

Types of Relations

Jan van Eijck

CWI and ILLC, Amsterdam, Uil-OTS, Utrecht

jve@cwi.nl

NASSLLI, June 23, 2004

Abstract

Many arguments for flexible type assignment to syntactic categories have to do with the need to account for the various scopings resulting from the interaction of quantified DPs with other quantified DPs or with intensional or negated verb contexts.

We will define a type for arbitrary arity relations in polymorphic type theory. In terms of this, we develop the Boolean algebra of relations as far as needed for natural language semantics. The type for relations is flexible: it can do duty for the whole family of types $t, e \rightarrow t, e \rightarrow e \rightarrow t$, and so on.

If we use this flexible type for the interpretations of verbs, we can perform a ‘flexible lift’ on DP interpretations, so that DPs get interpreted as operations on verb meanings. This leads to an elegant implementation of Keenan’s proposal for polyadic quantification in natural language (‘Beyond the Frege Boundary’). It turns out that scope reorderings are particularly easy to implement in this framework.

The Frege Boundary

- Standard analysis of **Every lawyer cheated a firm**: this states a relation between the CN property of being a lawyer and the VP property of cheating firms, namely the relation of **inclusion**.
- Alternative analysis: look at the complex expression **Every lawyer ___ a firm**, and interpret this as a function that takes a relation as its argument (a denotation of a transitive verb, such as **cheated**, **accused**, **defended**) and produces a truth value.
- In many cases only the alternative analysis is available, e.g. **Every lawyer cheated a different firm**. This means that the relation C , when restricted to $L \times F$, is an injective function.

Reducible versus Irreducible Polyadic Quantifiers

Cases where the standard analysis is available involve [reducible](#) or [Fregean](#) functions, cases where only the alternative analysis is available are [irreducible](#).

- Van Benthem [[2](#)]
- Keenan [[6](#)]
- Ben-Shalom [[1](#)]
- Dekker [[3](#)]
- Van Eijck [[5](#)]

Examples of Irreducible (Non-Fregean) Quantifiers

- All boys fancied the same girl.
- John criticised Bill and noone else criticized anyone else.
- Every boy read a novel and every girl a play.
- Every student criticized everone but himself.
- The students criticized each other.
- Two detectives interviewed a total of twenty witnesses.
- The boys gave the same presents to the same girlfriends on the same occasions.

Quantifier Types

- A type $\langle 1 \rangle$ quantifier is a function of type $(e \rightarrow t) \rightarrow t$. Type $\langle 1 \rangle$ quantifiers are properties of sets of individuals.
- A type $\langle 2 \rangle$ quantifier is a function of type $(e \rightarrow e \rightarrow t) \rightarrow t$. Type $\langle 2 \rangle$ quantifiers are properties of binary relations between individuals,
- A type $\langle n \rangle$ quantifier is a function of type $(\underbrace{e \rightarrow \dots \rightarrow e}_{n \text{ times}} \rightarrow t) \rightarrow t$. Type $\langle n \rangle$ quantifiers are properties of n -ary relations between individuals.

Examples

Restricted universal quantifier, type $\langle 1 \rangle$.

$$\mathbf{forall}_A x \cdot \varphi(x) \quad :\Leftrightarrow \quad \forall x(Ax \Rightarrow \varphi(x))$$

Transitivity quantifier, type $\langle 2 \rangle$.

$$\mathbf{Tr} \, xy \cdot \varphi(x, y) \quad :\Leftrightarrow \quad \forall x \forall y (\varphi(x, y) \Rightarrow \forall z (\varphi(y, z) \Rightarrow \varphi(x, z)))$$

Injectivity quantifier, type $\langle 2 \rangle$.

$$\mathbf{Inj} \, xy \cdot \varphi(x, y) \quad :\Leftrightarrow \quad \forall x \forall y (x \neq y \Rightarrow \exists u \exists v (\varphi(x, u) \wedge \varphi(y, v) \wedge u \neq v)).$$

Set injectivity quantifier, type $\langle 2 \rangle$.

$$\mathbf{INJ} \, xy \cdot \varphi(x, y) \quad :\Leftrightarrow \quad \forall x \forall y (x \neq y \Rightarrow \lambda u \cdot \varphi(x, u) \neq \lambda v \cdot \varphi(y, v)).$$

The set injectivity quantifier captures the meaning of **Different students gave different answers.**

Quantifiers as Relation Reducers

- A type $\langle 1 \rangle$ quantifier corresponds to a function that maps $m+1$ -ary relations to m -ary relations.
- A type $\langle n \rangle$ quantifier corresponds to a function that maps $m+n$ -ary relations to m -ary relations.
- If $F :: \langle n \rangle$ and R is an $m+n$ -ary relation, then

$$F(R) = \{ \langle d_1, \dots, d_m \rangle \mid F \{ \langle d_{m+1}, \dots, d_{m+n} \rangle \mid \langle d_1, \dots, d_m, d_{m+1}, \dots, d_{m+n} \rangle \in R \} = 1 \}.$$

- Example:

$$(\mathbf{some} \ B)(R) = \{ d \mid B \cap \{ d' \mid dRd' \} \neq \emptyset \}.$$

Quantifier Reducibility

A type $\langle 2 \rangle$ function F is reducible if there are type $\langle 1 \rangle$ functions f, g with $F = f \circ g$, i.e., $F = \lambda R. f(\lambda x. g(\lambda y. Rxy))$.

Examples

Let $F = \lambda R. \forall x (Ax \rightarrow \exists y (By \wedge Rxy))$. Then $F :: \langle 2 \rangle$.

F is reducible: $F = \lambda R. (\mathbf{all} A) \circ (\mathbf{some} B)R$.

Let $F = \lambda R. \exists x (Ax \wedge \forall y (By \rightarrow Rxy))$. Then $F :: \langle 2 \rangle$.

F is reducible: $F = \lambda R. (\mathbf{some} A) \circ (\mathbf{all} B)R$.

Now take $F = \lambda R. \mathbf{Inj} xy \cdot R(x, y)$.

How do we find out whether F is reducible? The fact that we cannot easily find f, g with $F = f \circ g$ proves nothing ...

General Programme of Polyadic Quantification

- analyse a sentence as a combination of a list of DPs (corresponding to subject, direct object, indirect object, plus a list of argument PPs) and a VP.
- interpret a list consisting of n DPs as a polyadic quantifier of type $\langle n \rangle$.
- interpret a VP as an n -ary relation.
- interpret a sentence by applying its DP list interpretation to its VP interpretation.
- interpret scope reordering as a **parallel** operation on lists of quantifiers (all in reduced form) and VP interpretations.

Characterizing Irreducibility

Keenan [6] shows: if two reducible $\langle 2 \rangle$ functions F, G behave the same on product relations then $F = G$. This can be used to show irreducibility of type $\langle 2 \rangle$ functions.

1 *Different students answered different questions.*

For this to make sense, assume that there are at least two students.

The compound **Different students** ___ **different questions** is interpreted as the type $\langle 2 \rangle$ function expressing that its argument relation R satisfies the property that all the aR , with a ranging over students, are different.

The sentence asserts that there is a one-to-one correspondence between students and sets of questions they asked.

Showing Irreducibility

Let F be the type $\langle 2 \rangle$ function that interprets **different students** ___ **different answers**. Pick a universe containing at least three students s_1, s_2, s_3 and at least two questions q_1, q_2 .

Let $A \times B$ be a product relation, i.e., a relation that links every object in A to every object in B . If there are two students not in A , then they bear $A \times B$ to the same questions, namely, no questions.

If there are two students in A , then the questions they bear $A \times B$ to are again the same, namely $B \cap Q$. Again, $F(A \times B) = 0$.

Let $\mathbf{0}$ be the type $\langle 1 \rangle$ function that is false for any argument. Then, by the above, $F(A \times B) = \mathbf{0} \circ \mathbf{0}(A \times B)$ for any product relation $A \times B$.

But obviously, F is different from the composition $\mathbf{0} \circ \mathbf{0}$, for F is true of $\{\langle s_1, q_1 \rangle, \langle s_2, q_2 \rangle\}$, and $\mathbf{0} \circ \mathbf{0}$ is not. Thus, by Keenan's theorem, F is not reducible.

Characterizing Reducible Functions

Fact 1 (Keenan) *Let f be a positive function of type $\langle 1 \rangle$ and let $P, Q \subseteq E$. Then:*

$$f(P \times Q) = \begin{cases} P & \text{if } f(Q) = 1 \\ \emptyset & \text{otherwise} \end{cases}$$

Proof. Suppose $P = \emptyset$. Then $P \times Q = \emptyset$ and $f(P \times Q) = \emptyset$. From this the fact follows.

Suppose $Q = \emptyset$. Then again $P \times Q = \emptyset$ and $f(P \times Q) = \emptyset$. Positivity of f implies $f(Q) = 0$, so the fact follows.

Suppose $P \neq \emptyset, Q \neq \emptyset$. Then $f(P \times Q) = \{d \in P \mid f(Q) = 1\}$, and from this the fact follows. ■

Corollary: $g \circ f(P \times Q) = 1$ iff $g(P) = 1 \wedge f(Q) = 1$.

Theorem 2 (Keenan) *If $F, G :: \langle 2 \rangle$ are reducible functions then $F = G$ iff F and G act the same on product relations $P \times Q$.*

Proof: If $F = G$ then their behaviour on products is the same.

For the other direction, first assume F, G both positive, and suppose F, G have the same behaviour on products. Because of reducibility there are positive f_1, f_2, g_1, g_2 with $F = f_1 \circ f_2$ and $G = g_1 \circ g_2$. Then by fact 1, $f_1 = g_1$ and $f_2 = g_2$. Thus $F = f_1 \circ f_2 = g_1 \circ g_2 = G$.

Now assume F, G negative. Then, because of reducibility there are f_1, f_2, g_1, g_2 , with f_1, g_1 negative, f_2, g_2 positive, $F = f_1 \circ f_2$ and $G = g_1 \circ g_2$. Clearly, if $f_2 = g_2$, then by Fact 1, $f_1 = g_1$, and $F = f_1 \circ f_2 = g_1 \circ g_2 = G$.

On the other hand, if $\exists Q : 0 = f_2(Q) \neq g_2(Q) = 1$, then for any P , $f_1 \circ f_2(P \times Q) = f_1(\emptyset) = 1 = g_1 \circ g_2(P \times Q)$. It follows that $f_1 = g_1 = \mathbf{1}$, and $F = G = \lambda R. \top$. ■

Characterizing the Reducible Functions

Keenan does give an indirect characterisation of the reducible functions. Dekker [3] proposes an improvement, in terms of a notion of ‘invariance for sets in products’. We will propose a direct criterion.

Definition 3 (Reduct) *The reduct F^\bullet of a positive type $\langle n \rangle$ function F is defined as $F^\bullet = f_1 \circ \dots \circ f_n$ with f_i given by:*

$$\begin{aligned} f_i(\emptyset) &:= 0 \\ f_i(Q \neq \emptyset) = 1 &:\Leftrightarrow \exists Q_1, \dots, Q_{i-1}, Q_{i+1}, \dots, Q_n \subseteq E, \\ &F(Q_1 \times \dots \times Q_{i-1} \times Q \times Q_{i+1} \times \dots \times Q_n) = 1 \end{aligned}$$

Theorem 4 *For all positive type $\langle n \rangle$ functions F :*

$$F = F^\bullet \text{ iff } F \text{ is reducible.}$$

Normal Forms for Characteristic Functions on n -ary Relations

- The fact that a function is not reducible to a composition of type $\langle 1 \rangle$ functions does not mean that it is in its simplest possible form.
- A function F of type $\langle 3 \rangle$ may be reducible to a pair $f \circ G$, with $f :: \langle 1 \rangle$ and $G :: \langle 2 \rangle$.
- Example: **Every boy gave different presents to different girlfriends**
This can be analysed as a type $\langle 3 \rangle$ function F , but in this case F is reducible to **forall** _{B} \circ **inj**, with **forall** _{B} $:: \langle 1 \rangle$ and **inj** $:: \langle 2 \rangle$.
- Fact: if a function $F :: \langle 3 \rangle$ is both $(1, 2)$ reducible and $(2, 1)$ reducible, then it is fully reducible.
- This fact can be generalized to a **Diamond Property** for reduction.

Theorem 5 (Diamond Property) *If $\mathbf{F} = F \circ G = K \circ M$ with $F :: \langle m \rangle$ and $G :: \langle n \rangle$, $m < m'$, $K :: \langle m' \rangle$, $M :: \langle m + n - m' \rangle$, all of \mathbf{F}, F, G, K, M positive, then there is a positive function $H :: \langle m' - m \rangle$ such that $\mathbf{F} = F \circ H \circ M$.*

$$\begin{array}{ccc}
 \mathbf{F} & \longrightarrow & F \circ G \\
 \downarrow & & \downarrow \\
 K \circ M & \longrightarrow & F \circ H \circ M
 \end{array}$$

Theorem 6 (Normal Form) *Every positive $\mathbf{F} :: \langle n \rangle$ is uniquely representable as*

$$\mathbf{F} = F_1 \circ \dots \circ F_k,$$

with F_i positive and irreducible for all $i : 1 \leq i \leq k$. Moreover, there exists an algorithm for finding this normal form $NF(\mathbf{F})$.

A Polymorphic Type for Relations

If we use r for the type of (arbitrary arity) relations, then r is shorthand for the types

- t (from nullary relations to truth values),
- $(e \rightarrow t) \rightarrow t$ (from unary relations to truth values),
- $(e \rightarrow e \rightarrow t) \rightarrow t$ (from binary relations to truth values),
- and so on.

All quantifier functions now have the polymorphic type $r \rightarrow r$.

Datatype for Relations

```
data Rel a = R1 Bool | R2 (a -> Rel a)

instance (Enum a, Bounded a, Show a) => Show (Rel a)
  where show rel = show (rel2lists rel)

rel2lists :: (Enum a, Bounded a) => Rel a -> [[a]]
rel2lists (R1 b) = [[] | b ]
rel2lists (R2 f) =
  [ x: tuple | x      <- [minBound..maxBound],
              tuple <- rel2lists (f x)      ]
```

```
apply :: Rel a -> a -> Rel a
apply (R1 _) _ = error "no argument position left"
apply (R2 f) x = f x
```

```
r2b :: Rel a -> Bool
r2b (R1 b) = b
r2b (R2 _) = error "wrong arity"
```

```
r2prop :: Rel a -> (a -> Bool)
r2prop (R2 f) = \ x -> r2b (f x)
```

Arity

We need a constraint on **a**, for we should be able to locate an instance of the type. If the type is bounded, then **minBound** gives us the desired instance.

```
arity :: Bounded a => Rel a -> Int
arity (R1 _) = 0
arity (R2 f) = arity (f minBound) + 1
```

Boolean Algebras of Relations

```
opREL :: (Bool -> Bool -> Bool)
        -> Rel a -> Rel a -> Rel a
opREL op (R1 b) (R1 c) = R1 (op b c)
opREL op (R2 f) (R2 g) =
    R2 (\ x -> opREL op (f x) (g x))
opREL op _ _ = error "different arities"
```

```
conjR, disjR :: Rel a -> Rel a -> Rel a
conjR = opREL (&&)
disjR = opREL (||)
```

Negation/Complement

```
negR :: Rel a -> Rel a
negR (R1 b) = R1 (not b)
negR (R2 f) = R2 (\ x -> negR (f x))
```

Boolean operations on relations needed:

- to hurt but not to kill $\mapsto H \sqcap \overline{K}$,
- to break or drop $\mapsto B \sqcup D$,
- to give or sell $\mapsto G \sqcup S$.

`negR` is a map from relations to relations (same type $r \rightarrow r$ as quantifiers).

Lifting the Types of DP Interpretations

Consider the problem of applying a DP interpretation function, i.e., a function f of type

$(\mathbf{a} \rightarrow \mathbf{Bool}) \rightarrow \mathbf{Bool}$

not to a property (type $\mathbf{a} \rightarrow \mathbf{Bool}$), but to a relation with arity > 1 .

General shape of a relation of arity $n + 1$ (with $r :: \mathbf{Bool}$):

$$\lambda x_1 \cdots \lambda x_n \lambda y. r \tag{1}$$

If the argument relation has the form $\lambda y. r$, then property function f can be applied to it, yielding a boolean $f(\lambda y. r)$.

If the argument relation has the form $\lambda x_1 \cdots \lambda x_n \lambda y. r$, the new relation is given by (2).

$$\lambda x_1 \cdots \lambda x_n. f(\lambda y. r) \tag{2}$$

Let (3) be the result of reducing f with $\lambda x_1 \cdots \lambda x_n \lambda y.r$.

$$R f (\lambda x_1 \cdots \lambda x_n \lambda y.r). \quad (3)$$

Definition of operation R by recursion:

$$\begin{aligned} R f (\lambda y.r) &:= f(\lambda y.r) \\ R f (\lambda x_1 \cdots \lambda x_n \lambda y.r) &:= \lambda x. R f ((\lambda x_1 \cdots \lambda x_n \lambda y.r)(x)). \end{aligned}$$

```
reduce :: Bounded a =>
  ((a -> Bool) -> Bool) -> Rel a -> Rel a
reduce f r
  | arity r == 1 = R1 (f (r2prop r))
  | otherwise    = R2 (\ x -> (reduce f (apply r x)))
```

The Lifting Function

The function for lifting the types of DP interpretations is given by:

$$\text{lift} = \lambda f \lambda r. R f r.$$

```
lift :: Bounded a =>
      ((a -> Bool) -> Bool) -> Rel a -> Rel a
lift = \ f r -> reduce f r
```

If \mathbf{r} has arity $n + 1$, then $\text{lift } \mathbf{f } \mathbf{r}$ has arity n . Thus, lift is indeed a reducer of $n + 1$ -arity relations to n -ary relations.

Scope Reversal of Quantifiers

Generalized negation and generalized quantification are operators of type `Rel a -> Rel a`.

```
type Op a = Rel a -> Rel a
```

A difference between these two kinds of operators is that generalized negation does not reduce the arity of its argument relation, but generalized quantification does.

The scope swap of two quantifiers has the following type:

```
qscopeReversal :: Op a -> Op a -> Op a
```

For swapping the scopes of Q_1 and Q_2 it is not enough to switch from $Q_1 \circ Q_2$ to $Q_2 \circ Q_1$, The problem is that this switches the argument positions that the operators work on.

To correct this, we have to also swap the relevant argument places.

First swap the two outermost argument places, then apply Q_1 , and finally apply Q_2 :

$$Q_2 \cdot Q_1 \cdot S,$$

where S is a swap operation defined by:

$$S := \lambda r \lambda x \lambda y. (ry)x.$$

```
swap :: Op a
swap = \ r ->
      R2 (\ x ->
          R2 (\ y ->
              (apply (apply r y) x)))
```

```
qscopeReversal :: Op a -> Op a -> Op a
qscopeReversal = \ q1 q2 -> q2 . q1 . swap
```

For a scope reversal of a quantifier and a negation no argument swapping is needed, as the negation operation is not an arity reducer.

Quantifier Scoping

‘Clean’ version of Montague’s [quantifying in](#) and Cooper’s [quantifier storage](#). Cleanup made possible by the fact that the relational approach to quantifier scoping avoids the use of free variables.

To generate all scopings of Q_1, \dots, Q_n with respect to an n -ary relation R , proceed as follows:

- Permute the list of quantifiers Q_1, \dots, Q_n and the list of argument places $1, \dots, n$ of the relation R in parallel.
- For each permutation p , apply $Q_{p(1)}, \dots, Q_{p(n)}$, to the p -permutation of R .

All permutations of a finite list:

```
perms :: [a] -> [[a]]
perms [] = [[]]
perms (x:xs) = concat (map (insrt x) (perms xs))
  where
    insrt :: a -> [a] -> [[a]]
    insrt x [] = [[x]]
    insrt x (y:ys) = (x:y:ys) : map (y:) (insrt x ys)
```

Permuting two lists in parallel:

```
permsInPar :: [a] -> [b] -> [[(a,b)]]
permsInPar xs ys = map unzip (perms (zip xs ys))
```

Use `permsInPar` to generate lists of triples consisting of

- a list indicating the permutation of surface scope order,
- the relation under that permutation,
- the list of operators, again under the same permutation.

```
permRelQs :: (Eq a, Bounded a, Enum a) =>
    Rel a -> [Op a] -> [[Int], Rel a, [Op a]]
permRelQs rel qs =
    [ (perm, extract perm rel, pqs) |
      (perm,pqs) <- permsInPar [1..(length qs)] qs ]
```

```
extract :: (Enum a, Bounded a, Eq a) =>
          [Int] -> Rel a -> Rel a
extract is rel = extr is (rel2lists rel)
  where
    extr []    ess = R1 (not (null ess))
    extr (i:js) ess =
      R2 (\ x ->
          extr js (filter (\es -> role i es == x) ess))
    role n es = es !! (n-1)
```

```
HRITT> map allScopings
      (parse "Somebody respected every woman.")
[[([1,2],False),([2,1],True)]]
```

```
HRITT> map allScopings
      (parse "Nobody sold a thing to every woman.")
[[([1,2,3],True),([2,1,3],True),([2,3,1],True),
  ([1,3,2],True),([3,1,2],False),([3,2,1],True)]]
```

Finishing Touches

1. Syntactic constraints on scope reversals: see the classic Kroch [7].
2. Scoping with quantifiers of higher types: permutation on the relation should take numbers of argument places into account.
3. Scoping with respect to intensional contexts:

John is looking for a girlfriend.

Need for relations over types e and $w \rightarrow ((e \rightarrow t) \rightarrow t)$.

```
data R1 a b = R11 Bool
           | R12 (a -> R1 a b)
           | R13 (b -> R1 a b)
```

Website of Computational Semantics Textbook

Please visit:

<http://www.cwi.nl/~jve/cs>

- Draft of [Computational Semantics and Type Theory](#)
- Haskell programs and modules

Comments very welcome!

Modelling with Haskell Textbook

More on using Haskell as a tool for formal modelling in [4].

Acknowledgement

Thanks for comments to: Paul Dekker, Chris Fox, Shalom Lappin, Ed Keenan,

The trip to NASSLLI was sponsored by the NASSLLI organisation, ILLC and CWI.

References

- [1] Dorit Ben-Shalom. A tree characterization of generalized quantifier reducibility. In M. Kanazawa and C.J. Pi nón, editors, *Dynamics, Polarity and Quantification*, number 48 in CSLI Lecture Notes, pages 147–171. CSLI, 1994.
- [2] J. van Benthem. Polyadic quantifiers. *Linguistics and Philosophy*, 12(4):437–464, 1989.
- [3] P. Dekker. Meanwhile, within the Frege boundary. *Linguistics and Philosophy*, 26:547–556, 2003.
- [4] K. Doets and J. van Eijck. *The Haskell Road to Logic, Maths and Programming*, volume 4 of *Texts in Computing*. King’s College Publications, London, 2004.

- [5] J. van Eijck. Normal forms for characteristic functions on n -ary relations. Manuscript, CWI and ILLC, Amsterdam, 2004.
- [6] E. Keenan. Beyond the Frege boundary. *Linguistics and Philosophy*, 15(2):199–221, 1992.
- [7] A.S. Kroch. *The Semantics of Scope in English*. PhD thesis, MIT, 1974.