

Data Categories

C. Barry Jay

School of Computing Sciences
University of Technology, Sydney
P.O. Box 123 Broadway NSW 2007 Australia
cbj@socs.uts.edu.au

Abstract

Data categories and functors, and the strong natural transformations between them provide a universe in which to model parametric polymorphism. Data functors are distinguished by being decomposable into shape and data, i.e. they represent types that store data. Every strong transformation between two such is given by a uniform algorithm, and so may represent a polymorphic term.

The data functors are closed under composition, finite products and sums, exponentiation by an object, final co-algebras and initial algebras. For any two such, the collection of strong natural transformations between them is representable by an object.

The covariant type system supports parametric polymorphism on data types, and can be modelled in a data category. Since the category of sets is a data category, it follows that parametric polymorphism can have a set-theoretic model.

Keywords data categories covariance parametric polymorphism.

1 Introduction

This paper introduces data functors, the data categories in which they live, and establishes their ability to model a large class of the data types that occur in computing. In particular, they can be used to give a simple account of parametricity, and a set-theoretic account of some polymorphic types.

Various attempts have been made to represent data type constructors as functors, with polymorphic terms represented as natural transformations. For example, let $F(X)$ and $G(X)$ (or just FX and GX) be types built using a type variable X and let $\alpha : \forall X. FX \rightarrow GX$ be a polymorphic term. Then F and G should be modelled by functors and α should be modelled by a natural transformation $F \Rightarrow G$. That is, for each type A there is a morphism $\alpha_A : FA \rightarrow GA$. further, if $f : A \rightarrow B$ is any morphism then the following diagram commutes:

$$\begin{array}{ccc} FA & \xrightarrow{\alpha_A} & GA \\ Ff \downarrow & & \downarrow Gf \\ FB & \xrightarrow{\alpha_B} & GB \end{array}$$

Commutativity here asserts that the action of α is unaffected by substitutions that act on the arguments of the functor. There are various difficulties with this simple account.

First, if the type variable X appears in negative positions (say, as the domain of a function type) then the functors will have contravariant, as well as covariant arguments, and the simple notion of naturality is inadequate; dinatural transformations will be required, which bring their own difficulties. We will put this problem aside for now.

Second, even if the functors are both covariant, naturality is, in general, a weaker concept than parametricity, or uniformity. One way of describing the parametricity of the polymorphic term α is to require that the same algorithm is used for each choice of type A for X . For example, the algorithm for appending lists is typically parametric, while that for addition (of integers or reals) is not. It is certainly true that parametric terms yield natural transformations. The problem is that there can be many natural transformations that are not parametric in this sense. In particular, there may be a proper class of natural transformations, while the collection of algorithms is surely small.

So the question arises: which natural transformations are uniform? To date, I am unaware of any adequate answer. Such difficulties underly one of the fundamental results of the semantics of polymorphism: that the second-order polymorphic lambda-calculus has no set-theoretic models [9].

This work modifies the question to ask: which functors represent data types? The answer is (of course) the data functors. There are enough of them to model a generous class of data types, and yet they have the following key property, discovered after a remark of Eugenio Moggi: every natural transformation between a pair of data functors is

given by a uniform algorithm. Not only is the original question answered, but the construction of the algorithm from the transformation is particularly simple, and may yield new insights into the nature of parametricity.

The data functors are defined using ideas from shape theory. If F is a data functor of one argument (i.e. is unary) then values of type FA can be decomposed into their shape, of type $F1$ and their data, given by a partial function from an *object of positions* P (determined by F) into A . Here P represents all the possible positions where data may be found. The choice of actual positions is determined by the shape. For example, the positions in a matrix are indexed by $P = N \times N$. In a tree with branches labelled by Q the node positions are given by paths from the root, specified by lists of Q , so that $P = LQ$.

It will be proved that all natural transformations $\alpha : F \Rightarrow G$ between data functors act by filling positions in the G -shape with data from specified positions in the F -shape. The process is parametrized by the F -shape of the argument, so that every natural transformation is determined by a morphism $F1 \rightarrow GP$. Since data categories are cartesian closed, we can represent the natural transformations as a decidable sub-object of the function object $F1 \rightarrow GP$. Note that this object is independent of the data A , so that we have a parametric description of the algorithm for α .

Earlier versions of the theory stored the data in lists. While adequate for first-order shapely types, such an approach cannot handle higher-order types, since (unlike position functors) lists are not closed under exponentials.

Data categories are defined to provide a setting for the study of data functors. Examples of data categories include **Sets**, **Pos** _{ω} (the category of bottomless, ω -complete partial orders) [10] and **Eff**, the effective topos [4]. To the locos structure (lextensive categories with list objects) used to model shapely type constructors we must add cartesian closure for the higher-order structure.

The bulk of the paper is devoted to defining terms, establishing the above result, and showing that the data functors are closed under many of the usual constructions desired of data types. Finally, the data functors are used to construct semantics for a polymorphic type system, namely the *covariant type system* introduced in the companion paper to this [6]. Covariant types correspond to a subsystem of the second-order polymorphic lambda-calculus, which is strong enough to model the usual polymorphism of inductive types, such as lists and trees (though some higher-order polymorphism is lost). Since the category of sets is a data category, it follows that we have constructed a set-theoretic

model of polymorphism, the first such result of this kind.

Previous models of polymorphism, e.g. [8] have ingeniously built computational concepts into the structure of the semantics, at the cost of greater complexity, and reduced explanatory power. The current approach captures the uniformity in a purely semantic way; the structure of the functors enforces uniformity in the transformations.

The sections of the paper are: Introduction; Cartesian closed locales; Position functors; Unary data functors; General data functors, Semantics of the covariant types, and; Conclusions.

2 Cartesian closed locales

This section provides a brief overview of locales and shape theory, and develops the additional logical power derived from the interaction of cartesian closure with finite limits. Further details and all undefined list notation can be found in [7]. Typical examples of cartesian closed locales are **Sets**, toposes that have list objects, such as **Eff**, and **Pos** _{ω} .

A *locos* [3] is an extensive category [2] with all finite limits, and list objects, defined using initial algebras for linear polynomial functors, e.g. $FX = 1 + A \times X$.

The list functor $L : \mathcal{D} \rightarrow \mathcal{D}$ is an example of a *shapely* functor. That is, it has a *strength*

$$\tau_{A,B} : LA \times B \rightarrow L(A \times B)$$

and preserves all pullbacks. Other examples are the coproduct functor $+ : \mathcal{D}^2 \rightarrow \mathcal{D}$ and the *exponential* $P \rightarrow (-) : \mathcal{D} \rightarrow \mathcal{D}$ for each object P .

Shapely types are characterised semantically, by the property that their values can be decomposed into a pair consisting of a shape and some data. Typically, the data of type A can be represented by a finite list of type LA . Then each shape (of type S) has an arity (a natural number) and the shape-data pair must satisfy the constraint that the arity of the shape equals the length of the data list. That is, shapely types FA are given by pullbacks of the following form:

$$\begin{array}{ccc} FA & \xrightarrow{\text{data}} & LA \\ \downarrow \# & \lrcorner & \downarrow \# \\ S & \xrightarrow{\text{arity}} & N \end{array}$$

It follows that F is a functor, with $S \cong F1$, and $\text{data} : F \Rightarrow L$ is a cartesian (natural) transformation from F to lists. Such functors inherit various properties from the list functor; in particular, they are *shapely*. They are also closed under composition,

products, sums and formation of initial algebras [7], making them suitable for modelling the data types of a first-order language.

The object $2 = 1 + 1$ will play the role of the booleans in \mathcal{D} with $\text{true}, \text{false} : 1 \rightarrow 2$ given by the coproduct inclusions inl and inr respectively. The existence of finite limits means that every predicate $\chi : A \rightarrow 2$ defines a *decidable* subobject A_0 of A on which it is true, given by the pullback:

$$\begin{array}{ccc} A_0 & \longrightarrow & A \\ \downarrow & \lrcorner & \downarrow \chi \\ 1 & \xrightarrow{\text{true}} & 2 \end{array}$$

We can also combine predicates using classical boolean connectives.

The presence of cartesian closure means that we can also *quantify* predicates as follows. For each object B define $\text{all}_B : 1 \rightarrow 2^B$ to be the result of currying $\text{true} \circ ! : 1 \times B \rightarrow 2$. Then, given a predicate $\chi : A \times B \rightarrow 2$ define the subobject $\forall B.\chi$ of A by the following pullback:

$$\begin{array}{ccc} \forall B.\chi & \longrightarrow & A \\ \downarrow & \lrcorner & \downarrow \text{curry}(\chi) \\ 1 & \xrightarrow{\text{all}_B} & 2^B \end{array}$$

The existence of such quantification is a considerable increase in our expressive power. Note, however, that it is obtained by using pullbacks over exponentials, which do not usually have a decidable equality. Hence, $\forall B.\chi$ will not, in general, have a complement.

3 Position functors

Position functors will be used to store data, a role dominated by the list functor in earlier work.

Consider a function $\chi : A \rightarrow B + 1$. By extensivity, we can pull χ back along the coproduct inclusions to obtain a coproduct decomposition of A

$$\begin{array}{ccccc} A_0 & \xrightarrow{m} & A & \longleftarrow & A_1 \\ \downarrow f & \lrcorner & \downarrow \chi & \lrcorner & \downarrow \\ B & \xrightarrow{\text{inl}} & B + 1 & \longleftarrow & 1 \\ & & \text{inr} & & \end{array}$$

as $A_0 + A_1$. Hence, we have a partial function $(m, f) : A \rightarrow B$ with a decidable domain A_0 . Define $A \rightarrow B$ to be $A \rightarrow B + 1$. The right inclusion represents the undefined value. The expression $b \uparrow$ is an abbreviation for the boolean test $b = \text{inr}$.

Define $\text{rest}_{A,B} : (A \rightarrow B) \rightarrow (A \rightarrow 1) \rightarrow (A \rightarrow B)$ to be the function that restricts the domain of functions $A \rightarrow B$. That is,

$$\text{rest } f \ p \ x = \text{if } p \ x \text{ then } f \ x \text{ else inr} .$$

where $p \ x$ is here thought of as a boolean.

For any object P of \mathcal{D} , define the *position functor* $P \rightarrow : \mathcal{D} \rightarrow \mathcal{D}$ to be the functor which maps an object X to $P \rightarrow X$ and has the obvious action on morphisms. P is called its *object of positions*.

Lemma 3.1 *Position functors are shapely.*

Proof They are the composite of the shapely functors $(-) + 1$ and $P \rightarrow (-)$. \square

Theorem 3.2 *Every position functor on a cartesian closed locus has a final co-algebra.*

Proof The final co-algebra for $P \rightarrow$ consists of trees with countable paths such that the branches from each node are labelled by distinct elements of P , though not all elements of P need be used. A finite path from the root of such a tree is given by a finite list of positions. Thus the tree can be described by a predicate $f : LP \rightarrow 2$ which determines whether a given path defines a node of the tree, or not. Indeed, any such predicate f will define a tree, provided that the empty path defines a node, i.e. $f \ \text{nil} = \text{true}$ and that if a long path defines a node, then so does any shorter path, i.e. $f \ (s@[p])$ implies $f \ s$ (where $@$ is append).

Thus, the desired collection of predicates is given by a pullback:

$$\begin{array}{ccc} P_1 & \xrightarrow{\iota} & LP \rightarrow 2 \\ \downarrow & \lrcorner & \downarrow \chi \\ 1 & \xrightarrow{\text{all}} & LP \rightarrow 2 \end{array}$$

where

$$\begin{aligned} \chi \ f \ \text{nil} &= f \ \text{nil} \\ \chi \ f \ (s@[p]) &= f \ (s@[p]) \rightarrow f \ s . \end{aligned}$$

(The final \rightarrow is the boolean implication.)

The co-algebra action $q : P_1 \rightarrow P \rightarrow P_1$ is the inverse of the node constructor for the trees. It is given by factoring $q' : P_1 \rightarrow P \rightarrow (LP \rightarrow 2)$ through ι where

$$q' \ g \ p \ s = (\iota \ g) \ (\text{cons } p \ s) .$$

Now consider an arbitrary co-algebra $a : X \rightarrow P \rightarrow X$. If $g : X \rightarrow P_1$ is a co-algebra morphism then $f = \iota \circ g$ must satisfy

$$\begin{aligned} f \ x \ \text{nil} &= \text{true} \\ f \ x \ (\text{cons } p \ s) &= q' \ (g \ x) \ p \ s \\ &= \iota((P \rightarrow g)(a \ x)) \ p \ s \\ &= f \ (a \ x \ p) \ s . \end{aligned}$$

This tail-recursive construction of f defines it uniquely. Since ι is a monomorphism, this shows that there is exactly one co-algebra homomorphism $X \rightarrow P_1$. \square

The status of initial algebras for $P \rightarrow$ is not yet clear, unless P is finite [5]. Then $P \rightarrow$ is cartesian over lists and its initial algebra can be constructed using existing techniques [7]. Consequently, we will find it necessary below to assume their existence.

4 Unary data functors

Data functors are defined using cartesian transformations into position functors, which are used to store data. This strictly generalises the earlier work based on data storage in lists [7], since the data functors are also closed under exponentiation, and so can be used to model higher-order types.

A *unary data functor* (F, P, data) is a functor $F : \mathcal{D} \rightarrow \mathcal{D}$ equipped with a cartesian transformation $\text{data} : F \Rightarrow P \rightarrow$ into a position functor. P is called its *object of positions*.

Various list functors arise as unary data functors with the natural numbers object N as object of positions. For example

$$\text{data} : LA \rightarrow N \rightarrow A$$

maps a list to the function which maps n to the n th entry in the list. If n exceeds the length of the list then the function is not defined. Infinite lists, lists of arbitrary length, etc. also appear in this way. Also, matrices have positions given by pairs of numbers; general arrays are handled similarly.

One can view FA as representing a dependent sum:

$$\Sigma_{s:F1} A^{a(s)}$$

where $a(s)$ is the domain of the partial function $\text{data}_1(s)$. When the base category is **Sets** then the unary data functors are exactly the *familially representable* functors [1] since the family of shapes yields a family of representable functors.

Since cartesian transformations compose, it follows that any functor cartesian over lists is also a unary data functor with object of positions N .

Lemma 4.1 *The unary data functors are closed under composition and identities, and under exponentiation by an object.*

Proof If F and G are unary data functors with objects of positions P and Q then GF is a unary data functor with object of positions $Q \times P$. The identity functor has the terminal object as object of positions.

If $\text{data} : F \Rightarrow P \rightarrow$ is a data functor and X is an object of \mathcal{D} then the functor F^X given by $F^X A = (FA)^X$ (and the obvious action on morphisms) is also a data functor with object of positions $X \times P$. This is because exponentials preserve pullbacks and

$$(P \rightarrow A)^X \cong (X \times P) \rightarrow A.$$

\square

By uncurrying **data** we obtain a natural transformation

$$\text{entry} : FA \times P \rightarrow A + 1$$

which picks out the datum of FA at the position given by P . Also, we can define an operation

$$\text{mark} : F1 \rightarrow FP$$

which marks the positions in a shape with their names. It is defined by:

$$\begin{array}{ccc} F1 & \xrightarrow{\text{data}} & P \rightarrow 1 \\ & \searrow \text{mark} & \searrow \text{rest id} \\ & & FP \xrightarrow{\text{data}} P \rightarrow P \\ & \searrow \text{id} & \downarrow \# \\ & & F1 \xrightarrow{\text{data}} P \rightarrow 1 \end{array}$$

Lemma 4.2 *If (F, P, data) is a data functor then **entry** and **mark** satisfy the following equations:*

1. $\# \circ \text{mark} = \text{id}$
2. $\text{entry} \circ (\text{mark} \times \text{id}) =$
if **entry** then π' else **inr**
3. $F\text{eval} \circ \tau' \circ \langle \text{data}, \text{mark} \circ \# \rangle = \text{id}$

(where $\tau'_{A,B} : A \times FB \rightarrow F(A \times B)$ is the dual of the strength τ).

Proof By straightforward reasoning using pullbacks. \square

Consider a natural transformation $\alpha : P \rightarrow \Rightarrow Q \rightarrow$ between position functors. If α is natural with respect to *partial* morphisms then the Yoneda Lemma implies that it is given by a partial morphism $Q \rightarrow P$. However, we only require that α be natural with respect to total morphisms, which complicates the picture slightly.

Theorem 4.3 *Let P and Q be objects of \mathcal{D} . The corresponding object of transformations from $P \rightarrow$ to $Q \rightarrow$ is given by the following pullback:*

$$\begin{array}{ccc} P \Rightarrow Q & \longrightarrow & (P \rightarrow 1) \rightarrow Q \rightarrow P \\ \downarrow & \lrcorner & \downarrow \chi \\ 1 & \longrightarrow & (P \rightarrow 1) \rightarrow (Q \rightarrow 1) \\ & \text{all} & \end{array}$$

where

$$\chi f m q = \uparrow(m(f m q)) \rightarrow \uparrow(f m q) .$$

Further, every morphism $(P \rightarrow 1) \rightarrow Q \rightarrow P$ determines a natural transformation $P \rightarrow \Rightarrow Q \rightarrow$.

Proof The way to interpret the pullback is that every $\alpha : P \rightarrow \Rightarrow Q \rightarrow$ is determined by a morphism $\alpha' : (P \rightarrow 1) \rightarrow (Q \rightarrow P)$ (called its *finder*). For each possible domain of a partial morphism out of P (represented by $P \rightarrow 1$) it picks out a partial morphism $Q \rightarrow P$, just as in the Yoneda Lemma. Conversely, such a morphism α' determines a natural transformation if, for every decidable subobject P_0 of P (characterised by a morphism $m : P \rightarrow 1$) the range of $\alpha' m : Q \rightarrow P$ lies within P_0 .

Given α as above, define

$$\phi(\alpha) m = \alpha_P(\text{rest id}_P m) .$$

It is an easy exercise to see that $\chi(\phi(\alpha)) = \text{all}$. Conversely, given any morphism $\alpha' : (P \rightarrow 1) \rightarrow (Q \rightarrow P)$ we can define a natural transformation $\psi(\alpha')$ from $P \rightarrow$ to $Q \rightarrow$ by

$$\psi(\alpha')_A = \text{comp} \circ \langle \text{id}, \alpha' \circ \# \rangle .$$

where comp is the composition of partial functions. Note, however, that we can replace α' by

$$\alpha'' m = (\text{rest id}_P m) \circ (\alpha' m)$$

since it takes the same value under ψ . Also, α'' satisfies the test χ . Hence, $P \rightarrow \Rightarrow Q \rightarrow$ supplies canonical representatives for equivalence classes of morphisms that yield the same transformation.

Now,

$$\begin{aligned} \phi(\psi(\alpha')) m &= \psi(\alpha')(\text{rest id}_P m) \\ &= \text{comp} \circ \langle \text{id}, \alpha' \circ \# \rangle (\text{rest id}_P m) \\ &= (\text{rest id}_P m) \circ (\alpha' m) \\ &= \alpha' m \text{ since } \chi(\alpha') = \text{all} . \end{aligned}$$

Conversely, define

$$\begin{aligned} \beta &= \psi(\phi(\alpha)) \\ &= \text{comp} \circ \langle \text{id}, \phi(\alpha) \circ \# \rangle \\ &= \text{comp} \circ \langle \text{id}, \alpha_P(\text{rest id}_P) \rangle . \end{aligned}$$

We must prove that $\beta = \alpha$. First, consider a “global element” $g : 1 \rightarrow P \rightarrow A$ of $P \rightarrow A$. It is given by a subobject $m : P_0 \rightarrow P$ and a morphism $f : P_0 \rightarrow A$. Thus,

$$\begin{aligned} \beta g &= g \circ (\alpha_P(m, m)) \\ &= (Q \rightarrow g)((Q \rightarrow m)(\alpha_{P_0}(m, \text{id}))) \\ &= (Q \rightarrow f)(\alpha_{P_0}(m, \text{id})) \\ &= \alpha_A(m, f) . \end{aligned}$$

The second and last equations are by the naturality of α with respect to total morphisms. The middle equation follows since $(m, \text{id}_A) \circ (\text{id}_P, m) = \text{id} : P_0 \rightarrow P_0$.

This argument completes the proof for categories like **Sets** and **Pos** _{ω} in which morphisms are determined by their action on global elements. In the general case we must consider an arbitrary “element” $g : X \rightarrow P \rightarrow A$ of $P \rightarrow A$. But by the universal property of the partial function object, this corresponds to a global element of $X \times P \rightarrow A$, which reduces the problem to the earlier form. \square

Corollary 4.4 Every natural transformation between position functors has a canonical strength.

Proof The transformations $P \rightarrow \Rightarrow Q \rightarrow$ induced by morphisms $(P \rightarrow 1) \rightarrow Q \rightarrow P$ are automatically strong. \square

It remains to consider initial algebras and final co-algebras.

Theorem 4.5 Unary data functors have final co-algebras.

Proof Let $\text{data} : F \Rightarrow P \rightarrow$ be a unary data functor. Its final co-algebra F_1 is given by trees whose nodes have the property that they are labelled by some $s \in S = F1$ and have a branch for each position in the domain of $\text{data}(s) : P \rightarrow 1$. Paths in such trees are given by a list of positions. To each such can be assigned a list of the node labels encountered along the way. In other words, the data to describe the tree is given by a function $f : LP \rightarrow LS$. If the path ps of length m describes a node, then the resulting list $f ps$ will have length $n = m + 1$ (the root shape plus one for each branch of the path). Otherwise n will be less than $m + 1$. Also, for each $0 \leq k < n$ the next branch must be in the arity of the node shape, while at n either this condition fails or $n = m + 1$. That is, if $p_k = \text{entry } k \text{ } ps$ and $s_k = \text{entry } k \text{ } (f ps)$ then:

$$\begin{aligned} \text{data } s_k p_k &= \text{true for all } 0 \leq k < n \\ \text{data } s_n p_n &= m = n + 1 . \end{aligned}$$

Finally, if qs is another path such that $\text{take } n \text{ } qs = \text{take } n \text{ } ps$ then $f ps = f qs$. These equations can be represented by a test

$$\chi : (LP \rightarrow LS) \rightarrow LP \times LP \times N \rightarrow 2$$

so that the final co-algebra is given by the pullback:

$$\begin{array}{ccc} F_1 & \xrightarrow{\quad} & LP \rightarrow LS \\ \downarrow & \lrcorner & \downarrow \chi \\ 1 & \xrightarrow{\quad \text{all} \quad} & LP \times LP \times N \rightarrow 2 \end{array}$$

The proof of finality is similar to that for Theorem 3.2 above. \square

Unfortunately, it is not yet clear whether unary data functors have initial algebras in an arbitrary cartesian closed locus. One would expect to define them as sub-algebras of the corresponding final co-algebras, but the problem is still open. We will return to this issue at the end of the section.

Now let us consider natural transformations between position functors. In general, there may be a proper class of natural transformations between a pair of functors from **Sets** to **Sets** (e.g., from the covariant power set functor to a constant functor). This problem does not arise for position functors, however. In order to make this precise, we must specify the universal property of the object of natural transformations. Let $F, G : \mathcal{D}^m \rightarrow \mathcal{D}^p$ be functors and let X be an object of \mathcal{D}^p . Define $X \times F$ to be the product of the functor which is constantly X with F . Then an object $F \Rightarrow G$ of natural transformations from F to G has the universal property that

$$\frac{X \times F \Rightarrow G}{X \rightarrow F \Rightarrow G}$$

natural transformations from $X \times F$ to G are in natural bijection with morphisms from X to $F \Rightarrow G$.

Given F and G as above, and an object A of \mathcal{D}^n we can instantiate transformations at A by

$$\text{inst}_A : (F \Rightarrow G) \rightarrow FA \rightarrow GA$$

which is obtained by applying the bijection above to the identity on $F \Rightarrow G$ and currying.

5 General data functors

In general, a data functor may require more than one kind of data. For example, the leaves and nodes of a tree may take data of different types.

A functor $F : \mathcal{D}^n \rightarrow \mathcal{D}$ is a *data functor* if there is an object $P = (P_i)$ of \mathcal{D}^n (called the *position objects*) and a cartesian transformation

$$\text{data} : F(A_i) \Rightarrow \prod_i P_i \rightarrow A_i .$$

Most generally, a functor $F : \mathcal{D}^n \rightarrow \mathcal{D}^m$ is a *data functor* if each of its m projections to \mathcal{D} are.

Lemma 5.1 *Data functors are closed under composition. The binary product and sum functors are data functors. Hence, if $F, G : \mathcal{D}^n \rightarrow \mathcal{D}$ are data functors then so are $F \times G$ and $F + G$.*

Proof The argument for composition is a simple generalisation of that for unary data functors. The binary product and sum functors both have the same objects of positions, namely $(1, 1)$ but the data is different in each case. \square

Define $\text{Data}(\mathcal{D})$ to be the sub-2-category of **Cat** consisting of finite powers of \mathcal{D} , data functors and all natural transformations between them.

Just as for position functors, we will establish that there is an object of natural transformations, with all its attendant consequences.

Theorem 5.2 *For each pair of data functors $(F, \text{data}), (G, \text{data}) : \mathcal{D} \rightarrow \mathcal{D}$ the object $F \Rightarrow G$ exists.*

Proof The object of transformations is given by the pullbacks

$$\begin{array}{ccc} F \Rightarrow G & \longrightarrow & F1 \rightarrow GP \\ \downarrow & \lrcorner & \downarrow \text{curry}(\chi) \\ F1 \rightarrow G1 & \xrightarrow{F1 \rightarrow G \text{true}} & F1 \rightarrow G2 \end{array}$$

where $\chi = \text{Gentry} \circ \tau' \circ \langle \text{snd}, \text{eval} \rangle$.

To see this, observe that every $\alpha' : F1 \rightarrow GP$ defines a natural transformation $FA \rightarrow G(A+1)$ (i.e. a partial natural transformation from F to G) by

$$\psi(\alpha') = \text{Gentry} \circ \tau' \circ \langle \text{id}, \alpha' \circ \# \rangle .$$

It is total iff α' passes the test given by χ . Conversely, given $\alpha : F \Rightarrow G$ then define

$$\phi(\alpha) = \alpha_P \circ \text{mark} .$$

We must prove that ϕ and ψ are inverses. The structure of the proof is the same as that for Theorem 4.3.

One direction is easy. Starting from $\alpha' : F1 \rightarrow GP$ we obtain

$$\begin{aligned} & (\text{Gentry} \circ \tau' \circ \langle \text{id}, \alpha' \circ \# \rangle) \circ \text{mark} \\ &= \text{Gentry} \circ \tau' \circ \langle \text{mark}, \alpha' \rangle \\ &= G(\text{entry} \circ (\text{mark} \times \text{id})) \circ \tau' \circ \langle \text{id}, \alpha' \rangle \\ &= G(\text{if entry then } \pi' \text{ else inr}) \circ \tau' \circ \langle \text{id}, \alpha' \rangle \\ &= G\pi' \circ \tau' \circ \langle \text{id}, \alpha' \rangle \\ &= \alpha' . \end{aligned}$$

The first and third equations hold by Lemma 4.2. The fourth holds because α' satisfies the test χ . The others are standard manipulations of strengths.

Now let us consider a natural transformation $\alpha : F \Rightarrow G$. We must prove that $\beta = \psi(\phi(\alpha))$ is α . Since they clearly have the same effect on shapes, it suffices to prove that $\text{data} \circ \alpha = \text{data} \circ \beta$ so we may as well assume that $G = Q \rightarrow$. In this case, we have: $\text{Gentry} \circ \tau' = \text{comp} \circ (\text{data} \times \text{id})$ as morphisms $FA \times (Q \rightarrow P) \rightarrow Q \rightarrow A$. Hence,

$$\beta_A = \text{comp} \circ \langle \alpha_P \circ \text{mark} \circ \#, \text{data} \rangle .$$

Now, consider a morphism $x : 1 \rightarrow FA$. It is determined by $s = \# \circ x : F1$ and $\text{data} \circ x = g : P \rightarrow A$. Further, $g = (m, f)$ where $m : P_0 \rightarrow P$ is a

decidable subobject of P and $f : P_0 \rightarrow A$ is total. Also, $\text{mark} \circ s = Fm \circ h$ for some $h : 1 \rightarrow FP_0$. We have

$$\begin{aligned}
\beta_A \circ x &= g \circ \alpha_P \circ \text{mark} \circ s \\
&= g \circ \alpha_P \circ Fm \circ h \\
&= g \circ m \circ \alpha_{P_0} \circ h \\
&= f \circ \alpha_{P_0} \circ h \\
&= \alpha_A \circ Ff \circ h \\
&= \alpha_A \circ Fg \circ \text{mark} \circ s \\
&= \alpha_A \circ x .
\end{aligned}$$

These equations exploit the naturality of α , that $(m, \text{id}) \circ m = \text{id}$ for a monomorphism m , and Lemma 4.2.3.

The general case requires us to consider a morphism $X \rightarrow FA$ (and then apply the Yoneda lemma). This corresponds to a morphism $1 \rightarrow F^X A$ which is a data functor with object of positions $X \times P$. It follows by the previous argument that $\alpha_A^X = \beta_A^X$ and hence that $\alpha = \beta$.

Note that the proof uses the fact that G is a data functor. It suffices to assume that G is strong, provided that we restrict our attention to strong natural transformations. However, it is not yet known if this weakening can be made in the general case. \square

Corollary 5.3 *Every natural transformation between data functors has a canonical strength.*

Proof The transformations $\alpha : F \Rightarrow G$ induced by morphisms $F1 \rightarrow GP$ are automatically strong. Thus, the assumption of strength is no longer required as a hypothesis. \square

Given data functors $(F, \text{data}) : \mathcal{D}^n \rightarrow \mathcal{D}^p$ and $(G, \text{data}) : \mathcal{D}^{m+n} \rightarrow \mathcal{D}^p$ define the functor $(F \Rightarrow G)$ which maps an object A to $F \Rightarrow G(A, -)$ and has the corresponding action on morphisms.

Theorem 5.4 *For each pair of data functors F and G as above, the functor $F \Rightarrow G$ has the structure of a data functor.*

Proof Without loss of generality we may assume that $m = n = p = 1$. If the objects of positions for G are Q and R then the required data for $F \Rightarrow G$ is given by

$$\begin{aligned}
(F \Rightarrow G)A &\longrightarrow F1 \rightarrow G(A, P) \\
&\longrightarrow F1 \rightarrow (Q \rightarrow A) \\
&\cong (F1 \times Q) \rightarrow A
\end{aligned}$$

a composite of cartesian transformations. \square

Theorem 5.5 *Let $F : \mathcal{D}^n \times \mathcal{D} \rightarrow \mathcal{D}$ be a data functor. The canonical functor $G : \mathcal{D}^n \rightarrow \mathcal{D}$ which maps A to the final co-algebra for $F(A, -)$ is itself a data functor.*

Proof Let $P = (P_i) \in \mathcal{D}^n$ and $Q \in \mathcal{D}$ be the objects of positions for F . For each $A = (A_i) \in \mathcal{D}^n$ define GA to be the final co-algebra for $F(A, -)$. G is a position functor whose objects of positions are all the $LQ \times P_i$. The P_i is used to locate the data at the node specified by the path in LQ . \square

As mentioned above, the status of initial algebras for position and data functors is not yet resolved. Until then (at least) we must assume their existence.

Definition 5.6 *A data category is a cartesian closed locus whose data functors all have initial algebras.*

Examples include **Sets**, **Pos** _{ω} and **Eff**.

Theorem 5.7 *In a data category the initial algebras for data functors are themselves data functors.*

Proof The algebra homomorphisms from the initial algebras to the final co-algebras form cartesian transformations, since any commuting square containing a parallel pair of isomorphisms is a pull-back. Hence the initial algebras themselves form data functors, over the same position functors as the final co-algebras. \square

6 Semantics of the covariant types

The covariant types are given by:

$$T ::= X \mid T \Rightarrow T \mid T \times T \mid T + T \mid \mu_X T .$$

These constructions represent transformation types, products, sums, and recursive types. Transformations are parametric functions. This parametricity is reflected in the following fact about transformation types: the type variables of S are all bound in $S \Rightarrow T$. Consequently, all free type variables appear in strictly positive positions, and so types can be modelled by covariant functors. Type contexts Δ are lists (without repetition) of type variables. Details of type judgements $\Delta \vdash T$ and the term calculus can be found in [6].

The categorical semantics of a derived type judgement $\Delta \vdash T$ is a data functor

$$[[T]] : \mathcal{D}^n \rightarrow \mathcal{D} .$$

where n is the length of Δ . Most of the interpretations are standard. Enlarging the context is modelled by projection, swapping of type variables in the type context is modelled by swapping arguments in the functor. The axiom is modelled by the identity functor. For the rest we have (assuming a given type context):

$$\begin{aligned}
[[S \Rightarrow T]] &= [[S]] \Rightarrow [[T]] \\
[[S \times T]] &= [[S]] \times [[T]] \\
[[S + T]] &= [[S]] + [[T]] \\
[[\mu_X.T]] &= [[T]]^\dagger .
\end{aligned}$$

where $\llbracket S \rrbracket \Rightarrow \llbracket T \rrbracket$ denotes the functor of strong natural transformations, defined in Section 5 and $\llbracket T \rrbracket^\dagger$ is the initial algebra with respect to the argument denoted by X .

The covariant terms can all be modelled by natural transformations between the interpretations of the types. Hence, we have sketched the proof of the following:

Theorem 6.1 *The covariant type system can be modelled in any data category, e.g. Sets.* \square

7 Conclusions

We have established that the data functors are closed under many of the fundamental constructions of category theory and type theory. In particular, a generalisation of the Yoneda lemma shows that every natural transformation between data functors has a uniform description. This allows us to model higher-order types using data functors, and so construct simple, set-theoretic models of a polymorphic type system.

8 Acknowledgements

I would like to thank J. Adamek, P. Dybjer, E. Moggi, T. Streicher, P. Wadler and the members of the Shape project for their suggestions and comments.

References

- [1] A. Carboni and P.T. Johnstone. Connected limits, familial representability and artin glueing. To appear in *mcs*, 1995.
- [2] A. Carboni, S. Lack and R.F.C. Walters. Introduction to extensive and distributive categories. *Journal of Pure and Applied Algebra*, Volume 84, pages 145–158, 1993.
- [3] J.R.B. Cockett. List-arithmetic distributive categories: locoi. *Journal of Pure and Applied Algebra*, Volume 66, pages 1–29, 1990.
- [4] J.M.E. Hyland. The effective topos. In A.S. Troelstra and D. van Dalen (editors), *The L.E.J. Brouwer Centenary Symposium*. North Holland, 1982.
- [5] C.B. Jay. Finite objects in a locos. *Mathematical Structures in Computer Science*, 1994. to appear.
- [6] C.B. Jay. Covariant types. Technical report, University of Technology, Sydney, 1995.
- [7] C.B. Jay. A semantics for shape. *Science of Computer Programming*, 1995. in press.
- [8] A. Pitts. Polymorphism is set theoretic, constructively. In *Proceedings of the Conference on Category Theory and Computer Science, Edinburgh, UK, Sept. 1987*, Volume 283 of *Lecture Notes in Computer Science*. Springer Verlag, 1987.
- [9] J. Reynolds. Polymorphism is not set-theoretic. In Kahn, McQueen and Plotkin (editors), *Symposium on semantics of data types*, Volume 173 of *Lecture Notes in Computer Science*. Springer Verlag, 1984.
- [10] M. Smith and G. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal of Computing*, Volume 11, 1982.