Institute for Language, Logic and Information

LOGICAL CONSTANTS ACROSS VARYING TYPES

Johan van Benthem

ITLI Prepublication Series
for Logic and Philosophy of Language LP-88-05

University of Amsterdam

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, amongst 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
(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)). But also, 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 operations 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 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 which change content, while leaving general structure intact. This is actually not one intuition, but a family: 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 behaviour on certain arguments and a vastly different one on 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 be encountered in what follows too.

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) expresses itself at least as follows, for all $X,Y \subseteq D_e$,

all XY if and only if *all* $\pi[X]\ \pi[Y]$,

for all permutations π of D_e.

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

$\pi\ [not\ (X)] = not\ \pi[X]$.

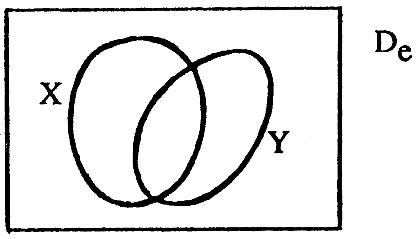
And finally, of course, the relation of identity will be unaffected by such permutations. For all $x,y \in D_e$,

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

Finally, we consider one case where permutation invariance by itself already comes close to singling out the usual logical constants (cf. van Benthem 1986b, chapter 3).

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.*

We sketch the proof, in order to show how the present kind of invariance enforces a certain *uniformity* of behaviour. Consider binary operations, for 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 $\pi(f(X,Y)) \neq f(\pi(X), \pi(Y))$. So, f must choose some union of regions: which is described by a Boolean combination of this 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', shall be called *Boolean types*, since they carry a natural Boolean structure (cf. Keenan & Faltz 1985). 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 had no invariants (provided that there be 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 our type 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 $y_1,\ldots,y_k$, say, then 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: 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 use 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. van Benthem 1986b; in general, the number will depend on the size of the individual base domain). No general result is available, however, enumerating the phenomenon in all types.

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

One is the location of the border-line with *non-invariant* 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. Yet, "all blonde sailors are brave" might be true, whereas "all blonde soldiers are π(brave)" might still 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 which 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, 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. For, 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 van Benthem 1987c, 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 Fregan ((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 of 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": $\exists xy \cdot Hxy$

"noone hates noone": $\neg \exists xy \cdot Hxy$ (!)

These particular dyadic quantifiers define predicates of binary relations which 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 permutations / invariance aspect of logical constants may have been sufficiently established by now.

2.2 Context Neutrality, and Other Uniformities

The invariances considered up till 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 such a context, or environment. For instance, already the earlier account of permutation invariance would not change at all, if we replaced permutations by arbitrary *bijections* between D_e and some other base set.

Another type of neutrality may be observed with generalized quantifiers again. 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"), etcetera. 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 Ys in X is higher than that of the Ys 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 only depend 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 van Benthem & Doets 1983). 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:

- $x \mid D_e = \begin{cases} x & \text{, if } x \in D_e \\ \text{undefined} & \text{, otherwise} \end{cases}$ for individuals x in D_e,
- $R \mid D_e = \{ (r_1,\ldots,r_n) \in R \mid r_i \mid D_e = r_i, \text{ for each } i \ (1 \leq i \leq n) \}$ 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 phenomenon to be observed more generally 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. Already C.S. Peirce observed how predicate logic obeys the following 'Copying Rule' (provided some conditions are observed on freedom and bondage of variables):

$$\begin{aligned} &\dots\ \phi \wedge (\dots\ \psi \dots) \dots \leftrightarrow \\ &\dots\ \phi \wedge (\dots\ \phi \wedge \psi \dots) \dots \quad . \end{aligned}$$

So, 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, behaviour of predicate operators might still be as erratic as this:

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

One might try to exclude such cases by means of stronger general postulates in the above spirit. (Compare the use of "Restriction" in van Benthem 1983.) Nevertheless, these attempts have not been overly succesful. 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 condition, 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 to the behaviour 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' ;

but left-downward:

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

In all, there are four types of monotonicity here, as displayed in the following table, following the Square of Opposition:

$$\begin{tikzcd}
\downarrow \text{every} \uparrow \arrow[drr, dash] & & \downarrow \text{no} \downarrow \arrow[dll, dash] \\
\uparrow \text{some} \uparrow & & \uparrow \text{not every} \downarrow
\end{tikzcd}$$

The interest of this observation shows e.g., in van Benthem (1986b, chapter 1), where it is proved that (modulo some further conditions) this double monotonicity uniquely *determines* the standard quantifiers.

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

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

(It lacks either kind of monotonicity in its left-hand argument, however.) But, 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, here is a general notion of *inclusion*, or *implication*, $\sqsubseteq$ on all types, defined 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 comes out to just 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 rather correspond to an 'inverse correlation' (*anti-tone*):

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

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

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

whereas Boolean negation is anti-tone:

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 van Benthem 1984b.) Therefore, we do not commit ourselves to this general requirement.

Nevertheless, there is an interest to 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 behaviour 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, as to 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 at the end of this Section.

Boolean Homomorphisms

There is an interest to even strengthening monotonicity towards preservation of further Boolean structure. Notably, Keenan & Faltz 1985 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):

$f(x \sqcap y) = f(x) \sqcap f(y)$

$f(x \sqcup y) = f(x) \sqcup f(y)$

$f(-x) \quad = -f(x)$.

[Monotonicity is then a derived property.] Examples of homomorphic behaviour may be found, e.g., with proper names:

Judith (jokes and juggles) $\leftrightarrow$ (Judith jokes) and (Judith juggles),

Judith (jokes or juggles) $\leftrightarrow$ (Judith jokes) or (Judith juggles),

Judith (does not jive) $\leftrightarrow$ not (Judith jives).

But, it occurs also e.g. 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, witness such equivalences as

(hate and despise) oneself $\leftrightarrow$ (hate oneself) and (despise oneself),

(not trust) oneself $\leftrightarrow$ not (trust oneself).

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

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

Proof: We assume an individual domain with a sufficiently large number of element to avoid trivialities. The argument presented here shows a typical form of determining the constraints induced by these conditions. So, let f be any permutation-invariant Boolean homomorphism in 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: being 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 too: they preserve *disjointness* of arguments. Together with permutation invariance, this already 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 $y' \neq y$, $y' \neq x$, it also holds that $f(\{(x,y')\}) = \{x\}$: 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 preservation of disjointness. Etcetera. 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.

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 van Benthem (1986b, chapter 3), 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·e,t),(e,t)), we have to find permutation-invariant functions in type (e,e·e). And, as was observed before, of those, there was only one, being 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:

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

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 still stronger than Monotonicity, is valid for quite a few important logical constants. For instance, together with permutation invariance, it is used in van Benthem 1986a, to analyze the earlier-mentioned *Quine operations* reducing binary to unary predicates. Essentially, in addition to the earlier-mentioned reflexivization, one obtains *projection*:

$$proj_1(R) = \{x \mid \exists y \cdot (x,y) \in R\} \qquad (= \text{domain } (R))$$

$$proj_2(R) = \{x \mid \exists y \cdot (y,x) \in R\} \qquad (= \text{range } (R)) \quad .$$

To obtain the broader case of non-reducing operations on binary relations, one needs a result from van Benthem (1986b, p.22):

The continuous permutation-invariant items in type ((e,(e,t)),(e,(e,t))) are precisely those definable in the following format:

$$\lambda R_{(e,(e,t))} \cdot \lambda x_e \cdot \lambda y_e \cdot \exists u_e \cdot \exists v_e \cdot \text{<Boolean combination of identities in } x,y,u,v\text{>}$$

This includes such prominent examples as Identity, Converse, or 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 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", etcetera.

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, conversely, 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 more in detail for the special case of generalized quantifiers (see van Benthem 1984b). Here, one can study various sets of *syllogistic 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 $\frac{QXY}{QYX}$ (Conversion) $\frac{QXY \quad QYZ}{QXZ}$ (Transitivity)
- outcomes may indeed be unique, as in the result that Conversion and Reflexivity characterize essentially the quantifier "some" (modulo some additional assumptions).
- But, 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 under-determination: 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. van Benthem 1986a): 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 non-trivial 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 van Benthem (1986b, chapter 8), on 'semantic automata'). Such quantifiers may be computed by *automata* which 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. Non-standard higher-order quantifiers, such as "most", will in general require computation by means of push-down store automata. 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 becomes if such a computational perspective can be generalized to all semantic types. And indeed, there are unmistakable computational aspects to other types of linguistic expression. 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 van Benthem 1987d for some broader development of the automata perspective in other types). Nevertheless, noo totally convincing generalization has emerged yet. Whatever the precise outcome, we would expect the basic logical constants to occur at some moderate computational level.

Appendix: Decidability of Semantic Properties

In general, questions about the fit between linguistic expressions and certain semantical properties need not be decidable. This may be illustrated already for the simple case of standard *predicate logic* itself, with respect to our notion of *invariance* from Section 1. We first give some background.

Every formula of predicate logic $\phi = \phi(\bar{P})$ may be viewed as expressing a condition on the predicates $\bar{P}$ occurring in it. Moreover, this condition is permutation-invariant, as may be seen by a simple induction on the construction of ϕ. Suppose that we keep those predicates fixed, however, then ϕ will in general only be invariant for those permutations which are $\bar{P}$-automorphisms.

Example. $\lambda P \cdot \lambda R \cdot \forall x(Px \rightarrow \exists y(Rxy \wedge Py))$ defines a permutation-invariant relation between unary P and binary R. But, the formula $\lambda R \cdot \forall x(Px \rightarrow \exists y(Rxy \wedge Py))$ describes a property of binary relations R which depends on the denotation of P in any model under consideration. Now, formulas can also contain vocabulary on which they do not really depend, as in the further example of

$\lambda R \cdot \forall x \exists y(Rxy \wedge (Py \vee \neg Py))$.

The latter formula is in fact invariant for arbitrary permutations of individuals, whether they respect P or not. Thus, it makes sense to ask, for any predicate-logical formula, which part of its vocabulary it really depends on – or semantically, which degree of automorphism invariance it exhibits.

Thus, the basic question becomes what is the *complexity* of determining, for any predicate-logical formula $\phi(\bar{A}, \bar{B})$, whether it is ***invariant for $\bar{A}$-automorphisms*** already. First, there are 'local' and 'global' versions of the problem. *Locally*, a model is given, and the question is whether ϕ is invariant for $\bar{A}$-automorphisms in that model. With finite base domains, an effective inspection will solve this. With infinite base domains, the problem may already be hard. Often, a more natural version of the problem is a *global* one however:

When is ϕ $\bar{A}$-invariant on *all* models?

First, there is an upper bound on complexity here. By Svenonius' Theorem, global $\bar{A}$-invariance for ϕ is equivalent to *explicit definability* of the remaining B up to disjunction. I.e., there exist formulas $\psi_1, \ldots, \psi_n$ in the vocabulary $\bar{A}$ only such that

$\phi \models \forall\bar{x}(B\bar{x} \leftrightarrow \psi_1) \vee \ldots \vee \forall\bar{x}(B\bar{x} \leftrightarrow \psi_n)$.

The complexity of the latter notion is clearly Σ^0_1 (*recursively enumerable*), and hence so is that of $\bar{A}$-invariance.

To establish a lower bound, we reduce the following *undecidable* notion effectively to a question of invariance:

α holds in all models of size greater than 1

(for predicate-logical sentences α).

Proposition: Let α be any predicate-logical formula, in vocabulary L.

Let Q be some unary predicate letter not occurring in L.

Then the following two assertions are equivalent:

1. α holds in all models with more than one individual,
2. $\forall x(\alpha \rightarrow Qx)$ is invariant with respect to permutations (i.e., = -automorphisms).

Proof: From 1 to 2: Let D be any model for $\forall x(\alpha \rightarrow Qx)$.

Case1: α holds in D.

Then so does $\forall xQx$. But then, any permutation will leave a model for $\forall xQx$, and hence for $\forall x(\alpha \rightarrow Qx)$.

Case 2: α fails in D.

Then there is only one individual. So, the only permutation is the identity, and invariance is automatical.

From 2 to 1: By Svenonius' Theorem, the assumed invariance implies the existence of some finite number of pure identity-formulas $\psi_1,\ldots,\psi_n$ such that

$$\forall x(\alpha \rightarrow Qx) \models \forall x(Qx \leftrightarrow \psi_1(x)) \vee \ldots \vee \forall x(Qx \leftrightarrow \psi_n(x)).$$

Hence, in particular, $\neg\alpha$ implies the latter disjunction. Now, pure identity-formulas, being permutation-invariant, can only define either the empty set or the whole universe, in any model. Therefore, we must have

$$\neg\alpha \models \forall xQx \vee \forall x\neg Qx.$$

Since Q does not occur in $\neg\alpha$, this rules out the possibility of $\neg\alpha$'s having models with more than one individual: otherwise Q might be chosen so as falsity both disjuncts while leaving $\neg\alpha$ true. Thus, condition 1 obtains for α.

It may be noted, to conclude, that similar tricks will establish non-decidability for a wide range of other natural semantic notions, including the monotonicity of Section 3. On the other hand, *in practice*, we can usually make do with a variety of explicit syntactic formats guaranteeing the presence of the desired semantic behaviour. This will be seen, e.g., in the discussion of 'positive occurrences' and monotonicity in Section 4.

4. DEFINABILITY

There is a standard logical language used for interpretation in the type-theoretic models employed up till 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.

But also, there is 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 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 an observation about arbitrary terms, which is easily proved by induction:

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

The converse does not hold generally; as 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 van Benthem 1987b):

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 a remarkable power of definition. One illustration concerns *functional completeness* of *Boolean operators*. As all beginners learn, the standard logical constants "not", "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'), etcetera? Perhaps surprisingly, the above few constants still suffice, in the following sense (van Benthem 1987a):

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* only have to supply 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 which sense could one say that the standard first-order quantifiers are 'expressively complete'? Here, already 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 van Benthem 1984b).

Next, we consider the effect of another general desideratum on logical constants, viz. the *context-neutrality* of Section 1.2. Here, it turned out convenient to shift to a relational perspective. Moreover, it will also be useful to change over to another type-theoretic language, having the usual quantifiers $\exists,\forall$ (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 $f^+ \mid D_e = f$.

The following gives at least a sufficient condition:

Proposition: Let ϕ define a unique object in every model. Let ϕ have every quantifier occurring *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 also obtain context-neutral items like the quantifier "most", one has to allow impredicative definitions ϕ too, referring to higher types; provided that they stay within the sub-hierarchy 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. Smirnova 1986). 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, etcetera) *themselves*?

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·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 an man"). Again, there occurs a 'canonical' transfer of meaning, as was obseved already by Montague:

$\lambda x_{((e,t),t)} \cdot \lambda y_e \cdot x(\lambda z_e \cdot BE_{(e,(e,t))}(z)(y))$

('y is a man' if 'a man' holds for 'being y').

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

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

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

In fact, *type changing* is a general phenomenon in natural language, which shows many systematic traits (see van Benthem (1986b, chapter 7), 1987b). We shall outline a few points, in as far as necessary for further application to logical constants.

Generally speaking, expressions occurring in one type a can also 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→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) ⇒ ((e,t),(e,t)):

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

- (e,(e,t)) ⇒ (((e,t),t),(e,t)) is quite analogous:

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

- (e,e·e) ⇒ ((e,(e,t)), (e,t)) becomes analogous again,

  if we rewrite to (e,e·e) ⇒ ((e·e,t),(e,t)).

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

(x,y) ⇒ ((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. Particularly frequent are, in addition to the above inference of Transitivity (often called 'Geach's Rule' in this context), so-called rules of Raising (also called 'Montague's Rule'):

x ⇒ ((x,y),y).

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

e ⇒ ((e,t),t).

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

Example:

proof tree

$$\cfrac{\cfrac{\cfrac{\cfrac{\overset{1}{e}\quad \overset{2}{(e,t)}}{t}\,\text{MP} \qquad (t,t)}{t}\,\text{MP}}{(e,t)}\,\text{C}}{((e,t),(e,t))}\,\text{C}$$

lambda terms

$$\cfrac{\cfrac{\cfrac{\cfrac{x_e \quad y_{(e,t)}}{y(x)} \quad NOT_{(t,t)}}{NOT(y(x))}}{\lambda x_e\cdot NOT(y(x))}}{\lambda y_{(e,t)}\cdot\lambda x_e\cdot NOT(y(x))}$$

Note how application encodes Modus Ponens, and lambda abstraction 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 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. Keenan & Faltz 1985).

The type changing perspective raises many new questions in connection with the analysis of logicality 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, and later results.)

Matters are already more complex with *monotonicity.* Some type changes preserve it: the earlier Geach Rule is an example. Others do not: the Montague Rule is a counterexample. 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 van Benthem 1987a.)

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 van Benthem (1986, chapter 7) and 1987a,b: 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

$\exists x_e \cdot y_{(e,t)}(x)$ (type ((e,t),t)))

$\exists x_{(e,t)} \cdot y_{((e,t),t)}(x)$ (type (((e,t),t),t))

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 1 for a possible semantic background to this move.

Appendix: Lambda Definability

From the point of view of the 'receiving type', the above raises the question how many items in it can be defined using parameters from other ones, via typed lambda calculus (or full type theory). We consider some examples here, to show the variety of semantic questions raised by the present perspective.

For instance, which items in ((e,t),t) are lambda-definable using parameters in the individual domain D_e ? This 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. In such an investigation, we may always restrict attention to terms in *lambda normal form*, containing no more 'redexes' of the form $(\lambda x \cdot \alpha)(\beta)$. Also, in normal forms, the types of all variables must be subtypes of the resulting type or of types of the parameters. With these restrictions, we see that the only candidates which qualify are the earlier Montague liftings:

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

The situation can be much more complex, however. For instance, van Benthem 1987c has a discussion of the *polyadic* quantifiers in type ((e,(e,t)),t), 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):

$$Q_1 \;\; TV \;\; Q_2.$$

On the analysis of the preceding Section, the resulting meaning (corresponding to a valid implicational derivation again) will be

$$\lambda x_{(e,(e,t))} \cdot Q1_{((e,t),t)}(\lambda y_e \cdot Q2_{((e,t),t)}(x(y))).$$

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

We add one illustration here, concerning the central type ((e,t),((e,t),t)) of *determiners*. As was observed before, at least homomorphic determiners could be derived from objects in type ((e,t),e). And in fact, there is a valid Geach transition of the form ((e,t),e) $\Rightarrow$ ((e,t),((e,t),t)). So, there is a general rule for defining determiners from choice functions

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

according to the earlier prescription. (This formula is reminiscent of the use of *Hilbert's ε-symbol*: $\lambda x \cdot \lambda y \cdot y($ '$\varepsilon(x)$' $)$.)

We can compare this outcome with the representation formula presented in Section 3.1. The latter would work out here to

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

or equivalently,

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

Thus, again, we see that there are different possibilities.

Instead of enumerating all possible lambda-derivations of determiners from choice functions, we consider a more general sort of question. Given a type, we may be interested in all those of its items which are *lambda-definable from parameters* in its proper *subtypes*. In general, many things can happen here: there may be many or few of those. For instance, one function x in (e,e) generates infinitely many others via its successive powers under composition. On the other hand, the type structure may be such that no genuine 'cycles' appear. This is in fact the case with determiners, as we shall see.

In this case, the relevant subtypes are as follows:

e, t, (e,t) and ((e,t),t).

[Incidentally, on an alternative analysis, we would have ((e,t)·(e,t),t), without the 'higher-order' subtype ((e,t),t).] Now, we can describe possible lambda normal forms for determiners with parameters from these types using 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 which does not start with a lambda. The point of this division will become clear from the rules for terms in normal form to be presented now:

$X_a \Rightarrow X_a^*$, all a

$X_a^* \Rightarrow C_a$, all a

$X_a^* \Rightarrow V_a$, all a

Next, rules for application or lambda abstraction depend on the actual types present: (recall the earlier-mentioned restrictions on normal forms!)

$X_{((e,t),((e,t),t))} \Rightarrow \lambda V_{(e,t)} \cdot X_{((e,t),t)}$

$X_{((e,t),t)} \Rightarrow \lambda V_{(e,t)} \cdot X_t$

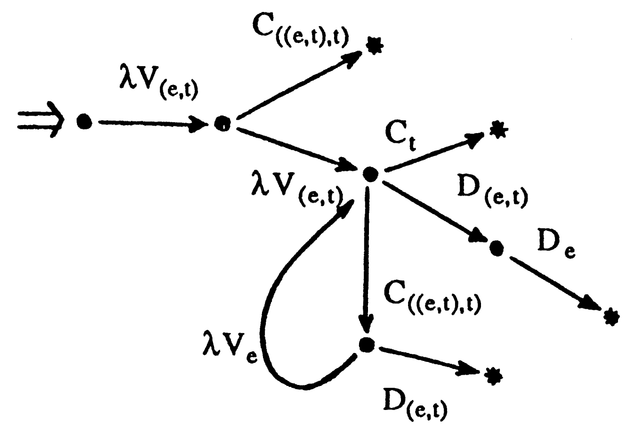
$X_t \Rightarrow X_{(e,t)}^*(X_e)$

$X_t \Rightarrow X_{((e,t),t)}^*(X_{e,t)})$

$X_{(e,t)} \Rightarrow \lambda V_e \cdot X_t$.

The description of possible readings is much facilitated, however, because we can make this grammar *regular*. This may be visualized in the following *finite state machine* for producing terms:

[Here 'D_a' stands for C_a, or V_a where applicable.]



This scheme produces determiner denotations of forms such as the following:

1. $\lambda x_{(e,t)} \cdot c_{((e,t),t)}$
2. $\lambda x_{(e,t)} \cdot \lambda y_{(e,t)} \cdot T$, $\quad \lambda x_{(e,t)} \cdot \lambda y_{(e,t)} \cdot \bot$ [in fact, this may be viewed as a subcase of 1]
3. $\lambda x_{(e,t)} \cdot \lambda y_{(e,t)} \cdot x(c_e)$, $\quad \lambda x_{(e,t)} \cdot \lambda y_{(e,t)} \cdot c_{((e,t),t)}(x)$
4. $\lambda x_{(e,t)} \cdot \lambda y_{(e,t)} \cdot c_{((e,t),t)}(\lambda z_e \cdot c'_{((e,t),t)}(\lambda u_e \cdot x(z)))$

Especially, the latter kind is 'iterative', in a modest way.

Nevertheless, it may be shown that all these forms are equivalent, in any model, to only a *finite* number of cases. We shall merely provide some evidence for this claim, by showing how one particular longer iteration may be reduced *uniformly*:

$$\lambda x_{(e,t)} \cdot \lambda y_{(e,t)} \cdot c_{((e,t),t)}(\lambda z_e \cdot c'_{((e,t),t)}(\lambda u_e \cdot c''_{((e,t),t)}(y(z)))).$$

The idea is to consider the crucial binding at the end (if there is one) and move it forward, using the fact that closed complex definitions for items in some relevant parameter type may be replaced by a single constant for a parameter of that type:

- replace the subterm beginning with λz_e by
  $\lambda z_e \cdot [y(z) \wedge c'(\lambda u \cdot c''(T))] \vee [\neg y(z) \wedge c'(\lambda u \cdot c''(\bot))]$,
  and then by
  $\lambda z_e \cdot [y(z) \wedge c_t] \vee [\neg y(z) \wedge c'_t]$.
  This again reduces to one of
  $\lambda z_e \cdot c_t$, $\lambda z \cdot y(z)$ or $\lambda z \cdot \neg y(z)$.
- but then, we can reduce the whole term still further, to one of the forms
  $\lambda x_{(e,t)} \cdot \lambda y_{(e,t)} \cdot c_t$ or $\lambda x_{(e,t)} \cdot \lambda y_{(e,t)} \cdot c_{((e,t),t)}(y)$,
  which both reduce to $\lambda x_{(e,t)} \cdot c_{((e,t),t)}$.

In fact, the general result is this: terms of the first three kinds always suffice.

Of the general questions suggested by this example, we mention only a few:

- When is the receiving type *not* filled by its items definable from lower ones? [Note that this was indeed the case with determiners: since none of the above terms can define a genuine interaction of the initial x and y arguments.] Can one prove general *Hierarchy Theorems* to such an effect ?
- When can a finite set of parameters only define a finite set of items in some higher type, even allowing all possible lambda terms for this purpose ?

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 shall 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* will 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 case 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 since Section 4 we know that these are not 'genuine' inhabitants of $((s,t),(s,t))$ (etcetera), 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 1, 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 van Benthem 1984a, 1985a. 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. van Benthem 1986c.) Tenses and temporal adverbs may be viewed as operators on propositions here which 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 which 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 flavour. 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" or "to-morrow".) Another interesting new aspect is the possible action of semantic *automata* on time lines (cf. Löbner 1987).

5.2 Dynamic Logic

A similar extension is possible to currently popular 'dynamic' logics, originally developed in the semantics of *programming languages*, but now also serving 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 merely viewed 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 Booleans, but also typically 'dynamic' operators, such as sequential *composition*.

One obvious question here is what would be a reasonable choice of basic logical items, given the broader options available now. 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 be 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, witness 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

$\lambda R.\lambda S.\lambda xy.\exists zu.Rzu \wedge \exists vw.Svw \wedge$

$<$ some Boolean condition on identities in x,y,z,u,v,w $>$.

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 preservation of propositional structure will lead to a collapse in options here. For instance, logical *homomorphisms* in this 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

$\lambda P.\lambda xy.Px$ and $\lambda P.\lambda xy.Py$.

(3) Eventually, as in Section 5.1, this setting too 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 van Benthem 1988).

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

$FF(R) = R$ or $FF(R) = F(R)$.

For binary operators, one has the usual commutativity, 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 ar 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 rather serve 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...). Then, our type structure will also involve propositions in type (s,t) after all. For some type-theoretic exploration of this richer structure, see van Benthem 1987b, 1988.

6. EPILOGUE

The stated purpose of this paper has been to analyse 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 a study of 'logical constants' (whatever these may be). Logic, in our view, is concerned with the study of *logical phemomena*: 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. van Benthem 1985b). With some adaptations to the Twentieth Century, this is still an appropriate banner to follow.

7. REFERENCES

1. J. van BENTHEM, 1983, Determiners and Logic,
Linguistics and Philosophy 6, 447-478.

2. J. van BENTHEM, 1984a, Foundations of Conditional Logic,
Journal of Philiosophical Logic 13, 303-349.

3. J. van BENTHEM, 1984b, Questions about Quantifiers,
Journal of Symbolic Logic 49, 443-466.

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

5. J. van BENTHEM, 1985b, The Variety of Consequence,
According to Bolzano,
Studia Logica 44, 389-403.

6. J. van BENTHEM, 1986a, A Linguistic Turn: New Directions in Logic,
in R. Marcus et al., eds., *Proceedings 7th International Congress of Logic, Methodology and Philosophy of Science. Salzburg* 1983,
North-Holland, Amsterdam, 205-240.

7. J.van BENTHEM, 1986b, *Essays in Logical Semantics*,
Reidel, Dordrecht, (Studies in Linguistics and Pholosophy, vol. 29).

8. J. van BENTHEM, 1986c, Tenses in Real Time,
Zeitschrift für mathematische Logik und Grundlagen der Mathematik 32,
61-72.

9. J. van BENTHEM, 1987a, Categorial Grammar and Lambda Calculus,
in D. Skordev, ed., *Druzhba Summer School in Applied Logic. 1986*,
Plenum Press, New York.

10. J. van BENTHEM, 1987b, Categorial Grammar and Type Theory,
report 87-07, Institute for Language, Logic and Information,
University of Amsterdam. [To appear in *Linguistic and Philosophy*.]

11. J. van BENTHEM, 1987c, Polyadic Quantifiers,
report 87-04, Institute for Language, Logic and Information,
University of Amsterdam. [To appear in *Linguistics and Philosophy*.]

12. J. van BENTHEM, 1987d, Towards a Computational Semantics,
in P. Gärdenfors, ed., *Generalized Quantifiers: Linguistic and Logical Approaches*,
Reidel, Dordrecht, (Studies in Linguistics and Philosophy, vol. 31),
31-71.

13. J. van BENTHEM, 1988, Semantic Parallels in Natural Languages and
Programming Languages,
Institute for Language, Logic and Information, University of Amsterdam.

14. J. van BENTHEM and K. DOETS, 1983, 'Higher-Order Logic', in D. Gabbay and F. Guenthner, eds., *Handbook of Philosophical Logic, vol.I*, Reidel, Dordrecht, 275-329.

15. E. KEENAN and L. FALTZ, 1985, *Boolean Semantics for Natural Language*, Reidel, Dordrecht, (Studies in Linguistics and Philosophy, vol. 23).

16. S. LOEBNER, 1987, Quantification as a Major Module of Natural Language Semantics, in J. Groenendijk, D. de Jongh and M. Stokhof, eds., *Studies in Discourse Representation Theory and the Theory of Generalized Quantifiers*, Foris, Dordrecht, (GRASS series, vol. 8), 53-85.

17. W.V.O. QUINE, 1966, Variables Explained Away, in *Selected Logic Papers*, Random House, New York.

18. E.D. SMIRNOVA, 1986, *Logical Semantics and the Philosophical Foundation of Logic*, Publishing House of Moscow State University.