "F_2" is translated into something that contains "$\exists$" but not "$\forall$".

The formalization of (10) as (11) is regarded as a violation of compositionality for a derivative reason. Sometimes both the universal quantifier and the existential quantifier must be used for formalizing occurrences of the indefinite article in one and the same sentence:

(12) If a man walks he finds a pen.

is formalized as

(13) $(\forall x)(\mathrm{man}(x) \mathbin{\&} \mathrm{walk}(x) \supset (\exists y)(\mathrm{pen}(y) \mathbin{\&} \mathrm{find}(x,y)))$.

Thus there is no possibility to define analysis trees in such a way that the indefinite article is uniquely represented, and the translation into intensional logic is determined by taking the constituents of analysis trees one by one.

Kamp (1981) was able to present a system of parsing rules that assign formulae of predicate calculus to sentences of a fragment of English, e.g. the formula (13) to the sentence (12). He used an intermediate level of _discourse representations_, which constitute a formalism whose semantics is given by a translation into predicate calculus. Thus the indefinite article is, essentially, interpreted now as the existential quantifier, now as the universal quantifier. Groenendijk and Stokhof (1989, p. 4) pointed out that Kamp's theory is not compositional in the sense required by the Montague grammarian. They went on by developing a formalism they called _dynamic predicate logic_, to represent grammatical structures of English sentences and texts.

The syntax of dynamic predicate logic differs from the syntax of ordinary predicate calculus only with respect to what occurrences of variables are regarded as bound. The interpretation deviates accordingly from the interpretation of standard predicate calculus, being essentially more complex. The operators & and $\supset$ are explained as _internally dynamic_, which means that & can "pass on variable bindings from its left conjunct to the right one" (op. cit., p. 8), and $\supset$ "passes on values assigned to variables in its antecedent to its consequent" (p. 9). & is _externally dynamic_ as well, "because of its capacity to keep passing on bindings to conjuncts yet to come" (p. 8); and so is $\exists$, which "can bind variables...outside its scope" (p. 9). Finally, $\exists$ and $\forall$ are internally dynamic already in standard predicate calculus, as they bind variable occurrences inside their scopes. v and $\neg$ are neither internally nor externally dynamic; $\supset$ and $\forall$ are not externally dynamic. Without going into the details about the precise mathematical formulation of all this, we can give as examples the formulae (14) and (15), which are now available as formalizations of the sentences (10) and (12), respectively:

(14) $(\exists x)(\text{man}(x)\ \&\ \text{walk}(x)) \supset \text{talk}(x)$,

(15) $(\exists x)(\text{man}(x)\ \&\ \text{walk}(x)) \supset (\exists y)(\text{pen}(y)\ \&\ \text{find}(x,y))$.

In contrast to these formulae, the last occurrence of x is not bound in

(16) $(\forall x)(\text{man}(x) \supset \text{walk}(x)) \supset \text{talk}(x)$,

since $\supset$ and $\forall$ are not externally dynamic. This accounts for the absence of anaphoric relation in

(17) If every man walks he talks.

The operators & and $\exists$ are contrasted to $\supset$ and $\forall$ in the purpose of assigning intuitively correct truth conditions to English sentences. Why & should have the power to pass on bindings to its right while $\supset$ should not, is not explained. What is more, there are English sentences such that something like externally dynamic implication is needed in their logical representation:

(18) If every man finds a pen, some man will lose it again.

needs something like

(19) $(\forall x)(\text{man}(x) \supset (\exists y)(\text{pen}(y)\ \&\ \text{find}(x,y)))$
$\supset (\exists z)(\text{man}(z)\ \&\ \text{lose}(z,?y))$.

(Cf. Hintikka and Kulas 1985, p. 101.)

The dynamic interpretation of formulae which, after all, do not differ from formulae of ordinary predicate calculus, lays a lot of weight on the metalanguage, where the dynamic meanings of logical constants are spelled out by using static metalogic. It seems urgent to design a formalism that reflects the intuitive ideas of dynamicity more directly.

3. Grammatical representations in intuitionistic type theory

In this section we shall take an informal look into Martin-Löf's type theory and its use in the grammatical representation of English. Section 4 will provide a formal presentation of type theory, and in section 5 a grammar of a fragment of English will be described in more detail.

In this section, _type theory_ will mean the _lower-level_ type theory to be distinguished from the _higher-level_ type theory in later sections. "Intuitionistic type theory", in the sense of Martin-Löf 1973, 1982, 1984, is precisely this lower-level theory. Making no reference to types, but only to sets, it is on the same level as predicate calculus, whereas the higher level type theory is on the same level as the simple type theory.

3.1. _The quantifiers Σ and Π_. Just like predicate calculus, intuitionistic type theory has expressions for propositions. We shall consider two forms of complex propositions,

$$(\Sigma x:A)B \text{ and } (\Pi x:A)B.$$

They correspond to $(\exists x)B$ and $(\forall x)B$, respectively, of predicate calculus interpreted in the domain A. The difference is that type theory makes the domain explicit. Type theory is thus more _formal_ than predicate calculus, where the domain indication cannot be read from the form of a quantified expression but only from its interpretation.

Type theory is more formal than predicate calculus also in making explicit _judgments_ (or _assertions_). Judgments are wider in scope than propositions. A proposition may occur as a part of a judgment, but not conversely. There are judgments of the following forms:

Judgment	where	means that
A:prop	-	A is a proposition
A=B:prop	A:prop, B:prop	A and B are equal propositions
a:A	A:prop	a is a proof of A
a=b:A	A:prop, a:A, b:A	a and b are equal proofs of A

Frege also had a notion of judgment, as contrasted to propositions. He used the sign "$\vdash$" in front of propositions judged true (see <u>Begriffsschrift</u>, §2). In contemporary logic, judgments are often only made in the metalanguage, e.g. judgments of form "A is a well-formed formula" corresponding to the form A:prop, and the form "A is true" corresponding to Frege's $\vdash$A. In intuitionistic type theory, $\vdash$A is explained in terms of a:A.

Judgments of form $\vdash$A, rather than bare propositions, correspond to English indicative sentences, whose propositional contents may occur in sentences of other moods, such as questions. But as the present fragment will only comprise indicative sentences, we shall often speak of propositions as formalizations of English sentences. For the formalization of sequences of sentences, however, proofs must be made explicit; see subsection 5.9.

The operators Σ and Π are used in the formalization of English sentences much in the same way as the existential and the universal quantifier, respectively, of predicate calculus. The explicit indication of domains of quantification makes it possible to avoid the use of connectives without any counterparts in plain English, but only regulated by the choice of the quantifier:

(20) A man runs.

will be formalized

(21) $(\Sigma x:\text{man})\text{run}(x)$,

which is to be compared to the formula of predicate calculus,

(22) $(\exists x)(\mathrm{man}(x)\ \&\ \mathrm{run}(x))$.

It is of course possible to start with a big domain, say D, out of which other domains are separated by predicates. Instead of (21), we would then have

(23) $(\Sigma x{:}D)(\mathrm{man}(x)\ \&\ \mathrm{run}(x))$,

but (21) is preferable from the compositional point of view (cf. ((30) to (32) in subsection 3.4). Such a form is also needed in the proper formalization of quantification with "most"; see Sundholm 1989. (24) will be formalized as (25):

(24) Every man runs.
(25) $(\Pi x{:}\mathrm{man})\mathrm{run}(x)$.

In subsection 5.5, a simple rule (Q) will be given for the sugaring of quantified propositions of intuitionistic type theory into sentences of plain English, by the substitution of a quantifier phrase for the variable bound by the quantifier. For instance, in sugaring the proposition $(\Sigma x{:}\mathrm{man})\mathrm{walk}(x)$, "a man" is substituted for "x" in "walk(x)". For the substitution, one of the terms in which the variable occurs must be sorted out as the main argument, but as long as we only discuss formulae with precisely one occurrence of the bound variable, we need not worry about the precise definition of the main argument.

By means of the rule (Q) and rules for sugaring atomic propositions, we can already derive some sentences of English, e.g. (20) from (21). Take an example with two quantifiers. The sign ">" between two expressions is read "can be sugared into".

(26) $(\Pi x{:}\mathrm{man})(\Sigma y{:}\mathrm{donkey})\mathrm{own}(x,y$
> Every man owns a donkey.

3.2. <u>The propositions as types principle</u>. We have interpreted English common nouns as expressions for sets. But the significance of sets is much more general, as <u>propositions</u> are identified with sets. A proposition is a set of <u>proofs</u>, and it is defined to be <u>true</u> if it is nonempty, i.e. if there is a proof.

Thus we have the two forms of judgment "A:set" and "A:prop" as heuristic variants of each other. So are "A=B:set" and "A=B:prop". We also have two readings for any judgment of form "a:A", switching between the pairs of terms "set"-"element" and "proposition"-"proof":

a is an element of the set A;
a is a proof of the proposition A.

The form "a=b:A" analogously has two readings. The form of judgment

A true,

corresponding to Frege's "$\vdash$A", is introduced as an abbreviation, by suppression of the proof, of the form "a:A".

It is usual in logical semantics to look upon common nouns as expressions for sets and upon sentences as expressions for propositions; but it is not usual to look upon common nouns as expressions for propositions or upon sentences as expressions for sets. However, there is a systematic way to switch from a common noun to a sentence that is true if the set that the common noun denotes has an element, by prefixing "there is" plus the indefinite article to the noun. From a sentence to a common noun denoting the set of its proofs, we cannot switch in such a uniform way. But we often have a verbal noun, such as "jump of Bill's" for "Bill jumps". In the sugaring procedure to be presented, we shall need a mechanism for turning nouns into sentences, but no mechanism in the opposite direction.

3.3. The meanings of Σ and Π. To explain the meaning of a propositional expression you must tell what set it denotes. You must be able to prescribe what it is to be an element of the set. For instance, the meaning of the expression "natural number" is explained by prescribing the natural numbers to be 0, s(0), s(s(0)), etc.

To explain an operator that forms complex sets and propositions, you must prescribe the elements or proofs of the complex under the assumption that its constituents have been explained. We shall give the explanations following this pattern for Σ and Π, which have up to now only been characterized as the existential and the universal quantifier, respectively.

We write

$$J\ (x{:}A),$$

where J can be a judgment of any form and contain free occurrences of x, for a hypothetical judgment, the judgment J made under the hypothesis x:A. The substitution of a:A for x in J yields a judgment not dependent on the hypothesis x:A. We write (a/x) to indicate the substitution of "a" for allowable occurrences of "x" in an expression of type theory. Allowable occurrences must be free, and free occurrences of variables in "a" must not become bound.

Given A:set and B:prop (x:A), $(\Sigma x{:}A)B$ is the set of pairs (a,b) where a:A and b:B(a/x). Given c:$(\Sigma x{:}A)B$, we can form the projections p(c):A and q(c):B(p(c)/x), which for c of pair form are computed in an obvious way:

$$p((a,b))=a{:}A \text{ for } a{:}A,\ b{:}B(a/x);$$
$$q((a,b))=b{:}B(a/x) \text{ for } a{:}A,\ b{:}B(a/x).$$

Given A:set and B:prop (x:A), $(\Pi x{:}A)B$ is the set of λ-abstracts $(\lambda x)b$ of functions b:B (x:A). Given c:$(\Pi x{:}A)B$ and a:A, we have the application instance ap(c,a):B(a/x), which for c in λ-abstract form is computed as follows:

$$ap((\lambda x)b,a)=b(a/x) \text{ for } b{:}B\ (x{:}A),\ a{:}A.$$

Heyting's (1956 pp. 102-107) explanations of the logical constants &, $\supset$, $\exists$, $\forall$, which have become standard in intuitionistic mathematics, can be summarized in the following way:

> A proof of A & B consists of a proof of A and a proof of B;
>
> a proof of A $\supset$ B consists of a method that assigns a proof of B to any given proof of A;
>
> a proof of $(\exists x)$B consists of an individual and a proof of the corresponding instance of the propositional function B;
>
> a proof of $(\forall x)$B consists of a method which to each individual assigns a proof of the corresponding instance of the propositional function B.

The identification of propositions and sets makes it unnecessary to have separate propositional operators & and $\exists$, which can be both regarded as versions of Σ, switching between the heuristic words "set" and "proposition". In a similar way, $\supset$ and $\forall$ are both variants of Π. The conjunction and implication defined in this way are stronger than & and $\supset$, respectively, as the second constituent may depend on the proof of the first one. This dependence makes the intuitionistic type-theoretical conjunction and implication "internally dynamic" in the same way as quantifiers are. Since we can consider the proofs of propositions of any form, all logical constants are "externally dynamic".

3.4. <u>More English readings for Σ and Π</u>. In section 5, we shall define operators for sugaring propositional expressions of type theory into noun-like and sentence-like expressions of English. All propositional expressions that can be sugared into nouns can also be sugared into sentences by prefixing "there is" and the indefinite article to the noun.

Reading A:set and B:prop (x:A), $(\Sigma x{:}A)B$ is the set of pairs consisting of elements of A paired with proofs of corresponding instances of B, i.e. elements of A <u>such that</u> B is true of them. Such a set expression can be sugared into a noun modified by a <u>relative clause</u>, by means of the rule (R) to be given in subsection 5.5.

Thus we have e.g.

(27) $(\Sigma x{:}\mathrm{man})\mathrm{run}(x)$ > man who runs;

(28) $(\Sigma x{:}\mathrm{man})(\Sigma y{:}\mathrm{donkey})\mathrm{own}(x,y)$
> man who owns a donkey;

(29) $(\Sigma y{:}\mathrm{donkey})(\Sigma x{:}\mathrm{man})\mathrm{own}(x,y)$
> donkey that a man owns.

Having the possibility to quantify over Σ-sets makes it possible to represent relative clauses in a uniform way, which is not possible in predicate calculus. Geach (1972, pp. 494-497) discussed the sentence

(30) Any man who hurts any man who hurts him hurts himself.

whose relation to its formalization in predicate calculus,

(31) $\forall x(\mathrm{man}(x)\supset(\forall y(\mathrm{man}(y)\supset(\mathrm{hurt}(x,y)\supset\mathrm{hurt}(y,x))\supset\mathrm{hurt}(x,x))$,

is hard to explain in a compositional way. But its formalization in type theory,

(32) $(\Pi x{:}(\Sigma y{:}\mathrm{man})(\Pi z{:}(\Sigma u{:}\mathrm{man})\mathrm{hurt}(u,y))\mathrm{hurt}(y,p(z)))\mathrm{hurt}(p(x),p(x))$,

can be sugared into the English sentence by applying to the quantifiers, from left to right, the rules (Q), (R), (Q), (R).

If we treat Σ and Π as <u>connectives</u> rather than as quantifiers, we can sugar $(\Sigma x{:}A)B$ and $(\Pi x{:}A)B$ into sentences where both A and B are sugared into sentences. By means of the rule (C) of subsection 5.6, $(\Sigma x{:}A)B)$ is sugared into the conjunction of A and B, and $(\Pi x{:}A)B)$ into the conditional sentence where A is sugared into the antecedent and B into the succedent. For instance,

(33) $(\Sigma x{:}\mathrm{man})\mathrm{walk}(x)$ > There is a man and he walks.
(34) $(\Pi x{:}\mathrm{walk}(\mathrm{John}))\mathrm{talk}(\mathrm{John})$ > If John walks he talks

We shall have numerous alternative English expressions corresponding to given propositions of type theory, generated by the rules (Q), (R), and (C). In the example (26), we sugared the propositional expression $(\Pi x{:}\mathrm{man})(\Sigma y{:}\mathrm{donkey})\mathrm{own}(x,y)$ into the English sentence "every man owns a donkey" by applying the rule (Q) twice. The following variants are also derivable:

(35) If there is a man he owns a donkey. By (C), (Q).
(36) If there is a man there is a donkey that he owns.
By (C), (R).
(37) If there is a man there is a donkey and he owns it.
By (C), (C).

To make these sugarings completely mechanical, we will have to give rules for sugaring singular terms into pronouns. At this stage, we only notice that in (35) to (37), the variable x of type man has been sugared into "he", and in (37), the variable y of type donkey has been sugared into "it". We also notice that even in notoriously problematic cases, we do have obvious terms in the formalism which can be sugared into pronouns of English. For instance, the sentence

(38) If a man owns a donkey he beats it.

will result from sugaring

(39) $(\Pi x{:}(\Sigma y{:}\mathrm{man})(\Sigma z{:}\mathrm{donkey})\mathrm{own}(y,z))\mathrm{beat}(p(x),p(q(x)))$

where $p(x)$ is of type man and $p(q(x))$ is of type donkey.

More details are also concealed in the sugaring of Σ and Π into quantifier words. In the sugaring of (39) into (38), Π is turned into "if" and Σ is twice turned into "a", in accordance to the rules (C) and (Q), respectively. Applying (Q) to Π, (R) to the first Σ, and (Q) to the second Σ in (39), we obtain

(40) Every man who owns a donkey beats it.

The original example of Geach (1962, p. 117) reads "any" instead of "every", which alternative is also provided by the rule (Q). But (39) cannot be sugared into the intuitively equivalent sentence

(41) If any man owns a donkey he beats it.

because Σ cannot be sugared into "any". By treating "any" as a universal quantifier word with a wider scope than "if", we obtain (41) from

(42) $(\Pi x{:}\mathrm{man})(\Pi y{:}(\Sigma z{:}\mathrm{donkey})\mathrm{own}(x,z))\mathrm{beat}(x,p(y))$,

which is equivalent to (39), but not intensionally equal (i.e. the same proposition, in the sense of a set of proofs).

Finally, observe that we can form the proposition

(43) $(\Pi x{:}(\Pi y{:}\mathrm{man})(\Sigma z{:}\mathrm{pen})\mathrm{find}(y,z))(\Sigma u{:}\mathrm{man})\mathrm{lose}(u,p(\mathrm{ap}(x,u)))$,

which can be sugared, by (C), (Q), (Q), (Q), into

(44) If every man finds a pen some man loses it.

This sentence cannot be treated in dynamic predicate logic (nor in discourse representation theory), where conditionals are externally static. Their staticity is inferred from sentences such as (17), "If every man walks he talks", which cannot be derived in our theory either, but now for the reason that a proof of $(\Pi x{:}\mathrm{man})\mathrm{walk}(x)$ is of wrong type: it is not a man, but a function on the set of men.

4. <u>Categorial grammar for intuitionistic type theory</u>

4.1. <u>Pure categorial grammar for predicate calculus</u>.

Type theory, in the sense of Russell and Church, is a general framework in which various lower-level theories can be presented. In the simple type theory in which classical predicate calculus can be presented, there is a hierarchy of types built on the <u>basic types</u> e and t by forming <u>function types</u> (α,β) where α and β are types. An object of the function type (α,β) can be <u>applied</u> to objects of type α to yield objects of type β.

Objects of type e are called <u>individuals</u>, objects of type t <u>truth values</u>. Objects of the function type (e,t) are also called <u>propositional functions</u>. We write

$$a:\alpha$$

for the <u>type assignment</u>

a is an object of type α.

Observe the terminology: "object of type α" vs. "element of the set A".

Logical constants can now be introduced by type assignments in this hierarchy:

$$\&:(t,(t,t)),$$
$$\supset:(t,(t,t)),$$
$$\exists:((e,t),t),$$
$$\forall:((e,t),t).$$

Working in a given hierarchy of types, if new constants are introduced they must be explained by telling what objects they denote. For instance, the type t may have been defined as having exactly two objects,

True:t, False:t.

Now that we have introduced $\&:(t,(t,t))$ we must explain what function it is, i.e. how it yields values for its arguments. This can be done by enumerating all possible cases. We write $c(a)$ for the application of $c:(\alpha,\beta)$ to $a:\alpha$, and $c(a,b)$ instead of $(c(a))(b)$ for $c:(\alpha,(\beta,\gamma))$, $a:\alpha$, $b:\beta$.

$$\&(\mathrm{True},\mathrm{True})=\mathrm{True},$$
$$\&(\mathrm{True},\mathrm{False})=\mathrm{False},$$
$$\&(\mathrm{False},\mathrm{True})=\mathrm{False},$$
$$\&(\mathrm{False},\mathrm{False})=\mathrm{False}.$$

To explain $\forall:((e,t),t)$, we cannot generally go through all possible cases, as e may be infinite. In classical logic, $\forall$ is explained as follows:

$$\forall(P)=\mathrm{True} \text{ if } P(a)=\mathrm{True} \text{ for every } a:e,$$
$$\forall(P)=\mathrm{False} \text{ if } P(a)=\mathrm{False} \text{ for some } a:e.$$

The function $\forall$ thus differs from the function & by not being effectively computable: its evaluation may demand that we run through infinitely many instances.

The definition of a language in a type theory establishes it as an interpreted formalism. When we introduce a term, e.g. the term "&" in the type assignment

$$\&:(t,(t,t)),$$

we not only determine its syntactic behavior in the formation of complex expressions, but we also use it as a name of an object. Type assignments of the form "$a:\alpha$" have both the _type-theoretical_, or _ontological_, reading

a is an object of type α

and the _grammatical_, or _linguistic_, reading "a" is a name of an object of type α.

Moreover, the _semantics_ of the term "a" is determined by the introduction of it in a type assignment:

"a" denotes a.

Certain types have idiomatic ontological and linguistic names. For instance, the type (t,(t,t)) is called ontologically the _type of two-place truth functions_, and linguistically the _category of two-place connectives_.

In mathematical logic, the ontological and linguistic readings are not always treated in the way we did above, as different viewpoints on one and the same theory, but as separate theories. There is a _syntax_ yielding what we called grammatical clauses, such as

(i) & is a two-place connective,

which introduces the expression "&" by only stating its syntactic properties. On the other hand, in _model theory_, there is a conjunction essentially defined as an operation on truth values,

(ii) *:(t,(t,t)).

Expressions introduced in the syntax are related to the model theory by _interpreting_ them, by employing an interpretation operator T which to each expression produced by the syntax assigns an object of some type. For instance,

(iii) T(A&B)=T(A)*T(B).

The content of (i)-(iii), as regards a language not containing "*" but only "&", is expressed in our theory with the single clause

&:(t,(t,t)).

A categorial grammar of this kind, resulting from the linguistic reading of a type theory, could be called a pure categorial grammar. It is obvious that on some level in actual theorizing some expressions are to be looked upon as meaningful, and if these meanings are clear enough the expressions can be introduced in a pure categorial grammar.

Type theories and pure categorial grammars are widely used in computer science as frameworks for defining logics to be used in computerized reasoning. Frameworks related to the one of Martin-Löf, which will be described in the next subsection, are the Edinburgh LCF (Harper & al., 1987) and the theory of constructions by Coquand and Huet (1988). This tradition of categorial grammars differs from most of the current approaches in linguistics, where the grammars used are not pure, but the focus is on the syntactic combination of expressions, without strictly parallel explanations of meaning (see e.g. the calculus of Lambek 1958, which has been given semantics in van Benthem 1988).

4.2. Martin-Löf's type hierarchy. The formal language of intuitionistic type theory used in section 3 cannot be defined in the simple type theory of the previous subsection. A richer type theory is needed, in which not only function types (α,β) can be formed, but also generalized function types $(x:\alpha)\beta$, where β is a type depending on $x:\alpha$. This theory will yield the simple type theory as a special case, where all types are independent.

When completely formalized, so as to carry enough information for computer implementations of logical formalisms, both in the simple type theory and in the generalized type theory judgments of four forms must be made. Judgments of three of these forms have presuppositions, which means that they make sense only if the presuppositions are fulfilled:

Judgment	which presupposes	means that
α:type	-	α is a type
$\alpha=\beta$:type	α:type, β:type	α and β are equal types
a:α	α:type	a is an object of α
a=b:α	α:type, a:α, b:α	a and b are equal objects of α

Judgments of each of these forms can be made <u>under hypotheses</u>, which are type assignments to <u>variables</u>. A variable can be any expression whose meaning has not been explained. <u>Contexts</u> are sequences of hypotheses, of the form

(*) $x_1:\alpha_1, \ldots, x_n:\alpha_n$.

When a judgment J is made in the context (*), the variables $x_1, \ldots, x_n$ may occur <u>free</u> in J. (Occurrences of the variable x are <u>bound</u> in "β" in expressions of form "$(x:\alpha)\beta$", and in "b" in expressions of form "(x)b"; elsewhere they are free.)

In simple type theory, all types are constant, which means that only judgments of forms "a:α" and "a=b:α" are ever made under hypotheses. As this restriction is given up, contexts can generally be progressive, in the sense that in (*) each type α_k depends on $x_1:\alpha_1, \ldots, x_{k-1}:\alpha_{k-1}$. I.e. we do not have

$$\alpha_k\text{:type},$$

but only

$$\alpha_k\text{:type } (x_1:\alpha_1, \ldots, x_{k-1}:\alpha_{k-1}).$$

If a judgment J is made in the context (*), and constants $a_1:\alpha_1, \ldots, a_n:\alpha_n(a_1,\ldots,a_{n-1}/x_1,\ldots,x_{n-1})$ are substituted for free occurrences of the variables $x_1, \ldots, x_n$, respectively, in J, a judgment not depending on the context (*) results.

For details about contexts and substitution, as well as rules for equality judgments, we refer to Nordström & al. 1990. We shall only present five rules of type theory, in the natural deduction form where only those parts of contexts are made explicit which may be discharged when the rules are applied. The five rules are the formation of function types, the application of objects of function types to arguments, the abstraction of objects of function types from hypothetical judgments, the ß-conversion, and the η-conversion:

$$\frac{\alpha:\text{type} \quad \beta:\text{type}\ (x:\alpha)}{(x:\alpha)\beta:\text{type}}\ \text{form.} \qquad \frac{c:(x:\alpha)\beta \quad a:\alpha}{c(a):\beta(a/x)}\ \text{appl.}$$

$$\frac{b:\beta\ (x:\alpha)}{(x)b:(x:\alpha)\beta}\ \text{abstr.} \qquad \frac{b:\beta\ (x:\alpha) \quad a:\alpha}{((x)b)(a)=b(a/x):\beta(a/x)}\ \beta\text{-conv.}$$

$$\frac{c:(x:\alpha)\beta}{c=(x)(c(x):(x:\alpha)\beta}\ \eta\text{-conv.}$$

The rules of the simple type theory are like the ones above, with the omissions of the hypothesis x:α in the formation rule, and of the substitutions (a/x) in the application rule and in "ß(a/x)" in the ß-conversion rule. One can write (α,ß) or, as we shall do,

(α)ß instead of (x:α)ß when ß:type does not depend on x:α.

We also write

(x:α)(y:ß)γ	instead of (x:α)((y:ß)γ)	for repeated formation;
(x)(y)b	instead of (x)((y)b)	for repeated abstraction;
c(a,b)	instead of (c(a))(b)	for repeated application.

4.3. Types and sets. The notion of type, as employed in subsection 4.2, is more general than the notion of set, as employed in section 3. To introduce a set, a limited number of canonical forms - also called data forms - must be described, such that the elements of the set are precisely those having one of the canonical forms. For instance, elements of the set N of natural numbers have one of the forms 0 and s(a); elements of $(\Sigma x:A)B$ have the form (a,b); elements of $(\Pi x:A)B$ have the form $(\lambda x)b$. Elements not in canonical form of a set must be effectively computable into canonical form; noncanonical forms are also called program forms.

To introduce a type, you only have to explain what it is to be an object of it (and what it is for its objects to be equal), but you need not prescribe canonical forms. For instance, the application rule can be used for explaining what it is to be an object of function type $(x:\alpha)\beta$: it is to be something that can be applied to any object a of α to yield an object of $\beta(a/x)$. But there is no specific form such an object must have: the abstract form (x)b is just one possibility.

Now we do know what it is to be a set - a set is something that is defined by a prescription of canonical forms - but we do not prescribe canonical forms sets must have. Σ and Π are just two possibilities. Hence we have the type of sets,

set:type,

but the type set is not a set itself.

On the other hand, every set is a type, whose objects are the elements of the set, because if we know the canonical forms of the elements of a set, we know a fortiori what it is to be an element:

$$\frac{A:\text{set}}{A:\text{type}}$$

The converse of this rule is, of course, not valid.

The four forms of judgment employed on the level of sets (see subsection 3.1) are now obtained as instances of the forms "a:α" and "a=b:α" on the level of types. "A:set" and "a:A" are of form "a:α", and "A=B:set" and "a=b:A" are of form "a=b:α".

In Martin-Löf 1982, only the level of sets was employed, but sets were called types. Martin-Löf 1984 (pp.21-23) introduced the present distinction. Instead of "type", he used the word "category" then. The propositions as types principle would be more properly called the propositions as sets principle.

4.4. The higher-level definitions of Σ and Π. By the propositions as types principle, the type prop of propositions can be introduced, as equal to the type of sets:

prop:type,
prop=set:type.

Because of this identification, the different uses made of "set" and "prop" are only heuristic.

Σ is an operator that takes as its arguments a set and a propositional function defined on that set, and yields a proposition:

Σ:(X:set)((X)prop)prop.

The higher level syntax of a set expression whose outermost form is Σ is thus

Σ(A,B),

where A:set and B:(A)prop. But we shall often follow the convention of writing

(Σx:A)B(x)

instead.

To form an element of $\Sigma(A,B)$ by means of pairing, we need an element a:A and a proof of B(a):

$$\mathrm{pair}:(X:\mathrm{set})(Y:(X)\mathrm{prop})(x:X)(Y(x))\Sigma(X,Y).$$

The form of a pair, according to this, is

$$\mathrm{pair}(A,B,a,b),$$

where A:set, B:(A)prop, a:A, and b:B(a). It differs from the form

$$(a,b)$$

employed in section 3 by containing type information. The version of the lower level type theory defined by higher level type assignments is called <u>monomorphic</u>, as the form of any closed term determines its type uniquely. The version of section 3 is called <u>polymorphic</u>. We shall continue using polymorphic terms, obtained from monomorphic ones by suppression of type information.

The projections p and q, which from proofs $c:\Sigma(A,B)$ produce elements of A and proofs of B(p(c)), respectively, are introduced as follows:

$$p:(X:\mathrm{set})(Y:(X)\mathrm{prop})(z:\Sigma(X,Y))X;$$
$$q:(X:\mathrm{set})(Y:(X)\mathrm{prop})(z:\Sigma(X,Y))Y(p(X,Y,z)).$$

As the operator pair is taken as definitory for propositions of Σ form, i.e. it is a canonical form, it need not be explained further. But as the elements that p and q produce are not canonical, we must justify these type assignments by telling how p and q are computed for their arguments, the third argument assumed to be in canonical form:

$$p(X,Y,\mathrm{pair}(X,Y,x,y))=x:X \quad (X:\mathrm{set},\ Y:(X)\mathrm{prop},\ x:X,\ y:Y(x));$$
$$q(X,Y,\mathrm{pair}(X,Y,x,y))=y:Y(x) \quad (X:\mathrm{set},\ Y:(X)\mathrm{prop},\ x:X,\ y:Y(x)).$$

Π is of same type as Σ. The monomorphic λ-abstraction and ap-operator are introduced and explained in an obvious way.

Π:(X:set)(X)prop)prop;

λ:(X:set)(Y:(X)prop)((x:X)Y)Π(X,Y);

ap:(X:set)(Y:(X)prop)(Π(X,Y))(x:X)Y(x);

ap(X,Y,λ(X,Y,y),x)=y(x):Y(x)
(X:set, Y:(X)prop, y:(x:X)B, x:X).

For any operator $c:(x_1:\alpha_1)\ldots(x_n:\alpha_n)\alpha$, we can by successive applications derive the rule

$$\frac{a_1:\alpha_1 \quad \ldots \quad a_n:\alpha_n(a_1,\ldots,a_{n-1}/x_1,\ldots,x_{n-1})}{c(a_1,\ldots,a_n):\alpha(a_1,\ldots,a_n/x_1,\ldots,x_n)}\,c.$$

In this way it is possible to derive from category assignments monomorphic lower level rules, corresponding to the polymorphic ones by means of which type theory is presented in Martin-Löf 1982, 1984; for instance, the rules of Σ-formation and Σ-introduction:

$$\frac{A{:}\mathrm{set} \quad B{:}(A)\mathrm{prop}}{\Sigma(A,B){:}\mathrm{set}} \qquad \frac{A{:}\mathrm{set} \quad B{:}(A)\mathrm{prop} \quad a{:}A \quad b{:}B(a)}{\mathrm{pair}(A,B,a,b){:}\Sigma(A,B)}$$

5. An intuitionistic grammar for English

5.1. The structure of the grammar. The generative grammar we are going to present consists of a number of components. First, we have the lexicon consisting of category assignments to basic expressions; second, the categorial grammar consisting of the rules of the higher level intutionistic type theory, by means of which expressions of the lower-level type-theoretical formalism are built; and third, a set of sugaring rules, by means of which expressions of the formalism are turned into strings of words recognizable as English expressions. The sugaring rules are intended to be meaning-preserving, in the sense that any English expression obtained by sugaring from a formal expression has the same meaning as that formal expression.

The sugaring rules are by no means one-to-one. In general, a formal expression can be sugared into many alternative English expressions; and some English expressions can be obtained by sugaring from more than one formal expression. In the former case, we have synonymous English expressions; in the latter case, we have ambiguous English expressions. Notice that synonymy is not an equivalence relation among English sentences, since there may arise instances of the following pattern:

F > E and F > E', but not F > E";
F' > E' and F' > E", but not F' > E.

Then E and E' are synonymous, and so are E' and E", but E and E" are not. To instantiate the pattern, choose

F = $(\Sigma x{:}\mathrm{woman})(\Pi y{:}\mathrm{man})\mathrm{love}(x,y)$;
F'= $(\Pi y{:}\mathrm{man})(\Sigma x{:}\mathrm{woman})\mathrm{love}(x,y)$;
E = There is a woman who loves every man.
E'= A woman loves every man.
E"= If there is a man there is a woman who loves him.

Schematically, our grammar has the following structure.

$$\begin{tikzcd}\text{lexicon} \arrow[r, "\text{categorial grammar}"] & \text{formalism} \arrow[r, "\text{sugaring}"] & \text{English}\end{tikzcd}$$

This can be compared to the structure of Montague's PTQ grammar discussed in subsection 2.4. Montague's S-rules (Montague 1974, pp. 251-252) play the double role of (i) combining basic expressions into analysis trees and (ii) sugaring analysis trees into plain English.

$$\begin{tikzcd}
\text{basic expressions} \arrow[r, "\text{S-rules (i)}"] & \text{analysis trees} \arrow[r, "\text{S-rules(ii)}"] \arrow[d, "\text{translation}"] & \text{English} \\
& \text{intensional logic} &
\end{tikzcd}$$

The difference is just that we only have one representation formalism, which serves as the basis of both sugaring and meaning explanations.

If a logical representation formalism, rather than a purely syntactic formalism, is used as the basis of generation of English sentences, the generation can make effective use of semantic conditions of well-formedness. An alternative approach would be to add such conditions in the purely syntactic formalism. But we shall start with full semantic representations, and implement purely syntactic information in the sugaring rules.

5.2. <u>Three methodological principles</u>. The division of labor between the components of our grammar is regulated by three principles, which can be conceived as requirements of parallelism between meaning and form, i.e. of compositionality.

The first principle requires that as many expressions of English as possible be assigned direct type-theoretical meanings, i.e. that they be introduced in the lexicon:

Categorize what you can.

But as categorizations have their ontological readings as well, we must be able to make full type-theoretical sense of every expression we categorize. Hence the second principle,

Do not categorize what you cannot.

This principle rules out the categorization of English quantifier words "any", "every", "some", and the indefinite article, because these words cannot always be used for expressing what "Π" and "Σ" express. For instance, if we introduced "every" in the lexicon, by the entries

every:(X:set)((X)prop)prop,
every=Π:(X:set)((X)prop)prop,

we could, by substituting "every man" for "x", sugar

(45) (every x:man)(return(x) $\supset$ I am glad)

into

(46) If every man returns I am glad.

But this sugaring obviously does not preserve meaning. We must use "any" instead of "every" as the sugaring of "Π" in cases like this.

More generally, as English does not use bound variables, its quantifier words do not function in the same way as quantifiers in logical formalisms. The scopes of English quantifiers are determined by rules that constrain their usability in a way formal quantifiers are not constrained (by ordering principles, see subsection 5.6). Hence they cannot be identified with formal quantifiers in the lexicon.

But there is a weaker sense in which "every", etc, have unique meanings. For each of the words, there is a unique formal expression from which all its occurrences in English sentences result. For "every" and "any" it is "Π", for the indefinite article and "some" it is Σ. These properties of the sugaring rules are expressed by _quasicategorizations_, e.g.

every < Π:(X:set)((X)prop)prop,
INDEF < Σ:(X:set)((X)prop)prop.

Our third methodological principle is, in effect,

Quasicategorize what you can.

One of the consequences of having an expression quasicategorized is that the parsing rules will be able to treat it in a uniform way, e.g. always yield "Σ" for the indefinite article. (This is with the exception of expressions of forms "there is INDEF" and "a certain", of which the indefinite article is not treated as a detachable part. The meaning of both of these phrases is existential rather than universal.)

5.3. The lexicon. We categorize a number of common nouns as expressions for sets, a number of verbs and adjectives as expressions for propositional functions over those sets, and a number of nouns as singular terms or as expressions for functions that yield individuals as values. The lexical entries also indicate the sugaring patterns N0, V1, etc. For example,

```
man:set of N0
walk:(man)prop of V1
own:(man)(donkey)prop of V2
old:(man)prop of A1
John:man of T0
father:(man)man of T1.
```

Having this lexicon of course requires that we define the sets man, etc, and the functions walk, etc, in an appropriate way. As we do not give such definitions here, our type-theoretic language remains schematic, or uninterpreted, just like Montagues's intensional logic in the PTQ paper, where the passage on interpretation only assumes the domains of entities, of possible worlds, and of moments of time to be given, without actually giving them (Montague 1974, p. 257). It is also obvious that no single interpretation can capture the whole of the usage of those words. For instance, man can hardly be defined as a set once and for all, but at most as a type.

The lexicon also contains the operators Σ, Π, pair, λ, p, q, and ap, categorized and explained in subsection 4.4.

Those English words that occur in the example sentences are assumed to be categorized in the same way as those we have explicitly introduced in this subsection.

5.4. <u>The sugaring procedure</u>. We shall define two operators, S and N, that take propositional expressions of the lower level type theory as arguments and turn them into sentences and nouns, respectively, of English. Their execution starts with the outermost forms of formal expressions, and proceeds stepwise to smaller subexpressions, which are either of Σ or Π form, or formed by means of English predicates. Sugaring terminates in a string of English words and morphological operations.

Certain steps in the sugaring procedure are impossible to execute for certain expressions. In such a case, one must step back and choose an alternative path. For instance, there will be no way to execute $N((\Pi x:A)B)$. Thus if sugaring proceeds to something of this form, we must step back and find a path where $S((\Pi x:A)B)$ appears instead. This will be always possible.

Moreover, there is a guaranteed sugaring path, which introduces connectives rather than quantifier words. For all propositional expressions can be sugared into sentences, if not otherwise then by using the rule (THERE). Some expressions are sugared into nested conditionals, which are nearly unacceptable in ordinary English.

Before the sugaring rules proper, auxiliary rules are applied to singular terms, turning some of them into <u>reflexive pronouns</u> (REFL), marking them as <u>main arguments</u> (M), and equipping each of them with a <u>spectrum</u> of alternative <u>anaphoric expressions</u> (SPECTRUM).

5.5. <u>The system of sugaring rules</u>. The present system generates a fragment smaller than the one described in Mäenpää and Ranta 1989, which also comprised negative sentences. As for the notation used in the rules, observe the distinction made between the concatenation marks _ and -, of which the latter is used for tying together expressions belonging to one and the same term. The notation [E/F] indicates the substitution of the expression E for the expression F; its scope is the preceding expression delimited by parentheses. The notation {E,F,G} indicates that E, F, and G are alternatives.

The auxiliary rules.

(REFL) C(a,a) > C(a, REFL(A)) for C:(A)(A)prop atomic

(M) For each atomic C and variable x,
$C(d(p^{*}x)) > C(d(Mx))$
$C(d(p^{*}x),b) > C(d(Mx),b)$
$C(a,d(p^{*}x)) > C(a,d(Mx))$ if a contains no Mx;
$p^{*}x$ is one of x, p(x), p(p(x)), etc, and
d(c) is c, father(c), wife(father(c)), etc. ^

The rule (M) yields, for example,

walk(x) > walk(Mx);
admire(father(p(x)),p(x)) > admire(father(Mx),p(x));
beat(p(q(x)),p(x)) > beat(p(q(x)),Mx).

(SPECTRUM) C(a) > C(a{PRON(A),the-N(A)}) for C:(A)α

The operations S and N.

(THERE) S(A) > there_is_INDEF-N(A).

(Q) $S((\Sigma x:A)B)$ >
S(B[{INDEF-N(A),some-N(A),a-certain-N(A)}/Mx])
$S((\Pi x:A)B)$ >
S(B[{every-N(A),any-N(A),each-N(A)}/Mx])
provided there is exactly one Mx in B.

(C) $S((\Sigma x:A)B)$ > (S(A))_and_(S(B))
$S((\Pi x:A)B)$ > if_(S(A))_(S(B))

(R) $N((\Sigma x:A)B)$ > (N(A))-REL(x:A)-(S(B))

```
(N0) N(C) > C
(V1) S(C(a)) > a_VF(C)
(V2) S(C(a,b)) > a_VF(C)_ACC(b)
(A1) S(C(a)) > a_VF(be)_C
(T0) c > c
(T1) c(a) > GEN(a,c)
```

The morphological operations. Sugaring rules introduce morphological operators VF ("verb form", in the present fragment always the third person singular present indicative), ACC ("accusative", relevant for pronouns), GEN ("genitive", with "'s" or "of", idiomatic for pronouns), REFL(A) (the reflexive pronoun of type A), INDEF-A ("A" preceded by "a" or "an"), PRON(A) (the personal pronoun of type A, either, "he", "she", or "it"), and REL(x:A) (the relative pronoun). Their execution is straightforward, with the exception of REL(x:A). It must be executed for expressions of the form

(A)-REL(x:A)-(B).

In the following four cases we must sugar

REL(x:A) > such-that

while leaving B unaltered:

(i) if B is if_(C)_(D);
(ii) if B is (C)_and_(D);
(iii) if B is (there_is_C);
(iv) if B contains no Mx or more than one.

Otherwise, we can attach to REL(x:A) the term in B that contains Mx, and substitute the empty morph for its occurrence in B. As is clear from the definition of Mx, we then have to sugar an expression of one of the forms

REL(x:A)Mx,
REL(x:A)ACC(Mx),
REL(x:A)GEN(Mx,d),

REL(x:A)ACC(GEN(Mx,d)),
etc, iterating the genitive.

These can be sugared in obvious ways, e.g.

REL(x:man)Mx > who,
REL(x:man)ACC(Mx) > whom,
REL(x:man)GEN(GEN(Mx,father),wife) > whose father's wife.

5.6. The sugaring of Σ and Π. For both Σ and Π, English has two alternative modes of expression, by a quantifier and by a connective. The connective is, in principle, always possible, but the quantifier is heavily regulated. For the first, it demands that there be a noun-like expression for the domain of quantification; for the second, there must be exactly one main argument for which the quantifier phrase can be substituted. As the choice between a quantifier and a connective must often be made in the process of sugaring, the distinction has not been categorized.

To sugar $(\Sigma x:A)B$ into a sentence, one of the quantifier phrases "INDEF-N(A)", "some-N(A)", "a-certain-N(A)" is substituted for Mx in B. If there is no Mx in B, or several ones, $(\Sigma x:A)B$ must be sugared into the conjunction (S(A))_and_(S(B)) instead. For $(\Pi x:A)B$, we have analogously the quantifier phrases "every-N(A)", "any-N(A)", "each-N(A)", and the conditional if_(S(A))_(S(B)).

The choices between the alternative quantifier words are not completely free. Having neither bound variables nor parentheses, English indicates the scopes of quantifiers by reference to an order between different quantifier words. The words themselves are unambiguously either existential or universal, but they cannot always be used for expressing the meanings they have. See (41), (42), (45), and (46). For one more example, if we sugar

(47) $(\Sigma x:\mathrm{man})(\Pi y:\mathrm{walk}(x))\mathrm{talk}(x)$

by applying (Q) to Σ and (C) to Π, we cannot choose the indefinite article for Σ, for then we would obtain

(48) If a man walks he talks.

The proper formalization of (48) is rather

(49) $(\Pi y:(\Sigma x:\mathrm{man})\mathrm{walk}(x))\mathrm{talk}(p(y))$,

where Π has wider scope than Σ. To indicate the scope of Σ as wider than the scope of Π, we can choose "a certain" instead of the indefinite article:

(50) If a certain man walks he talks.

This way to look upon English quantifier words has been developed in game-theoretical semantics (see e.g. Hintikka and Kulas 1985, pp. 15-21). To each quantifier or connective word corresponds a game rule, which assigns to the word an unambiguous existential or universal meaning. The order in which game rules are applied to English input sentences is not completely free, but regulated by <u>ordering principles</u>, such as

Apply the rule for "if" before the rule for the indefinite article.

This approach can be contrasted to the treatment of English quantifier words as ambiguous, e.g. of the indefinite article as universal in antecedents of conditionals, existential elsewhere. As we have noticed in subsection 2.5, such a treatment is necessary if ordinary predicate calculus is used as the representation formalism. The treatment suggested by game-theoretical semantics has the methodological virtue of making quasicategorizations possible.

When implemented in a generative grammar, the ordering principles function as principles regulating the choices of alternative quantifier words. There are alternative ways of implementing them in the sugaring procedure; but for the purpose of exposition, it is clearest to regard them as metarules of the sugaring rules proper. We shall assume the following principles:

> "if" has wider scope than the indefinite article in the antecedent;
> "if" has wider scope than "every" in the antecedent;
> "a certain" has wider scope than "if";
> "any" has wider scope than "if";
> any expression in a relative clause has its scope inside the relative clause.

Some of the ordering principles suggested by game-theoretical semantics are not taken along, e.g. the left-to-right principle according to which the indefinite article has wider scope than "every" occurring to the right of it. That principle would block the ambiguity of the sentence (1), "A woman loves every man". Nor have we assumed the full any-thesis, which gives "every" and "any" complementary distribution. It would rule out Geach's original donkey sentence.

5.7. <u>Anaphoric expressions</u>. After the execution of the sugaring operations and morphological operation REL(x:A), there remain singular terms endowed with spectra of anaphoric expressions. The spectra of all terms are compared, and if the pronouns PRON(A) and PRON(B) are the same for some unequal a[A] and b[B], the-N(A) and the-N(B) must be chosen instead of PRON(A) and PRON(B). If "a" is a constant term, its leftmost occurrence must remain in the form "a".

This procedure captures something of the principle that

> the reference of each anaphoric expression must be uniquely determined in the context where the expression occurs.

To capture it better, the procedure must be elaborated. Such an elaboration has been attempted in Ranta 1990.

Pronouns and the-phrases have similar

quasicategorizations, as identity mappings of any set an element of which has been given in context:

PRON < (X)(x)x:(X:set)(X)X;
the < (X)(x)x:(X:set)(X)X.

The-phrases are acceptable more often than pronouns, as they are more specific. To guarantee uniqueness of reference in all cases, even more specific anaphoric expressions are needed; cf. Ranta 1990, sections 5 to 7.

5.8. An example of proposition formation and sugaring. The proposition that can be sugared into Geach's donkey sentence is derived by the rules of the higher-level type theory as follows. Discharges of hypotheses are indicated by numerals written both beside the infence lines at which discharges are made and above those hypotheses that are discharged.

$$\dfrac{\dfrac{\text{man:set} \quad \dfrac{\dfrac{\text{donkey:set} \quad \dfrac{\dfrac{\overset{2.}{\text{x:man}} \quad \overset{1.}{\text{y:donkey}}}{\text{own(x,y):prop}}\,\text{own}}{\text{(y)own(x,y):(donkey)prop}}\,\text{abstr. 1.}}{\Sigma\text{(donkey,(y)own(x,y)):prop}}\,\Sigma}{\text{(x)}\Sigma\text{(donkey,(y)own(x,y)):(man)prop}}\,\text{abstr. 2.}}{(\Sigma\text{x:man})(\Sigma\text{y:donkey})\text{own(x,y):prop}}\,\Sigma \qquad \dfrac{\dfrac{\dfrac{\overset{3.}{\text{z:}(\Sigma\text{x:man})} \atop (\Sigma\text{y:donkey})\text{own(x,y)}}{\text{p(z):man}}\,\text{p} \quad \dfrac{\dfrac{\overset{3.}{\text{z:}(\Sigma\text{x:man})} \atop (\Sigma\text{y:donkey})\text{own(x,y)}}{\text{q(z):} \atop (\Sigma\text{y:donkey})\text{own(x,y)}}\,\text{q}}{\text{p(q(z)):donkey}}\,\text{p}}{\text{beat(p(z),p(q(z))):prop}}\,\text{beat}}{\text{(z)beat(p(z),p(q(z))):} \atop ((\Sigma\text{x:man})(\Sigma\text{y:donkey})\text{own(x,y))prop}}\,\text{abstr. 3.}}{(\Pi\text{z:}(\Sigma\text{x:man})(\Sigma\text{y:donkey})\text{own(x,y))beat(p(z),p(q(z))):prop.}}\,\Pi$$

This can be sugared in the following ways. Choosing alternative quantifier words and anaphoric expressions multiplies the number of sugarings.

If there is a man and there is a donkey and he owns it he beats it. By (C), (C), (C).
If there is a man and he owns a donkey he beats it. By (C), (C), (Q).
If there is a man and there is a donkey that he owns he beats it. By (C), (C), (R).

If there is a donkey and a man owns it he beats it. By (C), (Q), (C).

If a man owns a donkey he beats it. By (C), (Q), (Q).

If there is a man such that there is a donkey and he owns it he beats it. By (C), (R), (C).

(By (C), (Q), (R) we do not get anything.)

If there is a man who owns a donkey he beats it. By (C), (R), (Q).

If there is a man such that there is a donkey that he owns he beats it. By (C), (R), (R).

Every man such that there is a donkey and he owns it beats it. By (Q), (R), (C).

(Starting by (Q), (C) or by (Q), (Q) does not give result.)

Every man who owns a donkey beats it. By (Q), (R), (Q).

Every man such that there is a donkey that he owns beats it. By (Q), (R), (R).

5.9. The dynamicity of texts. The derivation of the donkey proposition illustrates the way anaphoric dependencies are generated. The proposition beat(p(z),p(q(z))) formed in the context z:(Σx:man)(Σy:woman)own(x,y) contains free occurrences of the variable z referring to an "object given in the context". In virtue of the propositions as types principle, not only anaphoric dependencies are captured in this way, but presuppositions as well - not in the technical sense of type theory, but in the sense in which we can presuppose the truth of some given propositions to form a further one. For

B:prop (A true)

means

B:prop (x:A).

To represent texts consisting of sequences of sentences, we recall that the full representation of an indicative sentence is not a proposition but a judgment of form a:A. For a given indicative sentence, the proof cannot generally be restored as a constant, but only as a variable. Texts are represented as contexts of the form

$$x_1{:}A_1,\ldots,x_n{:}A_n,$$

where each proposition A_k depends on the foregoing context. This way to look upon texts is developed in further detail in Ranta (forthcoming).

5.10. Extension of the fragment to complex predicates. The rule (Q) for sugaring $(\Sigma x{:}A)B$ and $(\Pi x{:}A)B$ assumes there to be in B exactly one main argument containing Mx. Hence we cannot sugar e.g.

(51) $(\Pi x{:}\mathrm{man})(\Sigma y{:}\mathrm{walk}(x))\mathrm{talk}(x)$
> Every man walks and talks.

But we can retain the requirement of a unique main argument and need not modify the rule (Q) so as to allow for sugarings like this. For we can reduce the number of argument places into one, by defining a complex predicate:

$$(\mathrm{walk\_and\_talk})=(x)(\Sigma y{:}\mathrm{walk}(x))\mathrm{talk}(x):(\mathrm{man})\mathrm{prop}.$$

More generally, we can define variants of Σ and Π for the formation of predicates from predicates on any domain, e.g.

$$\Sigma_A=(Y)(Z)(x)\Sigma(Y(x),Z(x)):$$
$$(Y{:}(A)\mathrm{prop})((x{:}A)(Y(x))\mathrm{prop})(A)\mathrm{prop}.$$

The higher-level rules of abstraction and application thus produce an effect resembling the type change calculus of Lambek (1958) and van Benthem (1988). van Benthem is unwilling to admit the whole power of the abstraction rule, because it e.g. makes vacuous binding possible (op. cit., p. 45.) We have assumed the rule of abstraction simply because it is semantically acceptable. Vacuous binding means that x does not occur free in B in $(\Sigma x{:}A)B$ and $(\Pi x{:}A)B$. We have characterized the English words "and", "if" and "such that" as allowing for vacuous binding, in contrast to "some", "every", and "which". More accurately, words of the former group can arise in the sugaring of expressions with no main argument, whereas words of the latter group cannot.

References

R. Ahn and H-P Kolb, "Discourse representation meets constructive mathematics", to appear in the proceedings of the Second Symposium on Logic and Language held at Hajdúszoboszló in September 1989.

J. van Benthem, "The semantics of variety in categorial grammar", in W. Buszkowski, W. Marciszewski, and J. van Benthem (eds), Categorial Grammar, John Benjamins Publishing Co., Amsterdam and Philadelphia, 1988.

N.G. de Bruijn, "The mathematical language AUTOMATH, its usage and some of its extensions", in Lecture Notes in Mathematics 125, 1970, pp. 29-61.

Th. Coquand and G. Huet, "The calculus of constructions", in Information and Computation 76, 1988, pp. 95-120.

H. B. Curry and R. Feys, Combinatory Logic, Vol. 1, North-Holland, Amsterdam, 1958.

G. Frege, Begriffsschrift, Verlag von Louis Nebert, Halle, 1879.

P. Geach, Reference and Generality, Cornell University Press, Ithaca, New York, 1962.

P. Geach, "A program for syntax", in D. Davidson and G. Harman (eds), Semantics of Natural Language, D. Reidel, Dordrecht, 1972, pp. 483-497.

J. Groenendijk and M. Stokhof, "Dynamic predicate calculus", ITLI Prepublication series LP-89-02, to appear in Linguistics and Philosophy.

J. Groenendijk and M. Stokhof, "Dynamic Montague grammar", ITLI Prepublication Series LP-90-02.

R. Harper, F. Honsell, and G. Plotkin, "A framework for defining logics", Proceedings of the Symposium on Logic in Computer Science, Ithaca, New York, 1987, pp. 194-204.

A. Heyting, Intuitionism, North-Holland, Amsterdam, 1956.

J. Hintikka and L. Carlson, "Conditionals, generic quantifiers, and other applications of subgames", E. Saarinen (ed), Game-Theoretical Semantics, D. Reidel, Dordrecht, 1979, pp. 179-214.

J. Hintikka and J. Kulas, Anaphora and Definite Descriptions, D. Reidel, Dordrecht, 1985.

W. Howard, "The formulae-as-types notion of construction", J.P. Seldin and J.R. Hindley (eds), To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, Academic Press, London, 1980, pp. 479-490.

H. Kamp, "A theory of truth and semantic representation", in J.A.G. Groenendijk, T.M.V. Janssen, and M.B.J. Stokhof (eds), Formal Methods in the Study of Language, Part 1, Mathematical Centre Tracts 135, Mathematisch Centrum, Amsterdam, 1981, pp. 277-322.

J. Lambek, "The mathematics of sentence structure", in

American Mathematical Monthly 65, 1958, pp. 154-170.

P. Martin-Löf, "An intuitionistic theory of types: predicative part", H.E.Rose and J.C. Shepherdson (eds), in Logic Colloquium '73, North-Holland, Amsterdam, 1975, pp. 73-118.

P. Martin-Löf, "Constructive mathematics and computer programming", in L.J. Cohen, J. Los, H. Pfeiffer, and K-P Podewski (eds), Logic, Methodology, and Philosophy of Science VI, North-Holland, Amsterdam, 1982, pp. 153-175.

P. Martin-Löf, Intuitionistic Type Theory, Bibliopolis, Napoli, 1984.

R. Montague, Formal Philosophy, Collected papers edited by R. Thomason, Yale University Press, New Haven, 1974.

P. Mäenpää and A. Ranta, "An implementation of intuitionistic categorial grammar", to appear in the proceedings of the Second Symposium on Logic and Language held at Hajdúszoboszló in September 1989.

B. Nordström, K. Petersson, and J. Smith, Programming in Martin-Löf's Type Theory. An Introduction, Oxford University Press, Oxford, 1990.

A. Ranta, "Propositions as games as types", in Synthese 76, 1988, pp. 377-395.

A. Ranta, "Anaphora in game-theoretical semantics and in intuitionistic type theory", to appear in L. Haaparanta, M. Kusch, and I. Niiniluoto (eds), Language, Knowledge, and Intentionality - Perspectives in the Philosophy of Jaakko Hintikka, Acta Philosophica Fennica, North-Holland, Amsterdam, 1990.

A. Ranta, "Meaning in text", mimeographed, University of Stockholm; forthcoming.

G. Sundholm, "Proof theory and meaning", in D. Gabbay and F. Guenthner (eds), Handbook of Philosophical Logic, Vol. 3, D. Reidel, Dordrecht, 1986, pp. 471-516.

G. Sundholm, "Constructive generalized quantifiers", in Synthese 79, pp. 1-12.

The ITLI Prepublication Series

1990

Logic, Semantics and Philosophy of Language	
LP-90-01 Jaap van der Does	A Generalized Quantifier Logic for Naked Infinitives
LP-90-02 Jeroen Groenendijk, Martin Stokhof	Dynamic Montague Grammar
LP-90-03 Renate Bartsch	Concept Formation and Concept Composition
LP-90-04 Aarne Ranta	Intuitionistic Categorial Grammar
Mathematical Logic and Foundations	
ML-90-01 Harold Schellinx	Isomorphisms and Non-Isomorphisms of Graph Models
ML-90-02 Jaap van Oosten	A Semantical Proof of De Jongh's Theorem
ML-90-03 Yde Venema	Relational Games
Computation and Complexity Theory	
CT-90-01 John Tromp, Peter van Emde Boas	Associative Storage Modification Machines
CT-90-02 Sieger van Denneheuvel Gerard R. Renardel de Lavalette	A Normal Form for PCSJ Expressions
Other Prepublications	
X-90-01 A.S. Troelstra	Remarks on Intuitionism and the Philosophy of Mathematics, Revised Version
X-90-02 Maarten de Rijke	Some Chapters on Interpretability Logic
X-90-03 L.D. Beklemishev	On the Complexity of Arithmetical Interpretations of Modal Formulae
X-90-04	Annual Report 1989
X-90-05 Valentin Shehtman	Derived Sets in Euclidean Spaces and Modal Logic