QUANTIFIERS IN THE WORLD OF TYPES

Johan van Benthem

february 1991

Contents

to appear in

Jaap van der Does & Jan van Eyck, eds.
Quantification in the Netherlands
Institute for Language, Logic and Information
University of Amsterdam

1 From Special Properties to General Phenomena

Generalized quantifiers live in the special type ((e, t), ((e, t), t)) of linguistic determiners, being binary relations between properties of individuals according to their standard theory. Much of their behaviour has been studied in this special domain, including possible inferential 'syllogistic patterns' like Transitivity and Symmetry, as well as various 'denotational constraints' like Conservativity and Monotonicity. Standard expositions of the results achieved in this manner may be found in van Benthem 1986 or Westerståhl 1989. But quantification does not take place in isolation: generalized quantifiers function within wider languages, whether natural or artificial, which come with a full universe of all types that can be constructed over, at least, basic types e ('individuals') and t ('truth values'). Thus, the question arises how such inferential behaviour or such denotational constraints fare within this more general typed environment. For instance, inference is really a general process across an entire language, and the question becomes how quantifiers contribute systematically to this broader 'natural logic' (cf. Sánchez Valencia 1990). Thus, a move is made from specific conditions in one type to general phenomena across all types. As another illustration, the specific constraint of Conservativity for quantifiers by themselves becomes one local instance of a global phenomenon of what might be called 'domain restriction' across arbitrary types of linguistic expression. This general cross-categorial role is not confined to quantifiers: it also occurs with Boolean particles in natural language, as has been demonstrated and investigated at length in Keenan&Faltz 1985. Indeed, many denotational constraints for quantifiers involve some form of interaction with Boolean operators. The purpose of this Note is to investigate a few logical aspects of these general issues, showing that they give rise to some non-trivial questions concerning the range of our GQ-techniques in a wider setting.

2 Generalized Quantifiers in Lambda Calculus

A most convenient technical formalism for achieving type-theoretic generality in the description of natural or formal languages is that of the Typed Lambda Calculus. The latter system provides the 'logical glue' for constructing linguistic meanings from those for component expressions. This composition may be achieved in the Montagovian style, be it that recent research has come to insist on proper regard for denotational 'fine-structure', using various fragments of the full lambda calculus machinery to achieve 'expressive fine-tuning'. In principle, one could even ascend to a full Type Theory, manipulating explicit identities (cf. Gallin 1975), but this seems unnecessary in practice. Therefore, the technical question becomes how properties of quantifiers occurring in larger expressions interact with a lambda calculus machinery, in order to contribute to the inferential or denotational behaviour of the whole. Several examples of what happens here may be found in van Benthem 1991 (chapters 10, 11), which shows how a categorial analysis of linguistic expressions, with grammatical derivations corresponding to typed lambda terms via the Curry-Howard isomorphism ('formulas-as-types'), automatically imports much special quantificational behaviour into the meaning of general expressions. Here are two key cases.

Example. From Conservativity to General Domain Restriction.
Conservativity for single quantifiers has the form

$$Q\ A\ B \quad \text{iff} \quad Q\ A\ (B \cap A)\ ,$$

which restricts the argument of the predicate B to A-objects. Combining this with the mechanism of categorial derivation, appropriate argument restrictions may be computed for binary predicates in transitive sentences having multiple quantifiers:

$$Q1\ A\ \ TV\ \ Q2\ B \quad \text{iff} \quad Q1\ A\ \ (TV \cap (A{\times}B))\ \ Q2\ B.$$

But the procedure also covers, say, adverbial or prepositional phrases. Here is a categorial derivation of the types for the verb phrase "walk to every city" , whose common noun "city" restricts the individual argument of the preposition "to" :

```
                                                   "to"
            "walk"  e^1 _______________________(e, ((e, t), (e, t)))  MP
            (e, t) ____________________((e, t), (e, t))  MP
   e^2 ___________________(e, t)  MP               "every"               "city"
              t     COND, withdrawing 1          ((e, t), ((e, t), t))____________(e, t)
            (e, t)  restrict to CITY ____________________________________((e, t), t)  MP
                                   t     COND, withdrawing 2
                                 (e, t)
```

Its corresponding lambda term may be computed in Curry-Howard style as follows:

```
                         x_e ________________________TO_(e, ((e, t), (e, t)))
            WALK_(e, t) ___________________TO(x)
   y_e ____________________TO(x)(WALK)
             TO(x)(WALK)(y)
            λx_e• TO(x)(WALK)(y) ______________________EVERY CITY_((e, t), t)
                    EVERY CITY_((e, t), t)(λx_e• TO(x)(WALK)(y))
                  λy_e• (EVERY CITY_((e, t), t)(λx_e• TO(x)(WALK)(y))) .
```

Here, by ordinary Conservativity, the last three lines may replace their part

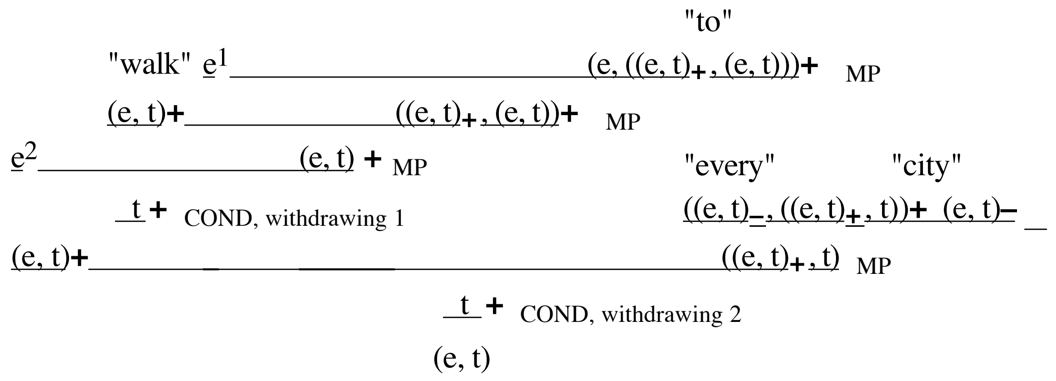
$$\lambda x_e\text{• } TO(x)(WALK)(y) \quad \text{by} \quad \lambda x_e\text{• } (TO(x)(WALK)(y) \wedge CITY(x))\ .$$

Along the lines of this 'restriction marking', a systematic calculus of argument restriction may be developed, showing how common nouns serve to restrict argument positions for individuals in event-type 'frames of predication' set up by verbal expressions.

Example. From Monotonicity to General Replacement.
Generalized quantifiers may display monotonicity in both their arguments, witness the case of ALL which is 'left downward' and 'right upward':

$A' \subseteq A \quad Q\ A\ B \quad B \subseteq B'$ imply $Q\ A'\ B'$.

But again, monotonicity also makes sense for general expressions. For instance, the above prepositional phrase "walk to every city" is upward in "walk" , and downward in "city" . Again, this may be computed directly from the above categorial analysis, either by means of 'monotonicity marking' on the grammatical derivation tree, or directly on its corresponding lambda term. The idea is that lexical items can have special monotonicity behaviour, which may be encoded in their types, and then gets passed on through categorial combination according to certain rules:



Here, monotonicity behaviour at the surface gets explained by calculating strings of **+** or **–** markers down to the root. E.g., "walk" has an unbroken string **+ + + + +** which explains explains its positive occurrence, whereas "city" has **– + +** . Essentially the same result may also be computed on the associated lambda term

$\lambda y_e \bullet (\text{EVERY CITY}_{((e,\ t),\ t)}(\lambda x_e \bullet \text{TO}(x)(\text{WALK})(y)))$,

using monotonicity behaviour of its parameters for their arguments together with two general 'rules of passage': in function applications, the function head is always positive, while in lambda abstractions, the body is always positive.

Thus, annotated categorial derivations, or adorned lambda terms, are a convenient medium for transferring the properties of quantifiers to a general linguistic environment. Again, there is often further fine-structure here, in the form of restrictions on the kind of lambda term that is actually needed. For instance, the main calculus for categorial combination in the literature is not the full Lambda Calculus, but rather something like its 'linear fragment' having only those terms in which each lambda operator binds *exactly one* free occurrence of its index variable. This is the semantic counterpart of the well-known 'Lambek Calculus' in the proof theory of Categorial Grammar. The complete picture is therefore rather one of a landscape of fragments of the full lambda machinery. Nevertheless, the latter system still provides a reasonable limit to semantic combination, and hence, some natural logical questions emerge about the power of this mechanism.

3 First-Order Predicate Logic as a Point of Departure

Quantificational patterns of inference or denotational constraints already arise in the core system of first-order predicate logic. The latter may be viewed as a small fragment of our full typed system, and some natural logical questions arising in the above are most easily demonstrated starting from this base. First, from a general categorial point of view, first-order predicate logic has variables of types e (individuals) as well as (e, t) (one-place predicates), $(e, (e, t))$ (two-place predicates), etcetera. Then, it also has a number of special constants, namely Booleans $\neg$ (type (t, t)) and $\wedge$, (type $(t, (t, t))$) as well as quantifiers $\forall$, $\exists$ in type $((e, t), t)$. All further combinatorics is supplied by general lambda calculus, witness a formula like

$$\forall x\, (Ax \to \exists y\, (By \wedge Rxy))$$

which amounts to

$$\forall\, (\lambda x\bullet \to (A(x))\, (\exists\, (\lambda y\bullet \wedge (B(y))\, (R(x)(y)))\,))\ .$$

Thus, predicate-logical formulas become terms of the truth value type t . [1] Moreover, statements of logical consequence from a sequence of premises $\varphi_1, ..., \varphi_n$ to a conclusion ψ may be rendered by the validity of the associated Boolean identity $\varphi_1 \wedge ... \wedge \varphi_n \wedge \psi$ = $\varphi_1 \wedge ... \wedge \varphi_n$: 'adding the conclusion has no effect after processing of the premises'. [2] Henceforth, we shall assume that the individual type e refers to some arbitrary non-empty base domain, whereas the type t refers to the standard domain of two truth values. [3]

Now, for a start, inferential behaviour in first-order logic is completely described by some well-known principles concerning these logical constants, combining basically the axioms of Boolean Algebra with some suitable quantifier postulates. Moreover, by Gödel's Completeness Theorem, the resulting set of proof-theoretic principles is 'complete', in that it derives all valid inferences. But of course, we can now do 'predicate logic' even within more complex terms of the lambda calculus, for instance, replacing equivalents at appropriate spots inside possibly higher-order syntactic environments. Thus, our first obvious question now becomes this:

Are the usual first-order principles, together with the standard rules of the typed lambda calculus, sufficient for deriving all valid inferences in the above logical constants for general typed lambda terms?

We shall make this somewhat more precise in Section 4 below, but the intent will be clear.

Next, the earlier denotational constraints also occur naturally inside first-order logic. For instance, Conservativity rests essentially on the following simple equivalence (see Sánchez Valencia 1990 for its historical origin with C. S. Peirce):

Ax □ (... ...) □ Ax □ (... Ax□ ...) ,

provided that x be free for substitution in this context.

In a sense, this is an inferential principle too, but at a higher level of generality than the postulates found in the above-mentioned axiomatizations of first-order logic, as it operates on more complex linguistic patterns. Another important example which already figures in predicate logic (again with a Peircean ancestry) is Monotonicity. A predicate-logical sentence □(P) is semantically *monotone* in the predicate P if its truth remains unaffected in any model by merely enlarging the extension of the predicate interpreting P . There is a number of interesting variants of this notion, restricting attention to specific occurrences of P , or to specific models for □, but the present one will do here. On the syntactic side, *positive* occurrence of a predicate P in a formula is defined as occurrence under an even number of negations. What is easy to see now is that a formula having only positive occurrences of a predicate P is monotone in it. In other words, the syntactic test of positive occurrence is 'sound' for semantic monotonicity. The converse is a much less obvious statement of 'completeness', being known as

'Lyndon's Theorem':

If a first-order sentence is monotone in the predicate P , then it is logically equivalent to one all of whose occurrences of P are positive.

And again, we have an obvious question:

Is there a similar 'preservation theorem' for monotonicity, supplying a complete syntactic format for this phenomenon in our general typed lambda calculus ?

In more practical terms, this amounts to the following concern. The above computations on categorial derivations gave us a means of spotting syntactically 'positive' positions, reflecting inferentially sensitive locations in a linguistic expression. But can we be sure that this mechanism in our natural logic really detects *all* such positions?

4 **Axiomatizing Inference**

The pure lambda calculus is a universal theory of function application and abstraction. Its typed variant arises when these functions are thought of as coming in hierarchical layers. Henceforth, we shall be thinking of 'standard models', consisting of function hierarchies over base domains including a non-empty set D_e of individual objects and a truth value domain D_t . Universal validity of identities M=N between terms of the typed lambda calculus may be explained semantically as their truth, under all variable assignments, in such structures. One fundamental result in the field is

'Friedman's Theorem':
Universal validity on standard models is axiomatized precisely by the Extensional Typed Lambda Calculus $□_□$, including the usual axioms for identity, lambda conversion and extensionality of functions.

This result does not extend from universal validity to the case of consequence from premises, and hence there is no easy way of adapting it to obtain complete axiomatizations for a lambda calculus with additional constants, such as the above ¬ , □ , □ with their usual theory. Nevertheless, it turns out possible to obtain an axiomatization for the logical constants of predicate logic in a full typed lambda calculus, using the following analysis in van Benthem 1991 (chapter 2).

The idea is to first introduce a new calculus $□_□^*$ of 'sequents' □•□ = □ , with □ any finite set of identities. 'Semantic validity' for these sequents says that " in all standard models, any assignment verifying all identities in □ also verifies □ = □ " . The resulting valid principles are obvious generalizations of those for $□_□$. For instance, 'Replacement of Identicals' returns in the axioms

□ = □•□(□) = □(□) □ = □•□(□) = □(□)

as well as the inference rule

if □•□ = □ (where x does not occur free in □) , *then* □•□x• □ = □x• □ .

Moreover, Extensionality assumes the following form:

if □•□(x) = □(x) (where x does not occur in □, □, □) , *then* □•□ = □ .

Finally, the usual structural rules hold for these sequents, such as Reflexivity, Monotonicity and Cut. A general completeness theorem for sequent-based lambda calculus may be proven at this stage, by modifying the usual arguments concerning term models. Its details are omitted here.

Next, an enriched calculus $□_□^*$B arises by adding all Boolean axioms as identities, together with the following rule of *Bivalence*

if □, $□_t$ = 1•□ = □ *and* □, $□_t$ = 0•□ = □ , *then* □•□ = □ .

Moreover, for technical reasons, *identity predicates* $=_{(e,\ (e,\ t))}$ are to be added for each ground domain e : requiring also principles of *Individual Identity*

$(□_e =_{(e, (e, t))} □_e) = 1$ • $□_e = □_e$ $\qquad$ $□_e = □_e$ • $(□_e =_{(e, (e, t))} □_e) = 1$. [4]

The system $□_□{}^*B$ is obviously sound for its intended semantic interpretation, over standard models with $D_t = \{0,1\}$. Indeed, a stronger result holds:

Proposition. An identity Ÿ•□ = □ is provable in $□_□{}^*B$ if and only if it is valid in all standard models having ground domain $D_t = \{0,1\}$.

Proof. The main steps in demonstrating the Proposition are as follows. Starting from a non-derivable sequent Ÿ•□ = □, one constructs some consistent set □ which does not derive □ = □, but which progressively decides all terms of the truth value type t (here is where the Bivalence Rule is crucial) and which makes sure that non-identical functions shall differ on at least one tuple of fresh arguments (this depends on Extensionality plus the new ground identity predicates). Then, a canonical term model may be constructed verifying □ while keeping □, □ distinct, whose domain D_t has just the two truth values 0, 1 . Now, the Friedman construction for a 'partial surjective homomorphism' from the standard model over the same ground domains still works, and it even maps the 'real' Boolean operators to their counterparts in the term model. Thus, the counterexample to □ = □ in the term model may be transferred to one in a standard model. ■

The crucial feature of Booleans underlying the preceding outcome is their 'first-orderness' in the hierarchy of types. And in fact, a similar analysis may be given for all of first-order predicate logic with a lambda calculus superstructure, in which the Booleans are supplemented with one more constant, namely the binary existential quantifier *some* in the earlier determiner type, interpreted as set overlap. Then, our final axiomatization is reached:

Proposition. The valid identities on standard models are axiomatized by $□_□{}^*B$ together with the following quantifier principles:

- *some* □□ = *some* □□ $\qquad$ Symmetry
  some □□ = *some* □(□□□) $\qquad$ Conservativity
  some □□ ≤ *some* □(□ □) $\qquad$ Monotonicity
  some 00 = 0 $\qquad$ Non-Triviality
- *some* □□ = 0 • □(x)□□(x) = 0 , for all variables x
  if □ • □(u)□□(u) = 0 , with u not occurring in □, □, □ ,
  then □ • *some* □□ = 0 .

Proof. We merely mention the crucial step in the proof for this result, which is otherwise similar to the one sketched before. The additional task this time is to make sure that the Friedman homomorphism h will map genuine set-theoretic overlap to the interpretation of the *some* functor in the term model. What is needed for this purpose is the introduction of 'witnesses' in the earlier construction of the set □ describing the term model for all statements of the form *some* □□ accepted at some stage. That this will work is ensured by the two final rules stated above. [5] The final check after the construction of h from the standard model to the term model runs as follows. If two sets X, Y of individuals are mapped onto term equivalence classes □~ , □~ , and the former share some element □~ , then we have □(□) = 1, □(□) = 1 derivable from □ , and hence *some* □□ = 1 as well (*some* □□ = 0 would derive that □(□)□□(□) = 0) : i.e., [[*some*]](□)(□) holds. Conversely, if the latter relation holds, then its witness will give a non-empty intersection for X, Y .

This result does not exhaust all questions of interest concerning general deduction in the lambda calculus. In particular, the analysis should be extended to cover non-standard quantifiers, such as "most" . Also, from a categorial viewpoint, various further operators are involved in first-order inference as it occurs in natural language that deserve separate investigation. One case are argument reducers of predicates, such as the reflexivizer "self" in type ((e, (e, t)), (e, t)) . This is the only logical 'Boolean homomorphism' in its type (cf. van Benthem 1986), which means, amongst others, that it commutes with negations and conjunctions. Would this suffice to characterize its inferential behaviour in the setting of a full lambda calculus?

5 Characterizing General Monotonicity

Our next concern is the earlier monotonicity. In a full lambda calculus, this phenomenon may be defined using a natural notion of 'Boolean inclusion' $Ó_a$ in all types a :

$Ó_e$ is $=_e$ $\qquad$ $Ó_t$ is $\leq_t$

in functional types (a, b) , $fÓ_{(a, b)}g$ iff $f(x)Ó_b\, g(x)$ for all x in D_a .

Now, a term $□_b$ with a free variable x_a may be called 'monotone' in x if its denotation [[□]] under any variable assignment depends monotonely on the object assigned to the variable x . Briefly,

' $uÓ_a v$ *only if* $[[□]]^x{}_u\ Ó_b\ [[□]]^x{}_v$ ' .

This generalizes the earlier case of predicate-logical formulas with individual predicates. Evidently, this 'upward' notion has a 'downward' dual too, reversing the relationship. On the syntactic side, there is also a natural notion of 'positive occurrence' for variables in lambda terms:

> x occurs positively in x itself, but in no other single variable y
> a positive occurrence of x in M is also one in an application $M(N)$
> a positive occurrence of x in M is also one in an abstraction $\lambda y\bullet M$.

This allows only one positive position, namely as the 'head variable' of a term. But in the presence of monotone parameters, such as the earlier constants $\neg$, $\wedge$, $\vee$, the situation changes. One now adds 'negative occurrences' as well, stating the obvious rules for argument positions of Boolean operators or quantifiers (upward monotone parameters preserve polarity in their arguments, downward monotone ones reverse it). Again, it is easy to prove 'soundness':

> If a variable x has only positive occurrences in a term of a lambda calculus with first-order logical constants as parameters, then that term is monotone in x .

As for a converse preservation theorem, the situation is subtle, and we only have some partial results so far.

What should be noted in any case is the following exception. The above inclusion relation reduces to identity on all 'non-Boolean types' whose final atomic type is e instead of t . Accordingly, each term is trivially monotone with respect to all its variables of non-Boolean type. Thus, in this case, no syntactic restrictions can be justified at all – and attention must be restricted to Boolean types, if anything of interest is to be discovered.

<u>First-Order Predicate Logic</u>

On the above view, even predicate-logical formulas have further positions where they might be monotone. For instance, the formula $\neg p$ is monotone in the connective occurrence $\neg$, as may be seen quite easily. Here, one refers to replacements in the ordering $Ó_{(t,\ t)}$ of unary Boolean operators, which is a diamond

$$\begin{array}{ccc} & \lambda x\bullet 1 & \\ \lambda x\bullet x & & \lambda x\bullet\neg x \\ & \lambda x\bullet 0 & \end{array}$$

Likewise, formulas may even be monotone with respect to certain occurrences of quantifiers in them, allowing substitution by 'more inclusive' quantifiers. What this requires, of course, is a more abstract view of predicate-logical formulas than the usual one, where propositional operators or quantifiers may now become subject to varying interpretation, so that the usual distinction between variable 'assignments' and more permanent 'interpretation functions' becomes fluid. By itself, such a change need not affect essential properties of the logic. For instance, a fundamental result like the Compactness Theorem still holds with 'Boolean variables'. [6] Here is a relevant observation.

Proposition. A predicate-logical formula is monotone with respect to a certain Boolean operator symbol if and only if it is equivalent to some formula having only positive occurrences of that symbol.

Proof. The argument rests on a simple trick, that will recur a number of times in what follows. If a formula □ is semantically monotone in the Boolean operator F (which we take to be unary, for convenience), then we have the following schematic equivalence:

□ □□F'Ó(t, t)F: [F'/F]□ .

Now, the latter formula may be defined explicitly via the disjunction of all four possible cases for F' :

□x•1Ó(t, t) F □ [□x•1/F]□ □x•xÓ(t, t) F □ [□x•x/F]□

□x•¬xÓ(t, t) F □ [□x•¬x/F]□ □x•0Ó(t, t) F □ [□x•0/F]□

Here, the substituted forms for □ on the right no longer contain occurrences of F at all. Moreover, the left-hand inclusions can be written entirely in positive terms for F, namely (respectively)

F(0)□F(1) F(1) F(0) 1 . [7]

But already for the case of quantifier symbols, the situation is less clear, and we merely have an open

Question. Does semantic monotonicity of a first-order formula □ with respect to some quantifier symbol Q imply definability of □ in a form which has only positive occurrences of Q in the above sense (that is, only directly under an even number of negations and an arbitrary number of monotone first-order operators such as □ , □ , □ , but not under itself) ?

<u>First-Order Predicate Logic with Generalized Quantifiers</u>

The preceding case really leads to a common kind of extension of first-order predicate logic, namely with additional generalized quantifiers interpreted via families of sets. (This restriction to unary quantifiers is merely for the sake of convenience.) Thus, the syntax now also allows operators of the form Qx• □(x) . One obvious question in this case is if the earlier Lyndon Theorem goes through for formulas with respect to occurrences of individual predicates. For extension by upward monotone quantifiers Q , this was shown by Makovsky & Tulipani: cf. the elaborate analysis in Doets 1991. For the case of arbitrary generalized quantifiers Q , even this simple question still seems open.

<u>Lambek Calculus and the Linear Lambda Fragment</u>
Now, let us pass to a general typed system. We start with a modest engine of categorial combination, namely the Lambek Calculus with its corresponding 'linear lambda terms'. Van Benthem 1991 (chapter 11) has the following result for the pure version of this system (without Boolean parameters):

Proposition. A linear term is monotone with respect to some variable of Boolean type if and only if it is equivalent to some term having that variable in head position.

Proof. The idea of the proof is as follows. First , one reduces the relevant term □ to its lambda normal form. Suppose that its occurrence of x still is not positive. Then, two variable assignments may be created progressively, differing at their x-values but otherwise the same, starting with two properly included Boolean values at the occurrence of x , and working outward across its successive 'contexts' in the term – whose effect is as follows:

> At each stage of the construction, some context in our term is under inspection, which is to receive two different values under the two assignments. If this context has a non-Boolean 'individual type', its 'leading variable' may be evaluated via extended assignments so as to make the whole context yield two distinct values in the appropriate individual type domain. If its leading variable has a 'Boolean type', then one reverses the Boolean inclusion once (working inside out), after which the resulting non-inclusion may be maintained across wider Boolean contexts encountered on the outward journey.

The single occurrence property of linear terms is crucial in maintaining the desired properties. At the end, the term will have non-included values for included values of x : quod non.

Now, this proof may be extended to cover the case of linear terms with Boolean parameters as well, by showing how to pass negations and conjunction in the above procedure.

Proposition. The preceding result also holds for linear terms with Boolean parameters, be it that the equivalent term may fail to contain the relevant variable at all.

Proof. The non-trivial direction runs as follows. The earlier reduction to lambda normal form may be performed first without using Boolean identities. (The latter might interfere in this process, witness the possible multiplication of redexes in a Boolean equation like $M = □(M)(M)$.) Afterwards, Boolean reductions may be made with impunity, as algebraic identities on terms of the non-functional type t do not introduce new redexes. Now, suppose that x still has an occurrence in the final form, which is not positive. Then, the procedure from the preceding proof still works, with this twist that Boolean (non-) inclusions must now be switched when passing negations. The only sensitive new case in maintaining the desired behaviour under the two assignments arises when a conjunctive context of the form $□(M)(N)$ is to be passed, with x occurring, say, in the conjunct M . Then it should be possible, by modifying the two assignments, to make N at least true: otherwise, the value becomes 0 in both cases, and the intended difference gets lost. But if N were unsatisfiable, $□(M)(N)$ would reduce to 0 by Boolean equivalence, and our original term would be equivalent to one without any occurrence of x after all. ■

Nevertheless, this result does not seem fully satisfactory, since Boolean conjunction in natural language seems to occur mainly in a 'coordinative' mode which results in lambda terms allowing multiple binding across conjuncts at the same level ("twists and shouts" naturally means $□x_e{\bullet}\ □(\text{TWIST}(x_e))\ (\text{SHOUT}(x_e))$) . [8] Therefore, we shall consider the full system of types and lambda terms after all.

Full Lambda Calculus

In the ordinary lambda calculus, apparent counter-examples exist to a preservation theorem for monotonicity, even at a simple Boolean level (cf. van Benthem 1991 (chapter 11)):

Example. Non-Positive Monotonicity.

The term $x_{(t,\ t)}\ (x_{(t,\ t)}\ (y_t))$ is semantically monotone in both its free variables, without having any definition by means of a pure term in which either of these variables occurs only positively. Nevertheless, there is a positive definition when Boolean parameters are allowed. Indeed, the following terms will both do (as may be checked by brute force):

$$((x_{(t,\ t)}(1)\quad x_{(t,\ t)}(0))\ □\ y)\qquad ((x_{(t,\ t)}(1)\ □\ x_{(t,\ t)}(0))\ □\ \neg y)$$

$$(y_t\ □\ x_{(t,\ t)}(x_{(t,\ t)}(1))\quad x_{(t,\ t)}(x_{(t,\ t)}(0))\ .$$

■

The preceding Example points at a more general result.

Proposition. For terms in a lambda calculus having only pure Boolean types (that is, involving t alone), semantic monotonicity implies positive definability.

Proof. An earlier trick may again be applied here. Suppose that a term τ_b is semantically monotone in its variable x_a. Then it is definable in the following schematic form:

$$£_{A \in D_a} \text{('AÓx'} \wedge [A/x]\tau) .$$

More precisely, an appropriate disjunction is to be taken here in the Boolean domain of type b. Now, this schema can be turned into a genuine definition, with x occurring only positively, because of the following facts:

(1) the enumeration of D_a can be made completely via closed terms of the Boolean lambda calculus. The reason is the Functional Completeness result in van Benthem 1991 (chapter 11) which shows that each pure Boolean object is explicitly definable in a lambda calculus having 0, 1, $\neg$, $\wedge$ added. In these closed terms, the variable x does not occur at all.

(2) inclusions between lambda terms of the form 'AÓx' are definable in a form which has only positive occurrences of x, again using a complete enumeration:

- A, x both of type t: $\neg A \quad x$
- otherwise, use a conjunction of assertions '$A(U)$Ó$_t x(U)$' running over all possible tuples of arguments U that take A, x to objects in type t.

Of course, available positive definitions may turn out much simpler in specific cases. [9]

The general case with individual types present seems much more difficult. In particular, the above kind of argument does not generalize, as it exploits the essential finiteness of the Boolean universe. [10] For the moment, we merely have a conjecture to offer concerning the pure lambda calculus with arbitrary types, but without special parameters. It is easy to see that the above pattern of analysis may be extended from Boolean inclusion to the case of arbitrary binary relations on ground domains lifted to more complex types componentwise.

Example. Natural Numbers.

Consider the binary relation $<$ on $D_e = \mathbf{N}$. For functions, we have, e.g., that $y^1 = \lambda x{\bullet}x < y^2 = \lambda x{\bullet}x{+}1$. Moreover, a lambda term like $\lambda z_{(e,\,e)}{\bullet}\, y_{(e,\,e)}(z_{(e,\,e)}(x_e))$ will be 'monotone' with respect to $<$ in an obvious sense.

Conjecture. General monotonicity with respect to arbitrary binary relations over the relevant ground domain for some typed variable implies positive definability.

Example, Continued. Non-Positivity.
Non-positive cases will typically not be semantically monotone with respect to variables occurring in irreducible argument positions. The term $y_{(e,\ e)}\ (z_{((e,\ e),\ e)}\ (y_e))$ does not respect the above binary relation $<$, as may be seen by taking functions $y^1 =$ □$x\bullet x$ $<$ $y^2 =$ □$x\bullet x+1$, and any functional z having $z(y^1) = 1$, $z(y^2) = 0$.

Theory of Types
Finally, it should be noted that the above question collapses in a full Theory of Types, allowing arbitrary identities in terms, or equivalently, 'higher-order' quantification over objects in arbitrary types. [11] In this system, we have the following reduction, again exemplifying an earlier trick:

A term □ is semantically monotone in the variable x_a if and only if
□ is definable by means of the formula □$U\bullet$ □v ([v/x]□(U) □ 'vÓx') ,
where 'U' is some tuple of variables reducing □ to ground level.
(Note that Boolean inclusion is explicitly definable in this type theory.) [12]

Nevertheless, certain more interesting versions of preservation results might be found even here, by placing restrictions on the complexity of proposed definitions, and looking for special cases where these might be available.

For a possible illustration, we return to the earlier open question of preservation for monotone quantifier occurrences in a first-order language with an added quantifier Q . The above trick would involve existential quantification over variables of the second-order type type $((e, t), t)$: but one can do better. An adaptation of a standard proof of monotonic preservation, based on elementary chains (cf. Chang & Keisler 1990), shows this [13]

Proposition. A formula □ is semantically monotone in the quantifier Q if and only if it is definable by means of some formula in which Q occurs only positively, allowing also existential quantification over predicates.

A simple formula in such an extended second-order formalism is 'non-emptiness': □$P_{(e,\ t)}\bullet\ Q(P)$, which is essentially outside of the original first-order language with Q . The earlier open question concerning this case then amounts to asking whether these existential quantifiers are really unavoidable when defining suitably simple Q-monotone formulas □(Q) .

6 Notes

1 This lambda calculus with constants for Boolean operators and first-order quantifiers should not be confused with another possible 'generalization' of first-order logic, having identities between lambda terms for its atomic statements, with Boolean operations over these, as well as quantifiers □x_a running over type domains D_a . This would amount to the earlier higher-order Type Theory. Note that quantifiers of the latter kind are not confined to any single category: they are 'transcategorial operators' just like the lambda abstractor itself.

2 In a common notation with Boolean inclusion, this would read as: '□$_1$ □ ... □ □$_n$ ≤ □ ' .

3 Actually, from the type-theoretic point of view, having arbitrary base domains of 'truth values', with appropriate operations over these, would be quite admissible too.

4 This calculus has been chosen for its meta-logical convenience, not for its practical utility. For instance, proving the Boolean 'conjunction rule' from two conclusions to their conjunction already requires a trick: " Suppose that □• □$_t$ = 1, □• □$_t$ = 1 . By Boolean Algebra and Substitution of Identicals, □$_t$ = 1, □$_t$ = 1 • □$_t$ □ 1 = 1 • (□$_t$ □ □$_t$) = 1 . Hence, by Cut and Contraction, □• (□$_t$ □ □$_t$) = 1 . "

5 The principles preceding them are merely some useful derived facts, allowing one to represent the binary quantifier *some* as a unary one in the usual manner.

6 The reason is that, since only finitely many possibilities exists for interpreting a Boolean operator, the usual ultraproduct proof for Compactness will still go through: one unique interpretation will be enforced in the ultrafilter.

7 Here is an illustration. The formula ¬(p□¬Fq) is monotone in F . By the recipe given, it must be equivalent, after some Boolean simplification, to the disjunction of F(0)□F(1), F(1)□¬(p□¬q), F(0)□¬(p□q), ¬p . And the latter may be reduced again to the obvious equivalent ¬p Fq .

8 There are many further subtleties to the process of 'argument management' in natural language, where, e.g., identifications must be lexicalized to a large extent. See the various discussions on this topic in van Benthem 1991, as well as the treatment of Boolean coordination in Sánchez Valencia 1990.

9 By way of example, the procedure may be applied to the earlier form xxy . What we get then for the case of y , is the disjunction of '0Óy'□xx0, '1Óy'□xx1 , which is indeed equivalent to the earlier form (xx1□y) xx0 . For the case of x , the result obtained reduces to that obtained via the earlier procedure for predicate logic, namely to something like (x1□x0) (x1□y) (x0□y) , which is again equivalent to the one found 'by hand'.

10 It will still work, of course, for purely Boolean objects even in a an {e, t}-type environment, witness the earlier case of truth-functional operators in predicate logic.

11 As was noted before, in a sense, the latter syncategorematic treatment of quantifiers is an alternative type-theoretic 'generalization' of first-order predicate logic, choosing the variable polymorphic type ((x, t), t) for unary generalized quantifiers.

12 This observation is related to the well-known 'trivialization' of Lyndon or Beth-type theorems in second-order logic.

13 The argument essentially treats the system as a two-sorted first-order one, with quantifiers standing for unary properties over individuals of the second 'predicate sort'.

7 References

van Benthem, J.

1986 *Essays in Logical Semantics*, Reidel, Dordrecht.

1991 *Language in Action. Categories, Lambdas and Dynamic Logic*, North-Holland, Amsterdam.

Chang, C. & H. Keisler

1990 *Model Theory*, North-Holland, Amsterdam.

Doets, H. C.

1991 'Interpolation and Preservation for Elementary Logic with an added Monotone Quantifier', Institute for Language, Logic and Information, University of Amsterdam.

Gallin, D.

1975 *Systems of Intensional and Higher-Order Modal Logic*, North-Holland, Amsterdam.

Keenan, E. & L. Faltz

1985 *Boolean Semantics for Natural Language*, Reidel, Dordrecht.

Sánchez Valencia, V.

1990 *Studies on Categorial Grammar and Natural Logic*, Dissertation, Institute for Language, Logic and Information, University of Amsterdam.

Westerståhl, D.

1989 'Quantifiers in Formal and Natural Languages', in D. Gabbay & F. Guenthner, eds., *Handbook of Philosophical Logic*, vol. IV, Reidel, Dordrecht, 1-131.