Notre Dame Journal of Formal Logic
Volume 30, Number 3, Summer 1989

Logical Constants Across Varying Types

JOHAN van BENTHEM

Abstract We investigate the notion of "logicality" for arbitrary categories of linguistic expression, viewed as a phenomenon which they can all possess to a greater or lesser degree. Various semantic aspects of logicality are analyzed in technical detail: in particular, invariance for permutations of individual objects, and respect for Boolean structure. Moreover, we show how such properties are systematically related across different categories, using the apparatus of the typed lambda calculus.

1 The range of logicality Philosophical discussions of the nature of logical constants often concentrate on the connectives and quantifiers of standard predicate logic, trying to find out what makes them so special. In this paper, we take logicality in a much broader sense, including special predicates among individuals such as *identity* ("be") or higher operations on predicates such as *reflexivization* ("self").

One convenient setting for achieving the desired generality is that of a standard *Type Theory*, having primitive types e for entities and t for truth values, while forming functional compounds (a,b) out of already available types a and b. Thus, e.g., a one-place predicate of individuals has type (e,t) (assigning truth values to individual entities), whereas a two-place predicate has type $(e,(e,t))$. Higher types occur, among others, with quantifiers, when regarded in the Fregean style as denoting properties of properties: $((e,t),t)$. For later reference, here are some types, with categories of expression taking a corresponding denotation:

e	entities	proper names
t	truth values	sentences
(t,t)	unary connectives	sentence operators

Received January 8, 1988

$(t,(t,t))$	binary connectives	sentence connectors
(e,t)	unary individual predicates	intransitive verbs
$(e,(e,t))$	binary individual predicates	transitive verbs
$((e,t),t)$	properties of predicates	noun phrases ("a man")
$((e,t),((e,t),t))$	relations between predicates	determiners ("every")
$((e,t),(e,t))$	unary predicate operators	adjectives/adverbs
$((e,(e,t)),(e,t))$	argument reducers	"self", "passive"

Many of these types (and their corresponding categories of expression) can contain logical items. For instance, identity lives in $(e,(e,t))$, reflexivization in $((e,(e,t)),(e,t))$. Moreover, the type $((e,t),(e,t))$ contains such logical items as the operation of *complement*, or identity, whereas $((e,t),((e,t),(e,t)))$ would add such operators as *intersection* or *union*. Conversely, existing 'logical' items may be fitted into this scheme. For instance, in his well-known set-up of predicate logic without variables, Quine introduced the following operators in addition to the usual connectives and quantifiers: reflexivization (as mentioned above) as well as various forms of permutation on predicates. One notable instance is the operation of *conversion* on binary predicates: a logical item (as we shall see) in type $((e,(e,t)),(e,(e,t)))$.

The purpose of this article is to analyze a general notion of logicality, encompassing all these examples, by drawing upon some insights from contemporary logical semantics of natural language (see [7]). In the process, a great number of questions concerning various aspects of logicality will be raised in a systematic manner.

In fact, the above sketch may already have suggested some of these general questions to the reader. First and foremost, What *is* a suitable notion of logicality, applicable to all types at once? But then also, Will logical items be found in all types, or if not, in *which ones*? And finally, What are the *connections* between logical items across different types? For instance, intersection of unary predicates, in the type $((e,t),((e,t),(e,t)))$, seems a very close relative of the more elementary conjunction of sentences, living in the type $(t,(t,t))$. We shall examine all these questions, as well as many others, in due course.

When analyzing the notion of 'logicality', it seems natural to start from the standard connectives and quantifiers. And in fact we shall find various aspects of their behavior which may be called 'logical', and which can be generalized so as to apply to items in other types as well. Most of these aspects fall within the *semantic* approach, characterizing logical constants by their *invariance* behavior across different semantic structures. Roughly speaking, logical constants will be those items whose denotation is invariant under quite drastic changes of semantic models. There is also another broad tradition, however, localizing logicality rather in some key role to be played in *inference*. On this view, logical constants would be those items that support rich and natural sets of inferential patterns. The latter perspective can be pursued entirely within *syntactic* proof theory. Nevertheless, we shall find some semantic echoes of it too, especially when we consider the interplay of logicality with general (Boolean) implication.

As may be clear from the preceding considerations, our aim is not so much to characterize such and such a set as consisting of precisely 'the' logical con-

stants, but we are more interested in the multi-faceted *phenomenon* of 'logicality', which may occur, to a greater or lesser degree, with many expressions in natural language. One telling example here is the type $((e,t),((e,t),t))$ of *generalized quantifiers*, as realized in determiner expressions such as "every", "some", "no", etc. Insisting on merely the standard first-order examples here would make us blind to the many 'logical' traits of such *nonstandard* quantifiers as "most" or "many". In fact, the standard examples have so many nice properties together, of quite diverse sorts, that it may not even be reasonable to think of their conjunction as defining one single 'natural kind'.

2 General invariance The traditional idea that logical constants are not concerned with real content may be interpreted as saying that they should be *preserved* under those operations on models that change content, while leaving general structure intact. This is actually not one intuition, but a family of them, as there are various ways of implementing such ideas. We shall consider several here, starting with perhaps the most natural one.

Related to the preceding aspect of logicality is the feeling that logical denotations should be *uniform*, in the sense that they should not have one type of behavior for certain arguments and a vastly different one for arguments only slightly different. Perhaps, in this day and age, it is even natural to localize this uniformity in the existence of some *computational procedure* recognizing the relevant logical connection. Such further traits will also be encountered in what follows.

2.1 Individual neutrality Logical constants should not be sensitive to the particular individuals present in a base domain D_e. This idea can be made precise using *permutations*, or more generally *bijections* defined on that domain, which shuffle individuals. For instance, the logicality of the quantifier "all" (i.e., inclusion among unary predicates, or sets of individuals) may be expressed as follows. For all $X, Y \subseteq D_e$

$$\textit{all}\ XY \text{ if and only if } \textit{all}\ \pi[X]\pi[Y], \text{ for all permutations } \pi \text{ of } D_e.$$

Likewise, a Boolean operation like complement on sets will show a similar 'commutation with permutations'. For all $X \subseteq D_e$

$$\pi[\textit{not}(X)] = \textit{not}\ \pi[X].$$

And finally, of course, the relation of identity will be unaffected by such permutations. For all $x, y \in D_e$

$$x = y \text{ if and only if } \pi(x) = \pi(y).$$

The common generalization of all these cases is as follows. Let the hierarchy of type domains D_a be given by the obvious induction:

D_e is the base set of individual entities
D_t is the set of truth values $\{0,1\}$
$D_{(a,b)}$ is the set of all functions with domain D_a and range D_b.

Now, any permutation π on D_e can be lifted to a family of permutations defined on all type domains in a canonical manner:

$\pi_e = \pi$
π_t is the identity map (truth values retain their individuality)
$\pi_{(a,b)}(f) = \{(\pi_a(x), \pi_b(y)) | (x,y) \in f\}$, for $f \in D_{(a,b)}$.

Definition An item $f \in D_a$ is *permutation-invariant* if

$$\pi_a(f) = f \text{ for all permutations } \pi \text{ of } D_e.$$

It is easy to see that this notion subsumes the above three examples.

Using this notion, we can now search types systematically for their invariant items. E.g., e itself will not contain any (if there is more than one object), whereas t—and indeed all types constructed from t only—will have all its inhabitants trivially invariant. (This means that all higher operations on truth values are also considered to be logically 'special'.) In fact, our main interest will lie in the 'mixed' types involving both e and t. Here are some important examples (cf. Chapter 3 of [7]):

- $(e,(e,t))$: the only invariant items are *identity*, *nonidentity*, and the *universal* and *empty* relations.
- $((e,t),t)$: the invariant items can all be described as accepting arguments with certain sizes and rejecting all others ('numerical quantifiers'). For instance, "everything" accepts only sets of the cardinality of D_e (i.e., only D_e itself), "something" accepts only sets of size at least one. But also, less smooth candidates qualify, such as

  $\#(X)$ iff X has between 2 and 7, or exactly 10, elements.

- $((e,(e,t)),(e,t))$: here too there are many invariant items. We merely show that the earlier reflexivization is among them:

  $$\text{SELF}(R) = \{x \in D_e | (x,x) \in R\}.$$

  The calculation is this:

  since $R = \pi(\pi^{-1}(R))$, by definition,
  $\pi(\text{SELF})(R) = \pi(\text{SELF}(\pi^{-1}(R)) = \pi(\{x \in D_e | (x,x) \in \pi^{-1}(R)\}) =$
  $\pi(\{x \in D_e | (\pi(x), \pi(x)) \in R\}) = \{x \in D_e | (x,x) \in R\} = \text{SELF}(R)$.

Finally, we consider one case where permutation invariance by itself already comes close to singling out the usual logical constants (cf. Chapter 3 of [7]).

Proposition *Among the n-ary operations on sets, the only permutation-invariant ones are those defined at each tuple of arguments by some Boolean combination.*

Proof: We sketch the argument, in order to show how the present kind of invariance enforces a certain *uniformity* of behavior (see Figure 1). Consider binary operations, for the sake of convenience. Let f be permutation-invariant. What could $f(X, Y)$ be? The relevant Venn diagram has four natural 'zones', and we can see that $f(X, Y)$ must either contain or avoid these in their entirety. E.g., if $f(X, Y)$ were to contain $u \in X - Y$, while missing $v \in X - Y$, then we could define a permutation π of individuals interchanging only u and v, while leaving everything else undisturbed. But then $\pi(X) = X$, $\pi(Y) = Y$, and yet

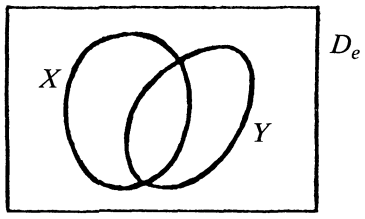


Figure 1.

$\pi(f(X, Y)) \neq f(\pi(X), \pi(Y))$. So, f must choose some union of regions: which is described by a Boolean combination of its arguments.

Thus, we see that permutation invariance already captures a lot of 'logicality'.

Moreover, the notion just defined raises several questions of its own. To begin with, Which categories possess invariant items at all? Here, it is useful to distinguish types into two main varieties. Every type can be written in one of the forms

$(a_1,(a_2,\ldots,(a_n,t)\ldots))$ or
$(a_1,(a_2,\ldots,(a_n,e)\ldots))$.

The first, with 'final t', will be called *Boolean types*, since they carry a natural Boolean structure (cf. [17]). The second may be called *individual types.*

Proposition *All types contain permutation-invariant items, except those individual types all of whose arguments a_i contain permutation-invariant items.*

Proof: By induction on types. Note first that every Boolean type has at least one invariant item, being the function giving the constant value 1 throughout (for all tuples of arguments). Moreover, e has no invariants (provided that there is more that one individual object). Now, consider any complex individual type of the above form.

Case 1: All a_i have invariant items, say $x_1,\ldots,x_n$. Let f be any item in the type being considered. Its value

$$f(x_1)(x_2)\ldots(x_n)$$

will be some individual object in D_e. Let π be any permutation shifting that object. Then $\pi(f)(x_1)(x_2)\ldots(x_n) = \pi(f)(\pi(x_1))(\pi(x_2))\ldots(\pi(x_n))$ (by the invariance of the x_i) $= \pi(f(x_1)(x_2)\ldots(x_n))$ (by the definition of $\pi(f)$) $\neq f(x_1)(x_2)\ldots(x_n)$ (by the choice of π). Thus, $\pi(f) \neq f$. I.e., our type has no invariant items f.

Case 2: At least one a_i has no invariant item, say a_1. By the inductive hypothesis, a_1 itself must be of the form

$$(a_{11},(a_{12},\ldots,(a_{1k},e)\ldots)),$$

where all a_{1i} have invariant items. (Or, a_1 may be the type e itself.) Now, we define an invariant item in our original type as follows:

- if a_1 was e, then take the function f defined by $f(x_1)(x_2)\ldots(x_n) = x_1$
- otherwise, if the a_{1i} have invariant items $y_1,\ldots,y_k$, take the function defined by $f(x_1)(x_2)\ldots(x_n) = x_1(y_1)\ldots(y_k)$.

That both these functions are permutation-invariant may be seen by direct calculation, or by an appeal to the more general results of Section 4 below.

As an application, we note that, e.g., the category of 'choice functions' from sets to objects, of type $((e,t),e)$, has no logical items.

Digression It may be useful, here and elsewhere, to also admit product types $a\cdot b$, whose meaning is fixed by the stipulation that

$$D_{a\cdot b} = D_a \times D_b.$$

For instance, we can then move back and forth, as convenience dictates, between such types as $(a_1,(a_2,b))$ and $(a_1\cdot a_2,b)$. Moreover, we obtain new cases of interest, such as $(e,e\cdot e)$. The earlier permutations, and the corresponding notion of invariance, are easily extended to this case. For instance, it may be checked that the type $(e,e\cdot e)$ has just one invariant item, being the 'duplicator'

$$x \mapsto (x,x).$$

We shall make use of this particular observation below.

As a refinement of the above result, one might ask for a general *counting formula* telling us, for each type a, how many invariant items it contains. For several types, the answer is known (cf. [7]; in general, the number will depend on the size of the individual base domain). No general result is available, however, for enumerating the phenomenon in all types.

Instead, we turn to some further issues concerning permutation invariance.

One is the location of the borderline with *noninvariant* items. Many expressions are not strictly invariant, but they still only refer to some specific structure on the universe, in the sense that they will be invariant for all *automorphisms*, i.e., all permutations of individuals which also respect this additional structure. For instance, as opposed to "all", the expression "all blonde" is not permutation-invariant. (Suppose there are as many sailors as soldiers, so that we can permute individuals by some π which maps sailors onto soldiers. However, "all blonde sailors are brave" might be true, whereas "all blonde soldiers are π(brave)" might be false: the blonde soldiers could be located anywhere.) Nevertheless, this complex determiner *is* invariant for permutations which have the additional property of respecting blondeness (only) as may easily be seen. Thus, in a sense, permutation invariance is only one extreme in a spectrum of invariances, involving various kinds of automorphisms on the individual domain. In particular, there are certain forms of automorphism invariance that still resemble logicality quite closely (see Section 5 below).

On the other hand, one could also consider *stronger* notions of invariance than the one with respect to arbitrary permutations. Permutations at least respect distinctness among individuals: they are, so to speak, 'identity automorphisms'. What about demanding invariance for arbitrary *functions* on individuals, whether one-to-one or not? We have not adopted this constraint, because many standard logical constants would not survive this test; e.g., "no X are Y" does not imply that "no $F[X]$ are $F[Y]$" for arbitrary functions F on D_e.

There is another line of possible strengthening, however. In higher types, there are many further kinds of permutation that could be considered, with their corresponding notions of invariance. An example is found in [11] which discusses *dyadic quantification* in natural language, i.e., quantifier complexes operating directly on binary relations, rather than on unary properties. The simplest relevant type here is $((e,(e,t)),t)$, as opposed to the original Fregean type $((e,t),t)$. Examples of dyadic quantification are provided by iterated unary cases such as "every boy *R* some girl", but also by more genuinely polyadic constructions such as "every boy *R* some *different* girl". Now, these various 'degrees' of polyadicity can be measured by accompanying forms of invariance for smaller or larger classes of *permutations on ordered pairs* of individuals. Of course, the earlier individual permutations induce permutations on pairs of individuals. But there are also many permutations of pairs which do *not* arise in this way. And yet, there certainly are dyadic quantifiers which are also invariant with respect to such larger classes of changes. For present purposes, it will suffice to mention one example. So-called 'resumptive' constructions in natural language essentially express dyadic quantification over pairs, as in

"someone hates someone": $\exists xy \cdot Hxy$
"no one hates no one": $\neg \exists xy \cdot Hxy$ (!)

These particular dyadic quantifiers define predicates of binary relations that are even invariant for *all* permutations of ordered pairs of individuals. It remains an open question as yet, however, what stronger notions of permutation invariance would be appropriate for *arbitrary* 'logical' items in the type of dyadic quantifiers.

Even so, the potential of the permutation/invariance aspect of logical constants should have been sufficiently established by now.

2.2 Context neutrality and other uniformities The invariances considered up until now take place within one type-theoretic structure, with a fixed base domain of individuals. But logical constants are also indifferent to certain changes of context, or environment. For instance, the earlier account of permutation invariance would not have changed at all if we had replaced permutations by arbitrary *bijections* between D_e and some other base set.

Yet another type of neutrality may be observed concerning generalized quantifiers. In general, a quantifier may be viewed as assigning, to each universe D_e, some binary relation among its unary predicates, such as inclusion ("all"), overlap ("some"), disjointness ("no"), etc. But in principle this relation might depend on the surrounding individual domain. For instance, there is a claimed reading for the determiner "many" where "many XY" would express that the proportion of Y's in X is higher than that of the Y's in the whole universe D_e. In that sense, "many" is indeed context-dependent. Nevertheless, for truly logical quantifiers, one usually adopts the following principle of *context neutrality*:

For all $X, Y \subseteq D_e \subseteq D_e'$,
$Q(X,Y)$-in-D_e if and only if $Q(X,Y)$-in-D_e'.

The general intuition here is that a logical constant should depend only on the 'individual content' of its arguments.

We can give a general formulation of this principle, to make it applicable to all types. For technical reasons, however, this is more easily done in a *relational* than in a functional Type Theory (see [16]). In this set-up, types are formed from one basic type e by (possibly iterated) formation of finite sequences, with the central stipulation that

$$D_{(a_1,\ldots,a_n)} = D_{a_1} \times \ldots \times D_{a_n}.$$

That is, $(a_1,\ldots,a_n)$ is the type of n-ary relations with arguments of the types indicated. There are mutual translations with our earlier functional Type Theory here. Let us merely observe that the *empty* sequence encodes the truth value type t, so that, e.g., the relational type $((e),e,(\ ))$ might correspond to the previous functional type $((e,t),(e,(t,t)))$ (being the characteristic function of the corresponding 3-place relation). Now, given two base sets $D_e \subseteq D_e'$, we can define an obvious *restriction* of objects in the hierarchy built on D_e' to those in the hierarchy on D_e:

- $x|D_e = \begin{cases} x, & \text{if } x \in D_e \\ \text{undefined}, & \text{otherwise} \end{cases}$ for individuals x in D_e
- $R|D_e = \{(r_1,\ldots,r_n) \in R | r_i|D_e = r_i,$ for each i $(1 \leq i \leq n)\}$ for relations R of type $(a_1,\ldots,a_n)$.

(In a functional hierarchy, there would be no guarantee that restrictions of functions remain total functions; but with relations this is not necessary.)

Now, we may demand that a logical constant be *context-neutral* in the sense that its denotation in one type hierarchy based on some individual domain D_e always be equal to the restriction of its denotations in type hierarchies based on larger individual domains.

This requirement does have some bite. For instance, it rules out an expression like "everything", which is crucially dependent on D_e. But of course, "every" as the binary inclusion relation between unary predicates does pass this test. Another problematic case is *negation*. The value of "not X" is context-dependent, in that it is a complement taken with respect to the current individual domain. Note, however, that the corresponding relation of type $((e),e)$, being '$y \notin X$', is in fact context-neutral.

Finally, there is a related and more general phenomenon to be observed in natural language. Context neutrality also expresses a certain *locality*: to evaluate a logical item on certain arguments, it suffices to restrict attention to the smallest universe containing all individuals occurring (hereditarily) 'in' those arguments. Now, various constructions in natural language also carry a certain restriction to specific subdomains. One well-known example is the phenomenon of *conservativity* with generalized quantifiers:

$$Q(X,Y) \text{ if and only if } Q(X,\ Y \cap X).$$

I.e., the first argument sets the scene of evaluation for the second. This phenomenon has a wider scope. C. S. Peirce already observed how predicate logic obeys the following 'Copying Rule' (provided some conditions are observed on freedom and bondage of variables):

$$\ldots\phi \wedge (\ldots\psi\ldots)\ldots \leftrightarrow$$
$$\ldots\phi \wedge (\ldots\phi \wedge \psi\ldots)\ldots.$$

Thus, there are general mechanisms operative which restrict evaluation to certain subdomains. In that sense, locality and context neutrality for logical constants are merely (extreme) instances of tendencies to be observed for all kinds of linguistic expression.

Finally, even with permutation invariance and context neutrality combined, some very curious customers will pass the test. For instance, the behavior of predicate operators might still be as erratic as this:

$$f(X,Y) = \begin{cases} X, & \text{if } |X| = 6 \\ Y, & \text{if } |Y| = 7 \text{ and } |X| \neq 6 \\ X \cap Y, & \text{otherwise.} \end{cases}$$

One might try to exclude such cases by means of stronger general postulates in the above spirit. (Compare the use of "Restriction" in [1].) Nevertheless, these attempts have not been overly successful. No generally reasonable notion of uniformity or 'smoothness' has emerged ruling out these cases. What we shall do instead is pass on to the study of some special types of conditions, which may not be reasonable as general constraints on all logical constants, but which certainly do determine some very natural classes among them.

3 Fine-structure There are various aspects of the behavior of the standard logical constants which should be brought out, if not as general desiderata on logicality, then at least as a means of creating finer divisions among the logical constants themselves. The examples selected here are largely derived from one particular area where this theme has been developed in detail, namely that of determiners and quantifiers. But as we shall see, there is always a possibility for type-theoretic generalization.

3.1 Monotonicity The standard predicate-logical quantifiers are *monotone*, in the sense of being unaffected by certain changes in their arguments. For instance, "every XY" is right-upward monotone:

every XY, $Y \subseteq Y'$ imply *every* XY'

and left-downward:

every XY, $X' \subseteq X$ imply *every* $X'Y$.

In all, there are four types of monotonicity here, as displayed in Figure 2, following the Square of Opposition. The interest of this observation may be seen, e.g. in Chapter 1 of [7], where it is proved that (modulo some further conditions) this double monotonicity uniquely *determines* the standard quantifiers.

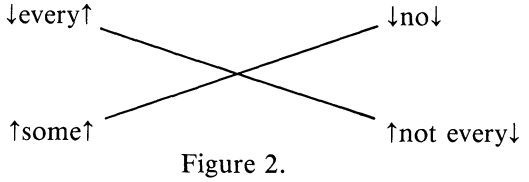


Figure 2.

But monotonicity is a much more general phenomenon in natural language, which other kinds of expression can share to some degree. Notably, a nonstandard quantifier like "most" is still upward monotone in its right-hand argument:

most XY, $Y \subseteq Y'$ imply *most* XY'.

(It lacks both kinds of monotonicity in its left-hand argument, however.) Moreover, monotonicity also makes sense in other categories. E.g., a *preposition* like "with" is monotone in the sense that, if you fall in love with an iron lady, and iron ladies are ladies, then you fall in love with a lady.

The proper general formulation of the phenomenon again involves the earlier 'Boolean' t-structure in our Type Theory. First, we define a general notion of *inclusion*, or *implication*, $\sqsubseteq$, on all types, via the following induction:

on D_t, $\sqsubseteq$ is $\leq$
on D_e, $\sqsubseteq$ is $=$
on $D_{(a,b)}$, $f \sqsubseteq g$ if, for all $x \in D_a$, $f(x) \sqsubseteq g(x)$.

In specific cases, such as the type (e,t), this meets our intuitive expectations (being set inclusion for (e,t)).

Now, a function f in type (a,b) may be called *monotone* if the following 'direct correlation' holds:

for all $x,y \in D_a$, $x \sqsubseteq y$ only if $f(x) \sqsubseteq f(y)$.

This generalizes the earlier 'upward' form of monotonicity. The 'downward' variant would correspond to an 'inverse correlation' (*antitone*):

for all $x,y \in D_a$, $x \sqsubseteq y$ only if $f(y) \sqsubseteq f(x)$.

For instance, an adjective like "blonde" is monotone:

if all X are Y, then blonde X are blonde Y,

whereas Boolean negation is antitone:

if all X are Y, then non-Y are non-X.

Should all logical items be monotone or antitone? The problem is that this would rule out many reasonable candidates. For instance, the quantifier *one thing* ($\exists!x$) is not monotone, either way (even though it has some related substitutes; see [3]). Therefore, we do not commit ourselves to this general requirement.

Nevertheless, there is an interest in studying monotone logical items. And in fact even stronger notions may be formulated, inspired by the 'double monotonicity' of the standard quantifiers. Recall that all types could be written in the form

$(a_1,(a_2,\ldots,(a_n,y)\ldots))$, with y some primitive type.

Let us call an item f in this type *totally monotone* if

$x_1 \sqsubseteq y_1,\ldots,x_n \sqsubseteq y_n$ imply $f(x_1)\ldots(x_n) \sqsubseteq f(y_1)\ldots(y_n)$.

And similar notions are possible with combinations of monotone and antitone behavior in all arguments. It would be of interest to classify all permutation-

invariant, totally monotone items in all types, thus generalizing the earlier characterization of the standard quantifiers. By way of example, the special case of binary operations on predicates may be taken. Here, the totally monotone items are essentially the expected ones:

$$f_1(X, Y) = X$$
$$f_2(X, Y) = Y$$
$$f_3(X, Y) = X \cap Y$$
$$f_4(X, Y) = X \cup Y.$$

Remark The preceding discussion of monotonicity has been ambivalent between two points of view. On the one hand, language-independent items can be monotone, according to the definition presented. On the other hand, linguistic expressions can *occur* monotonely in other expressions, in an obvious derived sense, via the corresponding denotations. There are some questions here, concerning the syntactic criteria enabling us to *recognize* when a given occurrence is monotone (with respect to all interpretations.) This is actually a more general issue. Even with an adequate set of semantic criteria for logicality, it might still be a difficult, perhaps even an *undecidable* matter, to show that any given linguistic expression satisfies them. An example will be found in the Appendix to Section 3.1.

Boolean Homomorphisms It is of interest to strengthen monotonicity towards the preservation of further Boolean structure. Notably, Keenan and Faltz [17] stress the importance of Boolean *homomorphisms*, respecting the natural Boolean structure which is present on all domains corresponding to the earlier Boolean types (ending in t):

$$f(x \sqcap y) = f(x) \sqcap f(y)$$
$$f(x \sqcup y) = f(x) \sqcup f(y)$$
$$f(-x) = -f(x).$$

(Monotonicity is then a derived property.) Examples of homomorphic behavior may be found, e.g., with proper names:

Judith (jokes and juggles) ↔ (Judith jokes) and (Judith juggles)
Judith (jokes or juggles) ↔ (Judith jokes) or (Judith juggles)
Judith (does not jive) ↔ not (Judith jives).

But it may also occur with prepositions, and other types of linguistic expression. In particular, there are some interesting *logical* homomorphisms. Here is one example. The earlier relation reducer "self" is a homomorphism in its category, as may be seen in such equivalences as:

(hate and despise) oneself ↔ (hate oneself) and (despise oneself)
(not trust) oneself ↔ not (trust oneself)).

In fact, the following observation explains the special position of this item:

Proposition *Reflexivization is the only permutation-invariant Boolean homomorphism in the type* $((e,(e,t)),(e,t))$.

Proof: We assume an individual domain with a sufficiently large number of elements so as to avoid trivialities. The argument presented here shows a typical way of determining the constraints induced by these conditions. So. let f be any permutation-invariant Boolean homomorphism in the type $((e,(e,t)),(e,t))$.

(1) As homomorphisms preserve *disjunction*, the following reduction is possible, for any relation R,

$$f(R) = \bigcup_{(x,y)\in R} f(\{(x,y)\}).$$

(2) Now, for these singleton relations, there are two cases: (I) '$x \neq y$' and (II) '$x = y$'. In both of them, the value of f is severely restricted by *permutation invariance.* In fact, the possibilities are as follows:

(I) $\emptyset$, $\{x\}$, $\{y\}$, $\{x,y\}$, $D_e - \{x,y\}$, $D_e - \{x\}$, $D_e - \{y\}$, D_e
(II) $\emptyset$, (x), $D_e - \{x\}$, D_e.

(3) Homomorphisms have another preservation property: they preserve *disjointness* of arguments. Together with permutation invariance, this rules out all possibilities for (I) except $\emptyset$.

Example If $f(\{(x,y)\}) = \{x\}$, then, by a suitable permutation leaving x fixed and sending y to some $y' \neq y$, $y' \neq x$, $f(\{(x,y')\}) = \{x\}$ also holds, whereas $\{(x,y)\}$ and $\{(x,y')\}$ are disjoint relations. If $f(\{(x,y)\}) = \{x,y\}$, then, by a suitable permutation, $f(\{(y,x)\}) = \{x,y\}$ too, again contradicting the preservation of disjointness. Etc. By similar arguments, one can rule out the third and fourth possibilities in Case (II).

(4) Finally, since homomorphisms cannot give value 0 everywhere, the value $\emptyset$ cannot be chosen in Case (II). So we have value $\emptyset$ in Case (I) and value $\{x\}$ in Case (II). But, the resulting formula is precisely the description of reflexivization. This concludes the proof.

On the other hand, logical homomorphisms are absent in other important types. For instance, they are not found in the determiner type $((e,t),((e,t),t))$: no plausible quantifiers are Boolean homomorphisms.

The reason behind both observations may be found in Chapter 3 of [7], which presents the following *reduction*:

> Boolean homomorphisms in type $((a,t),(b,t))$ correspond one-to-one in a natural fashion with *arbitrary functions* in the lower type (b,a).

In fact, given some function f in type (b,a), the corresponding homomorphism F in $((a,t),(b,t))$ is described by the formula

$$\lambda x_{(a,t)} \cdot \lambda z_b \cdot \exists y_a \in x \cdot f(z) = y_a.$$

Now, some calculation will show that in this correspondence F will be permutation-invariant if and only if f is. So, to find logical homomorphisms in $((e,(e,t)),(e,t))$, or equivalently, in $((e\cdot e,t),(e,t))$, we have to find permutation-invariant functions in type $(e,e\cdot e)$. And, as was observed before, of those there was only one, the 'duplicator', which indeed induces reflexivization via the above

formula. Likewise, homomorphic logical determiners would have to correspond to permutation-invariant (choice) functions in type $((e,t),e)$. But, as we said before, of the latter there are none.

Continuity One feature of homomorphisms in fact suggests a notion of wider applicability. Call a function *continuous* if it respects arbitrary unions/disjunctions:

$$f(\sqcup_i x_i) = \sqcup_i f(x_i).$$

As in the proof of the preceding Proposition, this expresses a certain locality, or 'pointwise' calculation of f: it is enough to collect values computed at singleton arguments.

Continuity, while stronger than Monotonicity, is valid for quite a few important logical constants. For instance, together with permutation invariance, it is used in [6] to analyze the *Quine operations* reducing binary to unary predicates. Essentially, in addition to the earlier-mentioned reflexivization, one obtains *projection*:

$$\text{proj}_1(R) = \{x \mid \exists y \cdot (x,y) \in R\} \quad (= \text{domain } (R))$$
$$\text{proj}_2(R) = \{x \mid \exists y \cdot (y,x) \in R\} \quad (= \text{range } (R)).$$

To obtain the broader case of nonreducing operations on binary relations, one needs a result from van Benthem ([7], p. 22):

> *The continuous permutation-invariant items in type* $((e,(e,t)),(e,(e,t)))$ *are precisely those definable in the following format*: $\lambda R_{(e,(e,t))} \cdot \lambda x_e \cdot \lambda y_e \cdot \exists u_e \cdot \exists v_e \cdot \langle$*Boolean combination of identities in* $x,y,u,v\rangle$.

This includes such prominent examples as Identity, Converse, and Diagonal.

Continuity may also be used to bring some order into the possibilities for logical operations in a *Relational Calculus*, being an algebraic counterpart to predicate logic (see Section 5 below for more on this).

3.2 Inverse logic In certain ways, the preceding discussion has also introduced a viewpoint whereby logical constants are approached through their role in validating patterns of *inference*. After all, the various notions having to do with the interaction with general inclusion, $\sqsubseteq$, may be thought of as inference patterns, involving the particular logical constant under consideration as well as Boolean "and", "or", "not", etc.

In this light, for instance, the earlier characterization of reflexivization may be viewed as a piece of 'inverse logic'. Usually, one starts from a logical item, and determines the inferences validated by it. Here, inversely, we gave the inferences, and found that only one logical constant in type $((e,(e,t)),(e,t))$ would validate the set of patterns

$$f(X \cap Y) \leftrightarrow f(X) \cap f(Y)$$
$$f(X \cup Y) \leftrightarrow f(X) \cup f(Y)$$
$$f(-X) \leftrightarrow -f(X).$$

Actually, these questions have been considered in more detail for the special case of generalized quantifiers (see [3]). Here one can study various sets of *syl-*

logistic patterns, and see if they determine some particular logical constant. Several outcomes are relevant for our general picture:

- The basic patterns here are not Boolean but purely *algebraic*, as in the forms

$$\frac{QXY}{QYX}\ \text{(Conversion)}\qquad \frac{QXY \quad QYZ}{QXZ}\ \text{(Transitivity)}$$

- Outcomes may indeed be unique, as in the result that Conversion and Reflexivity essentially characterize the quantifier "some" (modulo some reasonable assumptions).
- Without any further limiting assumptions, inferences may also *underdetermine* logical constants, in that different candidates would qualify. (In terms of pure syllogisms, for instance, "at least two" validates the same patterns as "some".) In fact, there is also a positive side to such underdetermination: different 'solutions' to a set of inferential patterns may exhibit a useful *duality*. The latter happens, for instance, with the algebraic inferences governing the Boolean operations on sets (cf. [6]): conjunction and disjunction cannot be told apart.
- Finally, there are also prima facie plausible inferential patterns which admit of no logical realization. For instance, there are no nontrivial permutation-invariant quantifiers validating the syllogistic pattern of *Circularity*:

$$\frac{QXY \quad QYZ}{QZX}\ .$$

Inverse logic is also possible in other types, however. One instance is provided by the above reference to operations on predicates. We shall return to such further examples in Section 5 below.

3.3 Computability Yet another perspective upon logical constants is provided, not by their semantic meaning, or their syntactic deductive power, but by the complexity of the *computations* needed to establish their truth. For instance, would a certain *simplicity* be one of their distinguishing traits?

Again, there may be no general argument for this view, but it can still serve as an interesting principle of classification. For instance, there is a natural hierarchy of computability for generalized quantifiers (see Chapter 8 of [7] on the relevant 'semantic automata'). Such quantifiers may be computed by *automata* that survey all individuals in the domain, marked for (non-)membership of the relevant two predicates to which the quantifier applies, and then, after this inspection, produce a truth value YES or NO. In this way, the well-known Automata Hierarchy comes into play. Notably, the lowest level of *finite state machines* turns out to suffice for computing all *first-order* quantifiers definable in standard predicate logic. Nonstandard higher-order quantifiers, such as "most", will in general require computation by means of push-down store automata having unbounded memory. Thus, the distinction between more traditional and other logical constants also turns out to have a computational basis.

In line with the general spirit of this paper, the question arises if such a computational perspective can be generalized to all semantic types. And, indeed, there are unmistakable computational aspects to other types of linguistic expressions. For instance, assigning adjectives like "tall" to individuals seems to presuppose systematic location in some comparative order of being-taller-than. And a modifier like "very" again prescribes an almost numerical 'intensification' in order to assign a predicate like "very tall". (See [12] for some broader developments of the automata perspective in other types.) Nevertheless, no totally convincing generalization has yet emerged. Whatever the precise outcome, we would expect the basic logical constants to occur at some moderate computational level.

4 Definability There is a standard logical language used for interpretation in the type-theoretic models employed up until now. Its major, well-known syntactic operations are *application, lambda abstraction*, and (perhaps) *identity*. One natural question is how far this language can serve as a uniform medium for defining logical constants. We shall look into this matter in Section 4.1.

There is, moreover, a more 'auxiliary' use of this language. As has been observed repeatedly, it seems as if 'the same' logical constant can occur in different guises. This *polymorphism* of, e.g., Boolean operators, but also of quantifiers or identity, may be described systematically using such a type-theoretic language. Section 4.2 contains an elaboration of this second theme.

4.1 Type-theoretic definitions Let us consider a language with variables for each type a, and the following rules of term construction for a typed lambda calculus:

- if τ is a term of type (a,b) and σ one of type a, then $\tau(\sigma)$ is a term of type b
- if τ is a term of type b and x a variable of type a, then $\lambda x \cdot \tau$ is a term of type (a,b).

Sometimes we shall also use a third rule, to obtain a *full theory of types*:

- if τ, σ are terms of the same type, then $\tau \equiv \sigma$ is a term of type t.

All these terms have standard interpretations in our earlier type hierarchies.

One immediate connection between terms in this language and logical constants is the following:

> all closed terms in the theory of types define *permutation-invariant* objects in their type.

This follows from the following observation about arbitrary terms, which is easily proved by induction:

Proposition *For every term τ with the free variables $x_1, \ldots, x_n$, every permutation π (lifted to higher types as usual), and every interpretation function $[\![\ ]\!]$ in a hierarchy of type domains,* $\pi([\![\tau]\!]^{x_1 \ldots x_n}_{d_1 \ldots d_n}) = [\![\tau]\!]^{x_1 \ldots x_n}_{\pi(d_1) \ldots \pi(d_n)}$.

The converse does not generally hold, e.g., infinite models will have uncountably many permutation invariants, outrunning the countable supply of type-theoretic terms. Nevertheless, the correspondence is one-to-one in an important special case (see [10]):

Proposition *In a type hierarchy starting from a* finite *domain of individuals, every permutation-invariant item in any type is definable by some closed type-theoretic term of that type.*

Thus, in a sense, studying further restrictions on logicality may be translated into searching for reasonable *fragments* of the full type-theoretic language.

One obvious fragment is that of the typed lambda calculus, which has much independent interest. This language does not suffice by itself for defining all permutation invariants; even so, it has remarkable powers of definition. One illustration concerns the *functional completeness of Boolean operators.* As all beginners learn, the standard logical constants "not", "and", and "or" suffice for defining all truth-functional connectives. In our type-theoretic perspective, this means that all 'first-order' pure t-types have their items defined using only three particular constants from the types (t,t) and $(t,(t,t))$. But what about higher Boolean types, such as $((t,t),t)$ ('properties of unary connectives'), etc.? Perhaps surprisingly, the above few constants still suffice, in the following sense (see [9]):

Proposition *Every item in the pure t-hierarchy is definable by some closed term of the typed lambda calculus involving only the constants* $\neg, \wedge (\vee)$.

Moreover, it is not hard to extend this result to cover the case of an arbitrary finite domain D_t ('many truth values'), with respect to some suitably enlarged set of basic connectives.

One interesting interpretation of this result for logical constants is the following. *We* have to supply only a few hard-core logical items at an elementary level: the lambda calculus machinery will take care of the rest.

Of course, within this broad general scheme, one can also consider much more detailed questions of functional completeness. For instance, it has often been observed that there is no predicate-logical analogue of the above functional completeness result for Boolean connectives. In what sense could one say that the standard first-order quantifiers are 'expressively complete'? Here, our earlier results provide an answer: The standard first-order formalism is certainly expressively complete for *doubly monotone* quantification (and indeed, for some wider forms too; see [3]).

Next, we consider the effect of another general desideratum on logical constants, viz. the *context neutrality* of Section 2.2. There it turned out to be convenient to shift to a relational perspective, as will be done here as well. Moreover, it will also be useful to change over to another type-theoretic language, having the usual quantifiers $\exists$ and $\forall$ (over all types). (The lambda operator then becomes redundant, in a sense.) We shall say that a t-type formula $\phi = \phi(x_a)$ *defines* an item f of type a in some model if f is the only object satisfying the statement ϕ.

When is such a definition ϕ context-neutral, in the sense of the following relativization?

Let ϕ define f in a model constructed on D_e,
and f^+ in the model constructed on $D_e^+ \supseteq D_e$.
Then $f^+ | D_e = f$.

The following gives at least a sufficient condition:

Proposition *Let ϕ define a unique object in every model, and let every quantifier occurring in ϕ be relativized, i.e., in the form $\exists x \leq y$, $\forall x \leq y$ (where '$x \leq y$' stands for 'x is a member of some tuple in y'). Then ϕ defines a context-neutral denotation.*

Example The universal quantifier is defined by the restricted formula

$$\forall y \leq x \, \forall z \leq x (x(y,z) \leftrightarrow \forall u \leq y : u \leq z).$$

The above condition is not necessary, however. In fact, it only produces *predicative* examples, referring to 'subobjects' of the argument x. In order to obtain context-neutral items such as the quantifier "most", one has to allow impredicative definitions ϕ too, referring to higher types, provided that they remain within the subhierarchy upward generated by the (transitive $\in$-closure of the) argument x. (Incidentally, this predicative/impredicative distinction itself provides another suggestive classification of logical constants.)

We conclude with a question (cf. [20]). The extensive use of type-theoretical languages itself raises a new issue of logicality. What is the logical status of *transcendental operations*, like application, lambda abstraction (or definite description, etc.)?

4.2 Changing types Some logical constants seem to cross boundaries between types, living in different guises. For instance, we saw in Section 3 how "self" in type $((e,(e,t)),(e,t))$ could be derived from duplication in type $(e,e \cdot e)$. Likewise, the basic identity between individuals in type $(e,(e,t))$ can also occur in type $(((e,t),t),(e,t))$, operating on complex noun phrases (as in "be a man"). Again, there occurs a 'canonical' transfer of meaning, as was observed by Montague:

$\lambda x_{((e,t),t)} \cdot \lambda y_e \cdot x(\lambda z_e \cdot \mathrm{BE}_{(e,(e,t))}(z)(y))$
('y is a man' if 'a man' holds for 'being y').

And finally, Boolean operations in higher types can be derived from their base meanings in the truth tables. A case in point is the metamorphosis from sentence negation to predicate negation:

$$\lambda x_{(e,t)} \cdot \lambda y_e \cdot \mathrm{NOT}_{(t,t)}(x(y)).$$

There is a system to such changes, as will be seen now.

In fact, *type changing* is a general phenomenon in natural language that shows many systematic traits (see Chapter 7 of [7], and [10]). We shall outline a few points that will be necessary for our further investigation of logical constants.

Generally speaking, expressions occurring in one type a can move to another type b, provided that the latter type is *derivable* from the former in a

logical calculus of *implication* (and perhaps conjunction). The basic analogy operative here is one discovered in the fifties: Functional types (a,b) behave very much like implications $a \to b$. Then, transitions as mentioned above correspond to derivations of valid consequences in implicational logic.

Example (Derivations are displayed in natural deduction trees)

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

$$\dfrac{\dfrac{\dfrac{\overset{1}{e} \quad \overset{2}{(e,t)}}{t} \quad (t,t)}{t}\ \text{withdraw 1}}{(e,t)}\ \text{withdraw 2}$$
$$((e,t),(e,t))$$

- $(e,(e,t)) \Rightarrow (((e,t),t),(e,t))$ is quite analogous:

$$\dfrac{\dfrac{\dfrac{\overset{1}{e} \quad (e,(e,t))}{(e,t)} \quad \overset{2}{((e,t),t)}}{t}\ \text{withdraw 1}}{(e,t)}\ \text{withdraw 2}$$
$$(((e,t),t),(e,t))$$

- $(e,e\cdot e) \Rightarrow ((e,(e,t)),(e,t))$ becomes analogous again, if we rewrite it as $(e,e\cdot e) \Rightarrow ((e\cdot e,t),(e,t))$.

Thus, the derivational analysis shows a common pattern in all three examples, being a form of Transitivity:

$$(x,y) \Rightarrow ((y,z),(x,z)).$$

In general, again, admissible type changes in natural language correspond to valid derivations in a *constructive* implicational logic, given by the usual natural deduction rules of modus ponens and conditionalization. Also frequent, in addition to the above inference of Transitivity (often called 'Geach's Rule' in this context), are the so-called rules of Raising (also called 'Montague's Rule'):

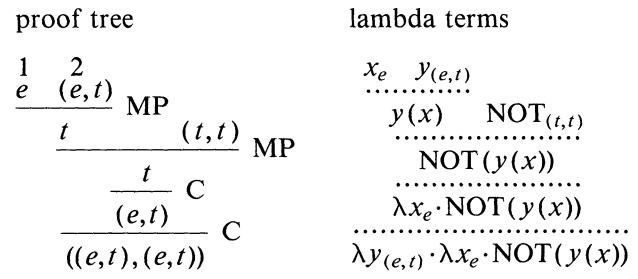
$$x \Rightarrow ((x,y),y).$$

For instance, the latter pattern is exhibited by proper names (type e) which start to behave like complex noun phrases, or semantically as 'bundles of properties':

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

Moreover, these derivations are not purely syntactic. For they correspond one-to-one with terms from the typed lambda calculus, explaining how denotations in the original type are changed into denotations in the new type. Here is an illustration for Boolean negation:

Example



Note how application encodes modus ponens, and lambda abstraction encodes conditionalization.

Thus, we see how logical constants can move from one category to another, provided that the corresponding change of meaning can be expressed using some 'wrappings' of the typed lambda calculus. Indeed, any object can undergo such type changes, as was observed above. And in the process it may become 'embellished', acquiring some logical traits it did not have before. (For instance, plain individuals in type e become *Boolean homomorphisms* in type $((e,t),t)$; cf. [17]).

The type changing perspective raises many new questions in connection with the analysis of logicality presented in Sections 1 and 2. Suppose that some logical item in a category has the properties discussed earlier. Will it retain them after its change of type/meaning? In more technical logical terms, which of the earlier semantic properties are *preserved* (or acquired) *under type change*?

To begin with, we have seen already that *permutation invariance* is indeed preserved. It follows directly from the earlier results that, if f is invariant, any term $\tau(f)$ will also define a permutation-invariant item. (We shall not inquire here into a possible *converse* of this, or of later results.)

Matters are more complex with *monotonicity*. Some type changes preserve it; the earlier Geach Rule is an example. Others do not; the Montague Rule is an example. What is required in general for such preservation is that the parameter x_a for the item being changed occur only *positively* in the defining term (for a fuller discussion, see [9]).

And finally, little is known yet concerning preservation or creation of such properties as *continuity* or being a Boolean *homomorphism*.

What this analysis stresses, in any case, is a perhaps unexpected aspect of logicality: it can be *gained* or *lost* to some extent in the process of type change. Thus, our world is much more dynamic than may have been apparent at the outset.

Remark The preceding account does not exhaust the story of polymorphism in natural language. On the one hand, the constructive logic system may be too rich, in that admissible type changes (mostly) fall within weaker calculi, and associated fragments of the typed lambda calculus. For instance, there is an important subsystem, due to Lambek, which may be closer to the mark, and which also possesses a logical theory with some nicer features than the full lambda calculus. (See Chapter 7 of [7], [9], and [10] on the topic of preserving monotonicity in this setting.) On the other hand, the type changes studied up

until now may be too poor, in that certain important phenomena are not represented. For instance, the system as it stands does not account for the similarity of, say, the existential quantifiers in

$$\exists x_e \cdot y_{(e,t)}(x) \qquad \text{(type } ((e,t),t)\text{)}$$
$$\exists x_{(e,t)} \cdot y_{((e,t),t)}(x) \qquad \text{(type } (((e,t),t),t)\text{)}.$$

A proper treatment here may require genuine variable polymorphism, assigning the following type to quantifiers:

$$((x,t),t).$$

Compare the discussion of generalized permutation invariance in Section 2 for a possible semantic background for this move.

5 Extensions The treatment so far may have given the impression that the type-theoretic analysis of logicality is restricted to handling *extensional* items in an $\{e,t\}$-based framework. This is far from being the truth, however. There is no problem whatsoever in adding further primitive types; in particular, a type s for possible worlds or situations. In this final section, we will survey some illustrations of how the earlier themes re-emerge in *intensional* settings.

5.1 Intensional logic The logical constants of traditional intensional logic exhibit a natural type structure, when viewed in the proper light. Thus, with propositions identified as usual with functions from possible worlds to truth values (i.e., in type (s,t)), *modal operators* have type $((s,t),(s,t))$, while *conditionals* have the binary type $((s,t),((s,t),(s,t)))$.

It is quite feasible to subject such types to denotational analysis, much as we did in previous sections. In fact, there are strong formal analogies between the cases of $\{e,t\}$ and $\{s,t\}$, as might be expected. There are also differences, however, between intensional operators and the earlier logical constants. For instance, the *permutation-invariant* items in the above types will be just the *Boolean operations*, as was established in Section 2. And due to the results of Section 4 we know that these are not 'genuine' inhabitants of $((s,t),(s,t))$, etc., but rather transmuted versions of items in the simpler, s-less types (t,t) and $(t,(t,t))$. So, genuine intensional operators cannot be permutation-invariant. In the terms of Section 2, they have to be sensitive to some further structure on the D_s-domain (being invariant only with respect to *automorphisms* of that structure). But this is reasonable, of course, reflecting precisely the usual approaches in intensional logic, which assume some structure like "accessibility" between possible worlds or "extension" among situations. Of course, the systematic question then becomes how to motivate (a minimum of) such additional structure independently.

More detailed studies of the above genesis of intensional operators may be found in [2] and [4]. Here it may suffice to remark that all earlier concerns of monotonicity, Boolean structure, or type change still make sense in this setting. For instance, we can also classify possible modal operators or conditionals by their patterns of inference. A particularly concrete example of all this occurs with temporal operators, where D_s represents points in time carrying an obvious ordering of *temporal precedence* (cf. [8]). Tenses and temporal adverbs may be

viewed as operators on propositions that are invariant for automorphisms of the temporal order. For instance, the basic Priorean tenses on the real numbers (viewed as a time axis) are exactly those $<$-automorphism-invariant ones that are *continuous* in the sense of Section 3. Relaxing this restriction to mere monotonicity will then bring in the other tenses studied in the literature.

Of course, the presence of additional structure will give many of the earlier topics a new flavor. For instance, what is invariant for temporal automorphisms may vary considerably from one picture of time to another, since different orderings may have quite different sets of automorphisms. Changing from the reals to the *integers*, therefore, already affects the class of tenses in the above-mentioned result, because the latter structure is so much poorer in automorphisms. (As a consequence, many more operators qualify as 'tenses' on the integers, including such items as "yesterday" and "tomorrow".) Another interesting new aspect is the possible action of semantic *automata* on time lines (cf. [18]).

5.2 Dynamic logic A similar extension to currently popular 'dynamic' logics is possible. These logics were originally developed in the semantics of *programming languages*, but now also serve as models for the more dynamic, sequential aspects of interpreting natural language.

The basic domain D_s will now represent *states* of some computer, or knowledge states of a person. Propositions may then be viewed as *state changers*, (in the simplest case) adding information to obtain a new state from the current one. This will give them the type (s,s) when viewed as *functions*, or $(s,(s,t))$ when viewed merely as *relations* ('many-valued functions'). Logical constants will now be the basic operations combining such functions or relations into complex ones. Obvious examples are the analogues of the earlier Boolean operations, but also typical 'dynamic' operators, such as *sequential composition*, will qualify.

One obvious question here is what would be a reasonable choice of basic logical items, given the broader options now available. What one finds in practice is often some variant of the operations in the usual *Relational Calculus* on binary relations. Is there some more basic justification for this? In any case, our earlier notions can be brought to bear. As is easily checked, all operations in the Relational Calculus are *permutation-invariant* (with respect to permutations of D_s, that is) precisely in the earlier sense, and also *continuous*. And the set of all possibilities within this class can be enumerated just as in Section 3.1, using a suitable 'lambda schema'. We forego spelling out all technical details here—but the general outcome is that the basic items are indeed those found in the usual literature, as may be seen in the following illustration.

Example Here are a few outcomes of simple denotational analysis in this setting, with programs considered as transition relations between states, that is, in type $(s,(s,t))$.

(1) Logical continuous binary *operations on programs* must have the form

$$\lambda R.\lambda S.\lambda xy.\exists zu.Rzu \wedge \exists vw.Svw \wedge$$
$$\langle \text{some Boolean condition on identities in } x,y,z,u,v,w \rangle.$$

Typical cases are as follows:

Union: $(x = z \wedge y = u) \vee (x = v \wedge y = w)$
Intersection: $x = z = v \wedge y = u = w$
Composition: $x = z \wedge u = v \wedge w = y$.

(2) Some operators take ordinary propositions, in type (s,t), to more program-like counterparts, in type $(s,(s,t))$. One example of such a *dynamic propositional mode* is the ordinary *test* operator ?. Its definition again satisfies the relevant schema for logical continuity:

$$\lambda P_{(s,t)}.\lambda xy.\exists u.(Pu \wedge y = x = u).$$

Stronger requirements on the preservation of propositional structure will lead to a collapse in options here. For instance, logical *homomorphisms* in type $((s,t), (s,(s,t)))$ must correspond with logical functions in the type $(s.s,s)$, by the analysis given in Section 3.1. But, of the latter, there are only two, namely left- and right-projection, generating only the following marginal cases:

$$\lambda P \cdot \lambda xy.Px \quad \text{and} \quad \lambda P.\lambda xy.Py.$$

(3) Eventually, as in Section 5.1, this setting also requires contemplation of additional primitive structure between states—such as 'growth of informational content'. Then a more refined analysis of the preceding operations becomes possible, in particular one allowing for more interesting dynamic modes (see [13]).

Another topic of some interest here is the matter of *inverse logic*. How far are particular logical operations in the above extension characterized by their algebraic inference patterns? (For unary operators, such as converse, one has to think now of properties such as the following:

$$FF(R) = R \quad \text{or} \quad FF(R) = F(R).$$

For binary operators, one has the usual commutativity and associativity, as well as several 'interaction principles', as exemplified by

$$(R;S)^{\smile} = (S^{\smile};R^{\smile}).)$$

Is there a unique 'solution' in this case, or are there some interesting dualities still to be discovered?

It should be added that the full picture may be richer yet. Some propositions in natural language may be used to *change* a state, others serve rather to *test* a state for some given property. And such testing of course is also essential in programming languages (compare the control instruction IF . . . THEN . . . ELSE . . .). In such a case, our type structure will also involve propositions in type (s,t) after all. For some type-theoretic exploration of this richer structure, see [10] and [13].

6 Epilogue The professed purpose of this paper has been to analyze various strands in the intuitive notion of logicality, and then to show these at work in the widest possible setting.

Perhaps it is only fair to add explicitly that this is an expression of a view opposed to the traditional idea of regarding logic as being primarily concerned with the study of 'logical constants' (whatever these may be). Logic, in our view,

is concerned with the study of *logical phenomena*: and these occur all across language, not just with any distinguished group of actors.

This view is more in line with that of Bernard Bolzano, who saw the task of logic as providing a liberal study of various mechanisms of consequence (cf. [5]). With some adaptations to the twentieth century, this is still an appropriate banner to follow.

Appendix to Section 3.1 In general, questions about the fit between linguistic expressions and semantic properties need not be decidable. This may be illustrated for the simple case of *standard predicate logic.*

Our first example is the important notion of *monotonicity.* As was pointed out by Yuri Gurevich, this semantic property is not decidable in predicate logic. Here is a simple argument to this effect:

Proposition *The question whether the truth value of a given first-order formula* $\phi_L(P)$ *depends monotonically on its predicate argument P is* RE *but not decidable.*

Proof: That this property is recursively enumerable follows from the Lyndon characterization of the monotone $\phi_L(P)$ as those formulas which are provably equivalent to some formula in which P occurs only positively.

The negative statement results from the following reduction of predicate-logical validity:

> Let α be any L-sentence, q some proposition letter not occurring in L. Then α is universally valid if and only if the formula $\alpha \vee \neg q$ is monotone in q.

Only if: $\alpha \vee \neg q$ will be equivalent to *true.*

If: Suppose that $\alpha \vee \neg q$ is monotone in q. Consider any L-model M. Expand it to an $L + q$-model M^+ by making q false. Thus, $M^+ \models \alpha \vee \neg q$. Now change M^+ to a model M^* by making q true. By monotonicity, we still have that $M^* \models \alpha \vee \neg q$. But then $M^* \models \alpha$, and hence $M \models \alpha$, since α refers only to L: on which vocabulary M^* and M agree. So, α is universally valid.

As a second example, consider the notion of *permutation* or, more generally, *automorphism invariance.* In general, a first-order formula $\phi(A,B)$ may express a condition on its component predicates, considered as its semantic arguments, so to speak. But there will only be genuine dependence on such a predicate if the truth value of the formula is sensitive to changes in the behavior of that predicate. Otherwise, there will be independence—and one way of defining the latter notion is as follows:

Definition $\phi(A,B)$ is *independent* from B if, for all bijections π between models $M_1 = (D_1,A_1,B_1)$ and $M_2 = (D_2,A_2,B_2)$ which are A-isomorphisms, $M_1 \models \phi$ if and only if $M_2 \models \phi$.

Note that this is still connected with the earlier uses of permutation invariance. We can also say that $\phi(A,B)$ is independent from B if its induced generalized quantifier predicate $\lambda A.\phi(A,B)$ in any model is invariant for permutations of individuals, even if these do not respect B.

Proposition *Independence from component predicates is a* RE *but not decidable notion for first-order formulas.*

Proof: Consider an arbitrary L-formula α, and let q be a new proposition letter outside of L. This time, the relevant effective reduction is that:

α is universally valid if and only if the formula $\alpha \vee q$ is independent from q.

Only if: $\alpha \vee q$ will be true in M_1 and M_2 no matter what q does.

If: Suppose that α is not universally valid: e.g., it fails in $M = (D,L)$. Then the two expanded models $(M,$ 'q true') and $(M,$ 'q false') are L-isomorphic via the identity map, and yet they disagree on $\alpha \vee q$—whence $\alpha \vee q$ is not independent from q.

As to an upper bound, the RE-ness of independence follows from the next claim:

$\phi(L,B)$ is independent from B if and only if ϕ is provably equivalent to some L-formula not containing B.

The nontrivial 'only if' direction is shown as follows. Let $\mathrm{CONS}_L(\phi)$ be the set of all B-free L-consequences of ϕ. As usual, it suffices to prove that

$$\mathrm{CONS}_L(\phi) \models \phi$$

since an L-equivalent can then be obtained from $\mathrm{CONS}_L(\phi)$ by compactness.

Now, let $M \models \mathrm{CONS}_L(\phi)$. By a standard model-theoretic argument, $\mathrm{Th}_L(M) \cup \{\phi\}$ must be satisfiable, say in some model N. Thus, we have that

$$M \equiv_L N \quad \text{and} \quad N \models \phi.$$

Next, take L-isomorphic elementary extensions M^+, N^+ of M, N respectively. (These exist, via Keisler's Theorem.) Then $N \models \phi$, $N^+ \models \phi$ (by elementary extension), $M^+ \models \phi$ (by independence) and hence $M \models \phi$ (by elementary descent).

Remark An alternative proof would proceed by first observing that independence is in fact equivalent to the following invariance condition:

$$(D,A,B) \models \phi \quad \text{if and only if } (D,A,B') \models \phi.$$

This again translates into the validity of the semantic consequence

$$\phi(A,B) \models \phi(A,B')$$

which, in its turn, is equivalent to ϕ's being purely A-definable, by the Interpolation Theorem for first-order logic.

Appendix to Section 4.2 As we have seen, one major source of transmission, or even creation, of logical behavior is the type-theoretical structure which relates objects in different categories. We will consider some examples here, in order to show how this structure can be investigated systematically.

For instance, which items in the noun-phrase type $((e,t),t)$ are lambda-definable using parameters in the individual domain D_e? Such a question reduces to surveying terms τ in the typed lambda calculus of type $((e,t),t)$ in which only free variables of type e occur. Here, we may always restrict attention to terms in *lambda normal form*, containing no more 'redexes' of the shape

$(\lambda x.\alpha)(\beta)$. Also, in normal forms the types of all variables must be subtypes of the resulting type or of the types of the parameters. With these restrictions, we see that the only candidates which qualify are the earlier Montague liftings:

$$\lambda x_{(e,t)}.x(y_e).$$

This can be stated in terms of the earlier terminology of 'type change': The derivable transition $e \Rightarrow ((e,t),t)$ is *unambiguous*, having essentially just one derivation.

The situation can be more complex, however. For instance, [11] has a discussion of the polyadic quantifiers in type $((e,(e,t)),t)$, as mentioned in Section 1. As was observed before, one case of such quantification arises by merely iterating two unary quantifiers in combination with a transitive verb:

$$Q^1 \mathrm{TV} Q^2.$$

Again, there is a derivable implicational transition involved here, namely

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

But this time there are at least four different derivations, giving rise to different scope readings with direct and passive readings of the verb. One example is the wide scope reading

$$\lambda x_{(e,(e,t))}.Q^1_{((e,t),t)}(\lambda y_e.Q^2_{((e,t),t)}(x(y))).$$

But, what are *all* polyadic quantifiers definable from two unary ones? As it turns out, there are only a finite number of candidates in each model: the remaining polyadics must fend for themselves.

The preceding examples were about various kinds of generalized quantifiers. Let us now take a look at the underlying type of *determiners* $((e,t),((e,t),t))$. One reduction occurred here when we saw how at least homomorphic determiners could be derived from choice functions in type $((e,t),e)$. And in fact there is a Geach transition of the form $((e,t),e) \Rightarrow ((e,t),((e,t),t))$. Thus, there must be a general rule for creating determiners out of choice functions with a lambda recipe matching its derivation. Upon computation, this turns out to be

$$\lambda x_{(e,t)}.\lambda y_{(e,t)}.x(u_{((e,t),e)}(y)),$$

a formula reminiscent of the use of *Hilbert's* ϵ*-symbol*, which turns out to be the same recipe as that found in the analysis of Boolean homomorphisms in Section 3.1, being

$$\lambda x_{(e,t)}.\lambda y_{(e,t)}.\exists z_e \in x.u_{((e,t),e)}(y) = z.$$

But there are also other possibilities; for instance, interchanging the variables x and y in the matrix.

Now, let us consider a more general question concerning this important logical category (cf. [15]):

> When is a semantic determiner lambda-definable from items in lower types; i.e., e, t, (e,t), $((e,t),t)$?

(Incidentally, on an alternative analysis, we should have a determiner type $((e,t).(e,t),t)$, without the 'higher-order' subtype $((e,t),t)$.) The argument here will

illustrate a useful general method. We can describe all possible forms for determiners with parameters from the types indicated by means of a kind of *context-free grammar*, having symbols (at most)

$$X_a,\ X_a^*,\ C_a,\ V_a$$

for each of the relevant types a. Here, V_a stands for a variable of type a, C_a for a constant (parameter), X_a for any term of type a, X_a^* for such a term that does not start with a lambda. The point of this division will become clear from the rewrite rules for terms in normal form to be presented now. First we have, for all types a,

$$\begin{aligned} X_a &\Rightarrow X_a^* \\ X_a^* &\Rightarrow C_a \\ X_a^* &\Rightarrow V_a. \end{aligned}$$

Next, rules for application or lambda abstraction depend on the actual types being present (recall the above observations about the shape of terms in lambda normal form):

$$\begin{aligned} X_{((e,t),((e,t),t)} &\Rightarrow \lambda V_{(e,t)} . X_{((e,t),t)} \\ X_{((e,t),t)} &\Rightarrow \lambda V_{(e,t)} . X_t \\ X_t &\Rightarrow X^*_{(e,t)}(X_e) \\ X_t &\Rightarrow X^*_{((e,t),t)}(X_{(e,t)}) \\ X_{(e,t)} &\Rightarrow \lambda V_e . X_t. \end{aligned}$$

The description of possible constructions is much facilitated here because we can make this grammar *regular*. This may be visualized in the following *finite state machine* (Figure 3), where 'D_a' stands for C_a or V_a, where applicable. This scheme produces determiner denotations of forms such as the following:

1. $\lambda x_{(e,t)} . c_{((e,t),t)}$
2. $\lambda x_{(e,t)} . \lambda y_{(e,t)} . c_{((e,t),t)}(x)$
3. $\lambda x_{(e,t)} . \lambda y_{(e,t)} . c_{((e,t),t)}(\lambda z_e . c'_{((e,t),t)}(\lambda u_e . x(z)))$.

Here, the latter kind is 'iterative', producing infinitely many forms by repeating the $c'_{((e,t),t)}(\lambda u_e .)$ subroutine. Thus, globally, there are *infinitely* many dis-

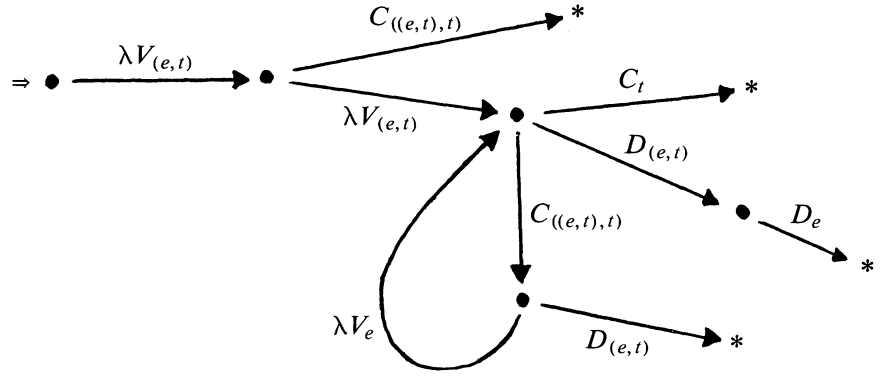


Figure 3.

tinct possibilities for constructing determiners. (Finitely many constructions will occur if the above grammar is *acyclic*, i.e., containing no iterative loops.)

Nevertheless, this global infinity is still 'locally finite', a phenomenon already mentioned in connection with polyadic quantifiers; for all the above forms are equivalent, in any model, to only a *finite* number of cases.

The reason is as follows. Any scheme of definition will start with an initial $\lambda x_{(e,t)}$, and then some further term τ of type $((e,t),t)$. Now, if the latter contains no free occurrences of the variable $x_{(e,t)}$, then it defines some fixed object, which also has one parameter $c_{((e,t),t)}$ denoting it. Hence, we are in the above case 1. Next, if the variable $x_{(e,t)}$ does occur in τ, then, analyzing the latter term one step further, we can rewrite the whole scheme of definition as

$$\lambda x_{(e,t)} . \lambda y_{(e,t)} . [(\lambda z_{(e,t)} . \tau[z/x])(x)],$$

where the subterm $\lambda z_{(e,t)} . \tau[z/x]$ does not contain any free occurrence of the variable $y_{(e,t)}$. (To see this, check the 'exit routes' in the above machine diagram.) Now, this subterm again denotes some fixed object in the $((e,t),t)$ type domain, and hence we arrive at the above form 2.

Therefore, the general result becomes this: Terms of the first two kinds listed above represent the only ways of constructing determiners from objects in lower types.

This outcome tells us that determiners admit of no nontrivial reductions to lower types: they are a genuinely new semantic category in the type hierarchy.

Evidently, this is just one of a number of questions about reducibility in type domains that may be investigated. For instance, can we prove general 'hierarchy results' on definability?

REFERENCES

[1] van Benthem, J., "Determiners and logic," *Linguistics and Philosophy*, vol. 6 (1983), pp. 447–478.

[2] van Benthem, J., "Foundations of conditional logic," *Journal of Philosophical Logic*, vol. 13 (1984), pp. 303–349.

[3] van Benthem, J., "Questions about quantifiers," *The Journal of Symbolic Logic*, vol. 49 (1984), pp. 443–466.

[4] van Benthem, J., *A Manual of Intensional Logic*, CSLI Lecture Notes 1, Center for the Study of Language and Information, Stanford University, 1985. (Second revised edition, 1988.)

[5] van Benthem, J., "The variety of consequence, according to Bolzano," *Studia Logica*, vol. 44 (1985), pp. 389–403.

[6] van Benthem, J., "A linguistic turn: New directions in logic," pp. 205–240 in *Proceedings of the 7th International Congress of Logic, Methodology and Philosophy of Science, Salzburg 1983*, edited by R. Marcus et al., North Holland, Amsterdam, 1986.

[7] van Benthem, J., *Essays in Logical Semantics*, Studies in Linguistics and Philosophy 29, Reidel, Dordrecht, 1986.

[8] van Benthem, J., "Tenses in real time," *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, vol. 32 (1986), pp. 61–72.

[9] van Benthem, J., "Categorial grammar and lambda calculus," pp. 39–60 in *Mathematical Logic and its Applications*, edited by D. Skordev, Plenum Press, New York, 1987.

[10] van Benthem, J., "Categorial grammar and type theory," Report 87-07, Institute for Language, Logic and Information, University of Amsterdam (to appear in *Journal of Philosophical Logic*).

[11] van Benthem, J., "Polyadic quantifiers," Report 87-04, Institute for Language, Logic and Information, University of Amsterdam. *Linguistics and Philosophy*, vol. 12 (1989), pp. 437–464.

[12] van Benthem, J., "Towards a computational semantics," pp. 31–71 in *Generalized Quantifiers: Linguistic and Logical Approaches*, edited by P. Gärdenfors, Studies in Linguistics and Philosophy 31, Reidel, Dordrecht, 1987.

[13] van Benthem, J., "Semantic parallels in natural language and computation," pp. 331–375 in *Logic Colloquium, Granada 1987*, edited by H-D. Ebbinghaus et al., Studies in Logic, North Holland, Amsterdam, 1989.

[14] van Benthem, J., "Modal logic and relational algebra," Institute for Language, Logic and Information, University of Amsterdam, 1989.

[15] van Benthem, J., "The fine-structure of categorial semantics," to appear in *Computational Linguistics and Formal Semantics*, edited by M. Rosner, Cambridge University Press, 1990.

[16] van Benthem, J. and K. Doets, "Higher-order logic," pp. 275–329 in *Handbook of Philosophical Logic, Vol. I*, edited by D. Gabbay and F. Guenthner, Reidel, Dordrecht, 1983.

[17] Keenan, E. and L. Faltz, *Boolean Semantics for Natural Language*, Studies in Linguistics and Philosophy 23, Reidel, Dordrecht, 1985.

[18] Löbner, S., "Quantification as a major module of natural language semantics," pp. 53–85 in *Studies in Discourse Representation Theory and the Theory of Generalized Quantifiers*, edited by J. Groenendijk, D. de Jongh and M. Stokhof, GRASS Series 8, Foris, Dordrecht, 1987.

[19] Quine, W. V. O., "Variables explained away," pp. 227–235 in *Selected Logical Papers*, Random House, New York, 1966.

[20] Smirnova, E. D., *Logical Semantics and the Philosophical Foundation of Logic*, Publishing House of Moscow State University.

Faculty of Mathematics and Computer Science
University of Amsterdam
Plantage Muidergracht 24
1018 TV Amsterdam
The Netherlands