Naturalness of Polymorphism

Peter J. de Bruin

CS 8916

Computing Science Notes

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science of Groningen University.

Please note that this report has been included as appendix D in my Ph.D. Thesis “Inductive Types in Constructive Languages”, Groningen 1995, with a slight change of notation, two proofs greatly simplified by means of addition applications of naturality, and one additional theorem.

The correspondence between this report and appendix D of the thesis is as follows.

CS 8016 “Naturalness of Polymorphism”	**Inductive Types in Constructive Language, appendix D “Naturality of Polymorphism”**
1 Introduction	D.1 Introduction
2 Our language	D.2 Polymorphic typed lambda calculus
3 Naturalness on binary relations	D.3 Turning type constructors into relation constructors
– (no heading)	D.4 Naturality of expressions
Naturalness theorem	Theorem D.1 (Naturality)
Theorem (on page 6) – proven by initiality of the type constructor	Replaced by the more general Theorem D.2 – proven by two applications of naturality
(not present)	Theorem D.3 (Any type expression that preserves arrows and identity, preserves composition as well)
4 Applications	D.5 Applications
5 Dinaturalness	D.6 Dinatural transformations
Theorem (page 8) – to be proven by induction on the type expression, and initiality of the inductive type constructor	Reformulated as Theorem D.4 (dinaturality) – proven by two simple applications of naturality
Corollary (page 9)	Corollary D.5
6 Second-order languages	D.7 Second-order languages
Peano's induction axiom	Theorem D.6
7 Overloaded operators	D.8 Overloaded operators
8 Application to inductive definitions	Moved to section 6.3 of the thesis; the equivalence result is reformulated as theorem 6.4

An early version of the new theorems D.2 and D.3 is included at the end of the current electronic distribution., page 14A.

Dr. Peter J. de Bruin, May 2009

Technical Report CS8916, Dept. of Mathematics and Computer Science, University of Groningen 1989

Naturalness of Polymorphism

Peter J. de Bruin*

September 1, 1989

Abstract

From the type of a polymorphic object we derive a general uniformity theorem that the object must satisfy. We use a scheme for arbitrary recursive type-constructors. Special cases concern natural and dinatural transformations, promotion and induction theorems. We employ the theorem to prove equivalence between different recursion-operators.

1 Introduction

Objects of a parametric polymorphic type in a polymorphic functional language like Miranda or Martin-Löf-like systems enjoy the property that instantiations to different types must have a similar behaviour. Specifically, We use greek letters as type-variables, and $T[\alpha]$ stands for a type-expression possibly containing free occurrences of α. Stating $t : T[\alpha]$ means that term t has polymorphic type $T[\alpha]$ (in some implicit context), and hence $t : T[A]$ for any particular type A. Such instances are sometimes written with a subscript for clarity: $t_A : T[A]$. Some type-expressions $T[\alpha]$ are *functorial* in α. I.e., for any function $p : A \to B$ there is a given function $T[p] : T[A] \to T[B]$, such that $T[\mathrm{I}_A] = \mathrm{I}_{T[A]}$ and $T[p\,;q] = T[p]\,;T[q]$, where I is the identity function and (;) denotes forward function-composition. It has often been observed (e.g. [1]) that polymorphic functions $f : U[\alpha] \to V[\alpha]$, where U, V are functors, must be natural transformations:

$$\text{For any } p : A \to A', \text{ one has } f_A\,;V[p] = U[p]\,;f_{A'} : U[A] \to V[A'] \tag{1}$$

Illustration:

$$\begin{array}{ccc} U[A] & \xrightarrow{f_A} & V[A] \\ \Big\downarrow{\scriptstyle U[p]} & & \Big\downarrow{\scriptstyle V[p]} \\ U[A'] & \xrightarrow{f_{A'}} & V[A'] \end{array}$$

For example, any function $rev : \alpha* \to \alpha*$, where $\alpha*$ is the type of lists over α, must satisfy for $p : A \to A'$:

$$rev\,;p* = p*\,;rev$$

Unfortunately, type-expression $(U[\alpha] \to V[\alpha])$ is generally not functorial itself (because $(\to)$ is *contravariant* in its first argument). One can extend statement (1) to *dinatural transformations* in the sense of Mac Lane [2] pp. 214-218, but such a statement is still not provable by induction on the type-correctness of f. In this paper we will develop a generalization to arbitrary types that is provable for all lambda-definable objects of some type. The generalization

*University of Groningen, Dep. of Computing Science, P.O.Box 800, 9700 AV Groningen, The Netherlands. E-mail: hp4nl!guvaxin!peterb

allows one to derive many properties of polymorphic objects from their type alone, which are conventionally proven by induction using their definition, for example promotion-theorems on functions like:

$$foldr : (\alpha \times \beta \to \beta) \times \beta \to (\alpha* \to \beta)$$

This promotion-theorem says for $p : A \to A'$, $q : B \to B'$, $c : A \times B \to B$ and $c' : A' \times B' \to B'$, if

$$c\,;q = (p \times q)\,;c' : A \times B \to B'$$

then:

$$foldr\langle c, b\rangle\,;q = p*\,;foldr\langle c', qb\rangle : A* \to B'$$

If one identifies natural numbers with objects of polymorphic type $(\alpha \to \alpha) \to (\alpha \to \alpha)$, one can even derive Peano's induction-axiom.

The problem was suggested to us by J.G. Hughes of Glasgow University who mentioned property (1) during a lecture in Groningen in October 1988. After sending him an earlier version of our notes we received a draft paper [3] from Philip Wadler (also of Glasgow University) who derives the same main theorem (but in a more formal setting) as we do, and gives many promotion-like applications. His paper gave us the references to the literature.

The essential theorem is in fact Reynolds' abstraction theorem [4]. (His paper contained a rather irrelevant inconsistency in regarding polymorphic objects as set-theoretic functions on the class of all types.) The basic idea had already been given by Plotkin [5], and a more complicated variant is formed by the logical relations of Mitchell and Meyer [6]. It was generally regarded as merely a representation independence theorem for datatype implementations, while its implications for deriving properties of functional programs seem to have been unrecognized at that time. Quite different approaches are used in Bainbridge, Freyd e.a. [7,8], based on dinatural transformations in a category called **LIN** of certain coherent spaces and linear maps, and in Carboni e.a. [9] where the so-called Realizability Universe is constructed. Both approaches appear to be conceptually far more complicated than Reynolds'. There is also an article by John Gray [10] which dealt only with naturalness of some particular operations like currying.

Our contribution consists of the inclusion of arbitrary initial types, some applications of a different kind than Wadler's applications, a very attractive proof of the dinaturalness-property, and a proof of equivalence between two different recursion-operators.

2 Our language

The proof of the naturalness theorem is by induction on the derivation of the type of an object. We therefore need to specify some polymorphic functional language. It may be either a programming language with fixpoints, where the semantic domain of a type is a cpo, or a purely constructive language where types are just sets and all functions are total. We will use a constructive typed lambda calculus. Besides type-variables α, we have $(\to)$ as the function-type-constructor, and an unspecified number of other type-constructors Θ (possibly user-defined), each constructing from a sequence of types U (of some specific length) the initial (least) datatype that is closed under some object-constructors θ_k, each having a sequence of arguments of some types $D_{kl}[U, \Theta U]$. The $D_{kl}[\alpha, \beta]$ must be functorial in β, so for $q : B \to B'$ we have $D_{kl}[\alpha, q] : D_{kl}[\alpha, B] \to D_{kl}[\alpha, B']$. There is an additional requirement on D_{kl}, to be stated later (3). We use a categorical elimination-construct Θ_elim (compare Hagino [11]), from which other eliminators may be defined.

The derivability relation 't has type T under the assumptions $x_j : S_j$' is denoted $x :: S \vdash t : T$, and is generated by the following rules:

$$\begin{array}{rcll} & & x :: S \vdash x_j : S_j & \{\text{Var-intro}\} \\ x :: S \vdash f : U \to V;\ x :: S \vdash u : U & \Rightarrow & x :: S \vdash fu : V & \{(\to)\text{-elim}\} \\ x :: S, y : U \vdash v : V & \Rightarrow & x :: S \vdash \lambda y.v : U \to V & \{(\to)\text{-intro}\} \\ & & x :: S \vdash \theta_k : D_k[U, \Theta U] \to \Theta U & \{\Theta\text{-intro}\} \\ x :: S \vdash v_k : D_k[U,V] \to V \text{ (each } k) & \Rightarrow & x :: S \vdash \Theta\text{\_elim}(v) : \Theta U \to V & \{\Theta\text{-elim}\} \end{array}$$

There is an untyped conversion relation ($\simeq$), characterized by:

$$\begin{array}{rcl} (\lambda y.v[y])u & \simeq & v[u] \\ \Theta\text{\_elim}(v)(\theta_k d) & \simeq & v_k(D_k[\alpha, \Theta\text{\_elim}(v)]d) \end{array}$$

We will define a typed extensional equivalence in the next section. The theorem can also be given in an extended language that includes final datatypes, and dependent types. In section 6 we will add second-order quantification (in terms of which initial and final datatypes can be defined).

Note that we deal with terms only, not with models. One can give the same results as we do in an arbitrary model for objects representable by terms [3].

3 Naturalness on binary relations

Observe that, although we cannot extend a function p from A to A' to a function from $(U[A] \to V[A])$ to $(U[A'] \to V[A'])$, property (1) suggests considering the binary relation between $f : U[A] \to V[A]$ and $f' : U[A'] \to V[A']$ given by $f ; V[p] = U[p] ; f'$. We will use this observation to extend a relation (instead of a function) between A and A' to one between $T[A]$ and $T[A']$.

Definition. A *relation* $R \subseteq A \times A'$ is a set of pairs of terms of types A and A', taken modulo conversion. (Had we used a programming language permitting the construction of non-terminating programs then we would use elements of the corresponding cpos and R would be required to be closed under directed limits: if $V \subseteq R$ is (pairwise) directed, then $\bigsqcup V \in R$. In particular, $(\bot_A, \bot_{A'}) \in R$ as $\bot$ is the limit of the empty set.)
While we use a colon (:) for typing, ($\in$) denotes relation-membership.

Definition. We will lift any type-constructor Θ to a relation-constructor such that for any relation sequence $R \subseteq A \times A'$ (i.e. $R_i \subseteq A_i \times A'_i$) one has:

$$\Theta R \subseteq \Theta A \times \Theta A'$$

First, as $(A \to B)$ is the greatest type such that for $f : A \to B$ one has $\forall x : A\ .\ fx : B$, we define for $Q \subseteq A \times A'$, $R \subseteq B \times B'$:

$$Q \to R \ := \ \{(f, f') : (A \to B) \times (A' \to B') \mid \forall (x, x') \in Q\ .\ (fx, f'x') \in R\} \qquad (2)$$

That is to say, the pair of functions should map related arguments to related results, as illustrated by:

$$\begin{array}{ccc} A & \xrightarrow{\ f\ } & B \\ \wr Q & & \wr R \\ A' & \xrightarrow{\ f'\ } & B' \end{array}$$

If $\Theta\alpha$ is the initial type closed under object-constructors

$$\theta_k : D_k[\alpha, \Theta\alpha] \to \Theta\alpha$$

where each D_{kl} satisfies for relations P, Q, R

$$(g, g') \in P \to Q \quad \Rightarrow \quad (D_{kl}[\alpha, g], D_{kl}[\alpha, g']) \in D_{kl}[R, P] \to D_{kl}[R, Q] \tag{3}$$

then we define ΘR to be the least relation that is closed under:

$$(\theta_k, \theta_k) \in D_k[R, \Theta R] \to \Theta R \tag{4}$$

This makes sense since for $P \subseteq Q$ we have $D_k[R, P] \subseteq D_k[R, Q]$ by (3), taking g and g' to be the identity I.

Thus, the induction principle for ΘR is: if we wish to prove a predicate P for all pairs in ΘR, then, considering P as a relation $P \subseteq \Theta A \times \Theta A'$, we must show for each k:

$$\forall (d, d') \in D_k[R, P] \,.\, (\theta_k d, \theta_k d') \in P \tag{5}$$

Check for example that all pairs in ΘR are convertible to $(\theta_k d, \theta_k d')$ for some k and $(d, d') \in D_k[R, \Theta R]$

Example. Some relation-constructors corresponding to common type-constructors are

$$\begin{aligned} Q + R &:= \{(\mathbf{inl}(x), \mathbf{inl}(x')) \mid (x, x') \in Q\} \\ &\quad \cup \{(\mathbf{inr}(y), \mathbf{inr}(y')) \mid (y, y') \in R\} \\ Q \times R &:= \{(\langle x, y\rangle, \langle x', y'\rangle) \mid (x, x') \in Q \wedge (y, y') \in R\} \\ \mathbf{Bool} &:= \{(\mathbf{true}, \mathbf{true}), (\mathbf{false}, \mathbf{false})\} \end{aligned}$$

and $\mathbb{N}$ and $R*$ are the least relations such that:

$$\begin{aligned} \mathbb{N} &= \{(0,0)\} \cup \{(\mathbf{succ}(z), \mathbf{succ}(z') \mid (z, z') \in \mathbb{N}\} \\ R* &= \{(\mathbf{nil}, \mathbf{nil})\} \cup \{(\mathbf{cons}\langle x, z\rangle, \mathbf{cons}\langle x', z'\rangle) \mid (x, x') \in R \wedge (z, z') \in R*\} \end{aligned}$$

Now, for any type-expression $T[\alpha]$ containing only type-variables from the sequence α, we have extended a relation-sequence $R \subseteq A \times A'$ to a relation $T[R] \subseteq T[A] \times T[A']$. However, T is not a functor on relations.

Notice that the same schemes (2) and (4) describe extensional equality on ΘA in terms of the equality on the A_i.

Definition. Extensional equality on a closed type T is given by T as a relation:

$$\models t = t' : T \quad :\equiv \quad (t, t') \in T$$

The symbol $\models$ and suffix $: T$ will often be omitted. We shall consider only relations that are closed under this extensional equality. Equality for terms of types containing variables will soon be defined.

Now, if we can derive $t : T[\alpha]$ then not only $t_A : T[A]$ for any type-sequence A (by some substitution theorem), but also $(t_A, t_{A'}) \in T[R]$ for any relation-sequence R. (It is to be understood that *overloaded* operators, like the effective equality-test $(==_A) : A \times A \to$ bool in Miranda, may not be used as if they where polymorphic.) Taking the context into account, we have the following main theorem, equivalent to the abstraction theorem in [4], the fundamental theorem of Logical Relations in [6], and the parametricity result in [3]:

Naturalness theorem. If $x :: S[\alpha] \vdash t[x] : T[\alpha]$ then for any sequences A, A', R, s, s' where $R \subseteq A \times A'$ and $(s_j, s'_j) \in S_j[R]$ one has:

$$(t_A[s], t_{A'}[s']) \in T[R]$$

Remark: we say that $t[x]$ is *natural*. The theorem may be generalized to relations of arbitrary arity.

Proof: by a straightforward induction on the derivation of $x :: S[\alpha] \vdash t[x] : T[\alpha]$:

Var-intro: $x :: S[\alpha] \vdash x_j : S_j[\alpha]$. By assumption we have $(s_j, s'_j) \in S_j[R]$.

$(\rightarrow)$-elim. The hypotheses say $(f[s], f[s']) \in U[R] \rightarrow V[R]$ and $(u[s], u[s']) \in U[R]$. Then by definition of $(\rightarrow)$ on relations we obtain:

$$(f[s]u[s], f[s']u[s']) \in V[R]$$

$(\rightarrow)$-intro. The hypothesis for the premise $x :: S[\alpha], y : U[\alpha] \vdash v[x, y] : V[\alpha]$ says that for any $(s, s') \in S[R]$ and $(u, u') \in U[R]$ one has $(v[s, u], v[s', u']) \in V[R]$.
So $((\lambda y.v[s, y])u, (\lambda y.v[s', y])u') \in V[R]$ as relations are closed under conversion. Hence:

$$(\lambda y.v[s, y], \lambda y.v[s', y]) \in U[R] \rightarrow V[R]$$

Θ-intro. We must show:

$$(\theta_k, \theta_k) \in D_k[U[R], \Theta U[R]] \rightarrow \Theta U[R]$$

This is an instance of (4).

Θ-elim. The (global) hypothesis is: $(v_k[s], v_k[s']) \in D_k[U[R], V[R]] \rightarrow V[R]$ for each k. We must show:

$$(\Theta\text{-elim}(v[s]), \Theta\text{-elim}(v[s'])) \in \Theta U[R] \rightarrow V[R]$$

We use a local induction on the generation of $\Theta U[R]$. Thus we will prove $\Theta U[R] \subseteq P$ where:

$$P \;:=\; \{(t, t') : \Theta U[A] \times \Theta U[A'] \mid (\Theta\text{-elim}(v[s])t, \Theta\text{-elim}(v[s'])t') \in V[R]\}$$

Note that:

$$(\Theta\text{-elim}(v[s]), \Theta\text{-elim}(v[s'])) \in P \rightarrow V[R] \tag{6}$$

We check (5) for each k:

$$\begin{array}{lcll}
 & (d, d') \in D_k[U[R], P] & \\
\Rightarrow & (D_k[\alpha, \Theta\text{-elim}(v[s])]d,\ D_k[\alpha, \Theta\text{-elim}(v[s'])]d') \in D_k[U[R], V[R]] & \{(3) \text{ on } (6)\} \\
\Rightarrow & (v_k[s](D_k[\alpha, \Theta\text{-elim}(v[s])]d),\ v_k[s'](D_k[\alpha, \Theta\text{-elim}(v[s'])]d')) \in V[R] & \{\text{global hyp.}\} \\
\equiv & (\Theta\text{-elim}(v[s])(\theta_k d), \Theta\text{-elim}(v[s'])(\theta_k d')) \in V[R] & \{\text{conversion}\} \\
\equiv & (\theta_k d, \theta_k d') \in P & \{\text{def. } P\}
\end{array}$$

This completes the proof.

The theorem suggests the following definition:

Definition. Extensional equality under assumptions,

$$x :: S \overset{[\alpha]}{\models} t[x] = t'[x] : T[\alpha]$$

holds iff, for all sequences $A, A', R \subseteq A \times A', (s, s') \in S[R]$ one has $(t_A[s], t'_{A'}[s']) \in T[R]$.

Many applications arise by using a sequence of function-like relations:

Definition. For $p : A \to A'$ let the *graph* of p be:

$$(p) \;:=\; \{(x, px) \mid x : A\}$$

$R \subseteq A \times A'$ is called *function-like* if $R = (p)$ for some $p : A \to A'$. (Note that, in a cpo, (p) is closed under directed limits iff p is continuous and strict, i.e. $p\bot_A = \bot_{A'}$.) The naturalness theorem specializes for polymorphic $t : T[\alpha]$ and $p : A \to A'$ to:

$$(t_A, t_{A'}) \in T[(p)] \tag{7}$$

Many type-constructors Θ can be extended to functors in a natural way, so that $(\Theta p) = \Theta(p)$. In particular:

Theorem. If $\Theta\alpha$ is the initial type closed under $\theta_k : D_k[\alpha, \Theta\alpha] \to \Theta\alpha$, and all $D_{kl}[\alpha, \beta]$ are functorial in α (besides β), then there is a unique functor-definition of Θ such that all θ_k are natural transformations. Moreover, if $(D_{kl}[p, q]) = D_{kl}[(p), (q)]$, then $(\Theta p) = \Theta(p)$.

Proof. θ_k being a natural transformation means, according to (1):

$$\theta_k \,;\Theta p = D_k[p, \Theta p] \,;\theta_k : D_k[A, \Theta A] \to \Theta A' \tag{8}$$

Equivalently:

$$\theta_k \,;\Theta p = D_k[\mathrm{I}_A, \Theta p] \,; D_k[p, \mathrm{I}_{A'}] \,;\theta_k : D_k[A, \Theta A] \to \Theta A'$$

By initiality, this has a unique solution for Θp. Further, we prove $(s, s') \in (\Theta p) \equiv (s, s') \in \Theta(p)$ by induction on $s : \Theta A$. Thus we will prove $\Theta A \subseteq P$ where:

$$P \;:=\; \{s : \Theta A \mid \forall s' \,.\, (s, s') \in (\Theta p) \equiv (s, s') \in \Theta(p)\}$$

Under hypothesis $d \in D_k[A, P]$ we check that $\theta_k d \in P$ (but we skip some details about the behaviour of relation-constructors when applying the hypothesis):

To be revised

$$\begin{array}{lll}
 & (\theta_k d, s') \in (\Theta p) & \\
\equiv & (D_k[p, \Theta p] \,;\theta_k) d = s' & \{(8)\} \\
\equiv & \exists d' \,.\, (d, d') \in (D_k[p, \Theta p]) \wedge \theta_k d' = s' & \{\text{one-point rule}\} \\
\equiv & \exists d' \,.\, (d, d') \in D_k[(p), (\Theta p)] \wedge \theta_k d' = s' & \{\text{assumption on } D_k\} \\
\equiv & \exists d' \,.\, (d, d') \in D_k[(p), \Theta(p)] \wedge \theta_k d' = s' & \{\text{induction hypothesis}\} \\
\equiv & (\theta_k d, s') \in \Theta(p) & \{\text{def. } \Theta \text{ on relations}\}
\end{array}$$

So we have, for example, $(p \times q) = (p) \times (q)$ and can safely omit the parentheses. Notice that $p \to q$ can only be read as $(p) \to (q)$.

Fact. For function-like relations we have, if $p : A \to A'$, $q : B \to B'$:

$$p \to q \;=\; \{(f, f') \mid f \,; q = p \,; f' : A \to B'\} \tag{9}$$

$$p^\circ \to q \;=\; \{(f,\; p \,; f \,; q) \mid f : A \to B\} \tag{10}$$

$$\text{using } R^\circ \;:=\; \{(y, x) \mid (x, y) \in R\}$$

So if $f : U[\alpha] \to V[\alpha]$, and hence $(f, f) \in U[p] \to V[p]$ by (7), then is f a natural transformation indeed. Also useful might be, for $f : A' \to B$:

$$(p \,; f,\; f \,; q) \in p \to q \tag{11}$$

4 Applications

A simple application is to prove that $f = \lambda x.x$ is the only solution to $f : \alpha \to \alpha$. Naturalness of f says: for any $R \subseteq A \times A'$ we have $(f, f) \in R \to R$.
Now, fix type A and $a : A$. Taking $R := \{(a, a)\}$ yields $(fa, fa) \in R$, as $(a, a) \in R$. So $fa = a$ for all a, hence $f = \lambda x.x$ by extensionality of functions.

(If types are cpos there are two solutions. If $a \neq \bot$, we must take $R := \{(\bot, \bot), (a, a)\}$ and get $fa \in \{\bot, a\}$. Furthermore, for any $b : B$ we can get $(fa, fb) \in \{(\bot, \bot), (a, b)\}$. So in case $fa = \bot$ we have $fb = \bot$ for all b, hence $f = \lambda x.\bot$; and in case $fa = a$ we obtain $fb = b$ and hence $f = \lambda x.x$.)

Next, let $(*)$ be a functor, say of lists, so for $p : A \to A'$ we have $p* : A* \to A'*$ and $(p*) = (p)*$. Let f be a function with the type of *foldr*, i.e.:

$$f : (\alpha \times \beta \to \beta) \times \beta \to (\alpha* \to \beta)$$

Naturalness of f on function-like relations says: if $p : A \to A'$, $q : B \to B'$ then

$$(f, f) \in (p \times q \to q) \times q \to (p* \to q)$$

i.e. if $(c, c') \in (p \times q \to q)$ and $(b, b') \in (q)$ then $(f\langle c, b\rangle, f\langle c', b'\rangle) \in (p* \to q)$.
Using (9) this equivales: if

$$c\,;q = (p \times q)\,;c' : A \times B \to B'$$

then:

$$f\langle c, b\rangle\,;q = p*\,;f\langle c', qb\rangle : A* \to B'$$

(This is a generalization of the promotion-theorem for forward lists [12].) Notice that the result is independent of the definition of f.

In particular, we have for $\oplus : A' \times B' \to B$, as by (11) $((p \times q)\,;\oplus,\ \oplus\,;q) \in (p \times q \to q)$:

$$f\langle(p \times q)\,;\oplus,\ b\rangle\,;q = p*\,;f\langle\oplus\,;q,\ qb\rangle \tag{12}$$

Another instance yields, as $\mathbf{cons}\,;foldr\langle c', b'\rangle = (\mathrm{I} \times foldr\langle c', b'\rangle)\,;c'$ and $foldr\langle c', b'\rangle\mathbf{nil} = b'$:

$$f\langle\mathbf{cons}, \mathbf{nil}\rangle\,;foldr\langle c', b'\rangle = f\langle c', b'\rangle$$

Finally, we give an application[1] using *ternary* relations. Let $(*)$ be a functor, say of non-empty lists, and $(/_B)$ a (polymorphic) mapping of operators $\oplus : B \times B \to B$ into $\oplus/ : B* \to B$.

We will prove: if $f, g : A \to B$, and $\oplus : B \times B \to B$ is commutative and associative, then for $l : A*$,

$$\oplus/(f*l) \oplus \oplus/(g*l) = \oplus/((f \oplus^A g)*l)$$

where $\oplus^A : (A \to B) \times (A \to B) \to (A \to B)$ is the lifted version of $\oplus$. We will regard $\oplus$ as a ternary relation $\oplus \subseteq B \times B \times B$ so that:

$$(x, y, z) \in \oplus \quad := \quad x \oplus y = z$$

We derive:

$$\begin{array}{ll} & \forall l : A* \,.\, (\oplus/(f*l),\ \oplus/(g*l),\ \oplus/((f \oplus^A g)*l)) \in \oplus \\ \Leftarrow & (f*, g*, (f \oplus^A g)*) \in A* \to \oplus* \ \wedge\ (\oplus/, \oplus/, \oplus/) \in \oplus* \to \oplus \end{array}$$

[1] Posed by Roland Backhouse

$$
\begin{array}{lll}
\Leftarrow & (f,g,(f \oplus^A g)) \in A \to \oplus \ \wedge \ (\oplus,\oplus,\oplus) \in \oplus \times \oplus \to \oplus & \{\text{naturalness of } (*),\ (/)\ \} \\
\equiv & \left\{ \begin{array}{l} \forall x : A \,.\, fx \oplus gx = (f \oplus^A g)x \\ \ \wedge\ \forall (x_k, y_k, z_k) \in \oplus \,.\, (x_0 \oplus x_1, y_0 \oplus y_1, z_0 \oplus z_1) \in \oplus \end{array} \right. & \\
\equiv & \left\{ \begin{array}{l} \text{true} \\ \ \wedge\ \forall x_k, y_k \,.\, (x_0 \oplus x_1) \oplus (y_0 \oplus y_1) = (x_0 \oplus y_0) \oplus (x_1 \oplus y_1) \end{array} \right. & \\
\Leftarrow & \oplus \text{ is commutative and associative} &
\end{array}
$$

5 Dinaturalness

We will now derive the dinaturalness result from our general naturalness.

Definition. A *difunctor* $T[\alpha\|\beta]$ is given by a mapping of sequences of types A, B to a type $T[A\|B]$ and a mapping of sequences of functions $p : A \to A'$, $q : B \to B'$ to a function

$$T[p\|q] : T[A'\|B] \to T[A\|B']$$

such that identity and composition are respected (note the contravariance in α):

$$
\begin{array}{rcl}
T[\mathrm{I}_A\|\mathrm{I}_B] & = & \mathrm{I}_{T[A\|B]} \\
T[p\,;p'\|q\,;q'] & = & T[p'\|q]\,;T[p\|q']
\end{array}
$$

Keep in mind that types such as $T[A'\|A]$ are lifted to their equality-relation. We will also use the simple fact that composition is natural:

$$(f,f') \in Q \to R \ \wedge \ (g,g') \in R \to S \ \ \Rightarrow \ \ (f\,;g,\ f'\,;g') \in Q \to S$$

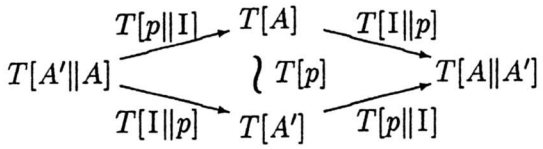
The following theorem is a reformulation of a lemma found by Roland Backhouse.

Theorem. With any type-expression $T[\alpha]$ we can associate a difunctor $T[\alpha\|\beta]$ such that $T[A] = T[A\|A]$, and for $p : A \to A'$ one has both:

$$(T[p\|\mathrm{I}_A], T[\mathrm{I}_{A'}\|p]) \in T[A'\|A] \to T[p] \tag{13}$$

$$(T[\mathrm{I}_A\|p], T[p\|\mathrm{I}_{A'}]) \in T[p] \to T[A\|A'] \tag{14}$$

That is to say, the first pair of functions map equal arguments to related results, the second pair map related arguments to equal results, as illustrated by:



Do not confuse relation $T[p] \subseteq T[A] \times T[A']$ and function $T[p\|p] : T[A'\|A] \to T[A\|A']$.

Proof: by induction on the structure of $T[\alpha]$.

If $T[\alpha] = \alpha_i$ then defining $T[A\|B] := B_i$ and $T[p\|q] := q_i$ does the job. To be revised

If $T[\alpha] = U[\alpha] \to V[\alpha]$ then the following lines define a difunctor:

$$
\begin{array}{rcl}
T[A\|B] & := & U[B\|A] \to V[A\|B] \\
T[p\|q] & := & U[q\|p] \multimap V[p\|q] \\
\text{using } (u \multimap v)f & := & u\,;f\,;v
\end{array}
$$

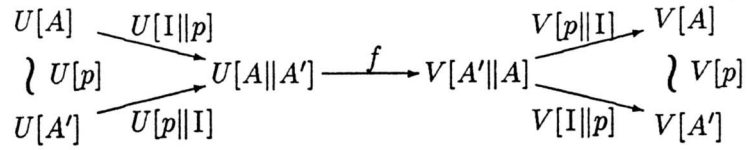
Remark that, as a relation, $(u \circ\!\!\rightarrow v) = (u^\circ \rightarrow v)$ by (10).
To check (13) we must show:

$$(U[\mathrm{I}\|p] \circ\!\!\rightarrow V[p\|\mathrm{I}],\ U[p\|\mathrm{I}] \circ\!\!\rightarrow V[\mathrm{I}\|p]) \in (U[A\|A'] \rightarrow V[A'\|A]) \rightarrow (U[p] \rightarrow V[p])$$

That is, for $f : U[A\|A'] \rightarrow V[A'\|A]$ we must have:

$$(U[\mathrm{I}\|p]\,;f\,;V[p\|\mathrm{I}],\ U[p\|\mathrm{I}]\,;f\,;V[\mathrm{I}\|p]) \in U[p] \rightarrow V[p]$$

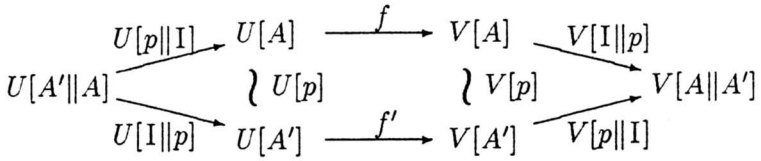
This holds by composition of the inductive hypothesis (14) for U, the typing of f, and hypothesis (13) for V:



Similarly we check (14), that is, for $(f, f') \in U[p] \rightarrow V[p]$ we must have:

$$U[p\|\mathrm{I}]\,;f\,;V[\mathrm{I}\|p] = U[\mathrm{I}\|p]\,;f'\,;V[p\|\mathrm{I}] : U[A'\|A] \rightarrow V[A\|A] \tag{15}$$

This holds by composition of hypothesis (13) for U, the requirement on (f, f'), and hypothesis (14) for V:



We skip initial datatypes, but remark that for functors Θ the definition is just $(\Theta U)[\alpha\|\beta] := \Theta(U[\alpha\|\beta])$.

To summarize, $T[p\|q]$ is constructed from $T[\alpha]$ by replacing all *negative* occurrences of α_i by p_i, all *positive* occurrences by q_i, and all arrows $(\rightarrow)$ by $(\circ\!\!\rightarrow)$. Now, combining naturalness-result (7) with (14) we get:

Corollary. All polymorphic $t : T[\alpha]$ satisfy $T[\mathrm{I}\|p]t_A = T[p\|\mathrm{I}]t_{A'}$ for $p : A \rightarrow A'$. In particular, all polymorphic $f : U[\alpha] \rightarrow V[\alpha]$ are *dinatural transformations*, that is to say, they satisfy (15) with $f, f' := f_A, f_{A'}$.

For example, any function

$$f : (\alpha \times \beta \rightarrow \beta) \times \beta \rightarrow (\alpha* \rightarrow \beta)$$

will satisfy for $p : A \rightarrow A'$, $q : B \rightarrow B'$:

$$((p \times q \circ\!\!\rightarrow \mathrm{I}) \times \mathrm{I})\,;f_{AB}\,;(\mathrm{I}* \circ\!\!\rightarrow q) = ((\mathrm{I} \times \mathrm{I} \circ\!\!\rightarrow q) \times q)\,;f_{A'B'}\,;(p* \circ\!\!\rightarrow \mathrm{I})$$

Applied to some $\langle \oplus, b\rangle : (A' \times B' \rightarrow B) \times B$, this is our result (12) in section 4.

As all pairs in $U[p]$ need not to occur in the range of $(U[p\|\mathrm{I}], U[\mathrm{I}\|p])$, the statement of dinaturalness seems to be weaker than general naturalness. However, we know of no example family $t_A : T[A]$ such that $T[\mathrm{I}\|p]t_A = T[p\|\mathrm{I}]t_{A'}$ for all $p : A \rightarrow A'$ but not $(t_A, t_{A'}) \in T[R]$ for all $R \subseteq A \times A'$.

6 Second-order languages

We may use a second-order language, where, say, $\forall\alpha.T[\alpha]$ is a type with:

$$\begin{array}{rcl} x :: S \vdash t : T[\alpha], \text{ where } \alpha \text{ does not occur free in } S & \Rightarrow & x :: S \vdash t : \forall\alpha.T[\alpha] \\ x :: S \vdash t : \forall\alpha.T[\alpha] & \Rightarrow & x :: S \vdash t : T[U] \end{array}$$

The appropriate extension of relations is, if $R[Q] \subseteq T[A] \times T[A']$ for any $Q \subseteq A \times A'$ (that is closed under extensional equality):

$$\forall\rho.R[\rho] \;:=\; \{(t,t') \mid \forall A, A' \textbf{ type}, Q \subseteq A \times A' \,.\, (t,t') \in R[Q]\}$$

As an application, we define type $\mathbb{N}$, $\mathbf{z} : \mathbb{N}$ and $\mathbf{s} : \mathbb{N} \to \mathbb{N}$ by:

$$\begin{array}{rcl} \mathbb{N} & := & \forall\alpha.((\alpha \to \alpha) \to (\alpha \to \alpha)) \\ \mathbf{z} & := & \lambda f.\lambda a.a \\ \mathbf{s} & := & \lambda m.\lambda f.\lambda a.f(mfa) \end{array}$$

We will prove Peano's induction-axiom, i.e. assuming

$$P \subseteq \mathbb{N},\ \mathbf{z} \in P,\ \forall m \in P \,.\, \mathbf{s}m \in P$$

we will prove $n \in P$ for all $n : \mathbb{N}$. Remember that naturalness of n says that for any types A, A', relation $Q \subseteq A \times A'$ we have $(n,n) \in (Q \to Q) \to (Q \to Q)$.
The proof is in two steps (but a generalization of the naturalness-property may allow for a single-step proof).

1. Take a *predicate-like* relation $Q := \{(n,n) \mid n \in P\}$. The assumptions say $(\mathbf{s},\mathbf{s}) \in (Q \to Q)$ and $(\mathbf{z},\mathbf{z}) \in Q$, hence by naturalness of n we get $(n\mathbf{s}, n\mathbf{s}) \in (Q \to Q)$ and $(n\mathbf{sz}, n\mathbf{sz}) \in Q$, i.e. $n\mathbf{sz} \in P$.

2. What remains to prove is $n\mathbf{sz} = n$, i.e. for any type A, $f : A \to A$, $a : A$ we must prove $n\mathbf{sz}fa = nfa$.
   Taking $Q := \{(m,x) : \mathbb{N} \times A \mid mfa = x\}$, naturalness guarantees $(n\mathbf{sz}, nfa) \in Q$ provided $(\mathbf{s},f) \in (Q \to Q)$ and $(\mathbf{z},a) \in Q$. But these properties hold by definition of $\mathbf{s}$ and $\mathbf{z}$.

7 Overloaded operators

We remarked that polymorphic functions may not use overloaded operators, like an effective equality-test $(==_A) : A \times A \to \text{bool}$. However, if we require types instantiated for type-variables to support certain operations, we can give similar requirements on relations. Such restrictions may be provided explicitly by a "type class" in the language Haskell [13].

Definition. Let z be the class of types α with associated operations $v_i : T_i[\alpha]$, and let A and A' be two "instances" of z with operations $t_i : T_i[A]$ and $t'_i : T_i[A']$. Written in Haskell:

$$\begin{array}{l} \texttt{class } z\ \alpha \texttt{ where } \{\ v_1 :: T_1[\alpha]\ ;;\ \ldots ;;\ v_n :: T_i[\alpha]\ \} \\ \texttt{instance } z\ A \texttt{ where } \{\ v_1 = t_1\ ;;\ \ldots ;;\ v_n = t_n\ \} \\ \texttt{instance } z\ A' \texttt{ where } \{\ v_1 = t'_1\ ;;\ \ldots ;;\ v_n = t'_n\ \} \end{array}$$

A relation $R \subseteq A \times A'$ is said to *respect* class z, iff for each i, one has $(t_i, t'_i) \in T_i[R]$.

For example, consider the class Eq of types a with equality-test:

```
class Eq a where (==) ::  a -> a -> Bool
```

Relation R respects Eq iff for all $(x, x') \in R$, $(y, y') \in R$ one has $(x == y) = (x' == y') : \mathtt{Bool}$. Note that not all relations have this property, hence $(==)$ is not natural. But one can prove the following variant:

Restricted naturalness. If expression s has type $S[\alpha]$ for any instance α of class z as above, expressed in Haskell by

$$s \ :: \ z \ \alpha \ \Rightarrow \ S[\alpha]$$

then for any relation $R \subseteq A \times A'$ that respects z we have that $(s, s) \in S[R]$.

8 Application to inductive definitions

If V is a functor on types, then one may form the inductive type μV, being the initial (least) type that has a constructor

$$\mathbf{C} : V[\mu V] \to \mu V$$

and also νV, being the final (greatest) type that has a destructor:

$$\mathbf{D} : \nu V \to V[\nu V]$$

Two different recursion operators were given by Hagino [11] and Mendler [14]. Using naturalness we can prove these to be equivalent: each is definable from the other one and can be shown to have the same properties. We will concentrate on ν-types.

The categorical approach from [11] (restricted to ν-types with a single destructor) says: if there is another type T with an arrow $f : T \to V[T]$ then there is a unique mediating arrow $\mathbf{P}(f) : T \to \nu V$ satisfying $\mathbf{P}(f) ; \mathbf{D} = f ; V[\mathbf{P}(f)]$:

$$\forall p : T \to \nu V \,.\, (p = \mathbf{P}(f) \ \equiv \ p ; \mathbf{D} = f ; V[p]) \tag{16}$$

So there is a conversion rule $\mathbf{D}(\mathbf{P}(f)t) \simeq V[\mathbf{P}(f)](ft)$.

The recursive definition approach from [14] used *containments* like $\nu V \subseteq \alpha$; we will use explicit embeddings instead. The recursion principle says: for any type T, if for any type α, in which νV is embedded via $e : \nu V \to \alpha$, and for which one assumes an induction hypothesis $h : T \to \alpha$ one has constructed a function $g_\alpha e h : T \to V[\alpha]$, then there is a unique[2] solution $\mathbf{G}(g) : T \to \nu V$ for p in the recursive equation $p ; \mathbf{D} = g_{\nu V} \, \mathrm{I} \, p$. So if

$$g : \forall \alpha . (\nu V \to \alpha) \to (T \to \alpha) \to (T \to V[\alpha]) \tag{17}$$

then $\mathbf{G}(g) : T \to \nu V$ and:

$$\forall p : T \to \nu V \,.\, (p = \mathbf{G}(g) \ \equiv \ p ; \mathbf{D} = g_{\nu V} \, \mathrm{I} \, p) \tag{18}$$

Thus, the conversion rule is $\mathbf{D}(\mathbf{G}(g)t) \simeq g \, \mathrm{I} \, (\mathbf{G}(g))t$.

[2] Uniqueness is essential but was not mentioned by Mendler.

Now we will prove the equivalence. Assuming (18) and $f : T \to V[T]$ we can easily give a definition for $\mathbf{P}(f)$, by finding a g such that $g\,\mathrm{I}\,p = f\,;V[p]$:

$$\mathbf{P}(f) := \mathbf{G}(g) \text{ where } g := \lambda e.\lambda h.f\,;V[h]$$

We check (16):

$$\begin{array}{lll} & p = \mathbf{P}(f) & \\ \equiv & p = \mathbf{G}(\lambda e.\lambda h.f\,;V[h]) & \{\text{definition}\} \\ \equiv & p\,;\mathbf{D} = f\,;V[p] & \{(18)\} \end{array}$$

Conversely, assuming (16) and g satisfying (17) we must define the $\mathbf{G}(g)$ satisfying (18). If we instantiate α to T we cannot give an embedding $e : \nu V \to \alpha$. We take $\nu V + T$ instead, and get:

$$g_{\nu V+T}(\mathbf{inl})(\mathbf{inr}) : T \to V[\nu V + T]$$

To apply $\mathbf{P}$ we need some $f : \nu V + T \to V[\nu V + T]$, so that $\mathbf{P}(f) : \nu V + T \to \nu V$. We take:

$$\begin{array}{rcl} f & := & +\text{-elim}(\mathbf{D}\,;V[\mathbf{inl}],\, g(\mathbf{inl})(\mathbf{inr})) \\ \mathbf{G}(g) & := & \mathbf{inr}\,;\mathbf{P}(f) \end{array}$$

We will use the following properties of +-elim. Distributivity property (20) is in fact a case of naturalness.

$$p = +\text{-elim}(v_0, v_1) \equiv \mathbf{inl}\,;p = v_0 \wedge \mathbf{inr}\,;p = v_1 \qquad (19)$$

$$+\text{-elim}(v_0, v_1)\,;q = +\text{-elim}(v_0\,;q,\; v_1\,;q) \qquad (20)$$

We will need a lemma stating $\mathrm{I} = \mathbf{inl}\,;\mathbf{P}(f)$. We have $\mathrm{I} = \mathbf{P}(\mathbf{D})$ by (16) from $\mathrm{I}\,;\mathbf{D} = \mathbf{D}\,;V[\mathrm{I}]$. And we have:

$$\begin{array}{lll} & \mathbf{inl}\,;\mathbf{P}(f) = \mathbf{P}(\mathbf{D}) & \\ \equiv & \mathbf{inl}\,;\mathbf{P}(f)\,;\mathbf{D} = \mathbf{D}\,;V[\mathbf{inl}\,;\mathbf{P}(f)] & \{(16) \text{ for } (\mathbf{inl}\,;\mathbf{P}(f))\} \\ \equiv & \mathbf{inl}\,;f\,;V[\mathbf{P}(f)] = \mathbf{D}\,;V[\mathbf{inl}\,;\mathbf{P}(f)] & \{(16) \text{ for } \mathbf{P}(f)\} \\ \equiv & \mathbf{D}\,;V[\mathbf{inl}]\,;V[\mathbf{P}(f)] = \mathbf{D}\,;V[\mathbf{inl}\,;\mathbf{P}(f)] & \{\text{definition } f,\ +\text{-elim}\} \\ \equiv & \text{true} & \{V \text{ is a functor}\} \end{array}$$

Now for (18):

$$\begin{array}{lll} & p = \mathbf{inr}\,;\mathbf{P}(f) & \\ \equiv & \mathrm{I} = \mathbf{inl}\,;\mathbf{P}(f) \wedge p = \mathbf{inr}\,;\mathbf{P}(f) & \{\text{lemma}\} \\ \equiv & +\text{-elim}(\mathrm{I}, p) = \mathbf{P}(f) & \{(19)\} \\ \equiv & +\text{-elim}(\mathrm{I}, p)\,;\mathbf{D} = f\,;V[+\text{-elim}(\mathrm{I}, p)] & \{(16) \text{ for } \mathbf{P}(f)\} \\ \equiv & +\text{-elim}(\mathrm{I}, p)\,;\mathbf{D} = +\text{-elim}(\mathbf{D}\,;V[\mathbf{inl}],\, g(\mathbf{inl})(\mathbf{inr}))\,;V[+\text{-elim}(\mathrm{I}, p)] & \{\text{definition } f\} \\ \equiv & +\text{-elim}(\mathbf{D}, p\,;\mathbf{D}) = +\text{-elim}(\mathbf{D}\,;V[\mathrm{I}],\, g(\mathbf{inl})(\mathbf{inr})\,;V[+\text{-elim}(\mathrm{I}, p)]) & \{2\times(20);\ +\text{-elim}\} \\ \equiv & \mathbf{D} = \mathbf{D} \wedge p\,;\mathbf{D} = g(\mathbf{inl})(\mathbf{inr})\,;V[+\text{-elim}(\mathrm{I}, p)] & \{(19)\} \\ \equiv & p\,;\mathbf{D} = g(\mathbf{inl}\,;+\text{-elim}(\mathrm{I}, p))(\mathbf{inr}\,;+\text{-elim}(\mathrm{I}, p)) & \{\text{naturalness of } g\} \\ \equiv & p\,;\mathbf{D} = g\,\mathrm{I}\,p & \{+\text{-elim}\} \end{array}$$

Then we are done.

The proof dualizes completely to μ-types by reversing all (first order) arrows and compositions. But the recursor for μ-types can be generalized to *dependent* functions. If *equality-types* are present then the uniqueness-condition is no longer needed for this recursor, because it will allow for inductive equality-proofs in the language itself. In fact, using naturalness one can then prove that the non-dependent recursor with a uniqueness-axiom is equivalent to the dependent recursor without such an axiom.

Acknowledgement. Thanks are due to Roland Backhouse and Wim Hesselink for many comments that greatly improved upon our presentation.

References

[1] David E. Rydeheard, *Functors and Natural Transformations*. In Category Theory and Computer Programming 1985, LNCS 240, pp. 43–57.

[2] S. Mac Lane, *Categories for the working mathematician*. Graduate Texts in Mathematics 5, Springer Verlag, Berlin 1971.

[3] Philip Wadler, *Theorems for free!* (draft). Submitted to "Functional Programming and Computer Architecture", September 1989, London.

[4] J.C. Reynolds, *Types, abstraction, and parametric polymorphism*. In Information Processing 1983, North-Holland, Amsterdam, pp. 513–523.

[5] G.D. Plotkin, *Lambda-definability in the full type hierarchy*. In "To H.B. Curry: Essays on combinatory logic, lambda calculus, and formalism", Academic Press, New York 1980, pp. 363–373.

[6] J.C. Mitchell and A.R. Meyer, *Second-order logical relations*. In Logics of Programs 1985, LNCS 193, pp. 225–236.

[7] E.S. Bainbridge, P.J. Freyd, A. Scedrov, P.J. Scott, *Functorial Polymorphism*. In Logical Foundations of Functional Programming, Austin, Texas 1987, Addison-Wesley.

[8] P.J. Freyd, J.Y. Girard, A. Scedrov, P.J. Scott, *Semantic Parametricity in Polymorphic Lambda Calculus*. In Logic in Computing Science 1988, IEEE, pp. 274–279.

[9] A. Carboni, P.J. Freyd and A. Scedrov, *A Categorical Approach to Realizability and Polymorphic Types*. In Mathematical Foundations of Programming Language Semantics 1987, LNCS 298, pp. 23–42.

[10] John W. Gray, *A Categorical Treatment of Polymorphic Operations*. In Mathematical Foundations of Programming Language Semantics 1987, LNCS 298, pp. 2–22.

[11] Tatsuya Hagino, *A Typed Lambda Calculus with Categorical Type Constructors*. In Category Theory and Computer Science 1987, LNCS 283, pp. 140–157.

[12] Grant Malcolm, *Homomorphisms and Promotability*. In Mathematics of Program Construction 1989, LNCS 375, pp. 335–347

[13] P. Hudak and P. Wadler, editors, *Report on the Functional Programming Language Haskell*. Technical Report, Yale University and University of Glasgow, Department of Computer Science, December 1988.

[14] P.F. Mendler, *Inductive Definition in Type Theory*. Ph.D. Thesis, Dep. of Computer Science, Cornell University, 1987.

Addition to 'Naturalness of Polymorphism', page 6.

We present a much easier and more general result about functors than the theorem on page 6.

Theorem. If a type- and relation-constructor Θ is also naturally defined on functions such that $\Theta p : \Theta A \to \Theta A'$ for $p : A \to A'$, and $\Theta \mathrm{I}_A = I_{\Theta A}$,
then $(\Theta p) = \Theta(p)$ for all functions p, and Θ is a functor (i.e. it preserves composition). So the parentheses can be safely omitted.

Proof. Remark first that, by the definition of graph and of the $\to$-constructor, we have for $Q \subseteq A \times A'$:

$$(p) \subseteq Q \;\equiv\; (\mathrm{I}_A, p) \in A \to Q$$

and

$$Q \subseteq (p) \;\equiv\; (p, \mathrm{I}_{A'}) \in Q \to A'$$

Thus:

$$\begin{array}{lll} & (\Theta p) \subseteq \Theta(p) & \\ \equiv & (\mathrm{I}_{\Theta A}, \Theta p) \in \Theta A \to \Theta(p) & \\ \equiv & (\Theta \mathrm{I}_A, \Theta p) \in \Theta A \to \Theta(p) & \{\Theta \text{ preserves identity}\} \\ \Leftarrow & (\mathrm{I}_A, p) \in A \to (p) & \{\text{Naturality of } \Theta\} \\ \equiv & (p) \subseteq (p) & \end{array}$$

Conversely:

$$\begin{array}{lll} & \Theta(p) \subseteq (\Theta p) & \\ \equiv & (\Theta p, \mathrm{I}_{\Theta A'}) \in \Theta(p) \to \Theta A' & \\ \equiv & (\Theta p, \Theta \mathrm{I}_{A'}) \in \Theta(p) \to \Theta A' & \{\Theta \text{ preserves identity}\} \\ \Leftarrow & (p, \mathrm{I}_{A'}) \in (p) \to A & \{\text{Naturality of } \Theta\} \\ \equiv & (p) \subseteq (p) & \end{array}$$

Thus $(\Theta p) = \Theta(p)$.

For composition, suppose $p : A \to B$ and $q : B \to C$. Then $(p\,;q,\, q) \in (p) \to C$.
So by naturality $(\Theta(p\,;q),\, \Theta q) \in \Theta p \to \Theta C$, i.e. $\Theta(p\,;q) = \Theta p\,;\Theta q$.