To appear in Canadian Mathematical Society Proceedings (Category Theory 1991, Montreal).

Strong Categorical Datatypes I

J. Robin B. Cockett
Dept. of Computer Science
University of Calgary
Calgary, Alberta T2N 1N4, Canada
robin@cpsc.ucalgary.ca

Dwight Spencer
Dept. of Computer Science and Engineering
Oregon Graduate Institute
Beaverton, Oregon 97006-1999
dwights@cse.ogi.edu

April 22, 1992

Abstract

An endofunctor of a cartesian closed category is often called strong if it is enriched over the exponential. Equivalently this strength can be provided as a natural transformation $\theta_{A,X} : F(A)\times X \longrightarrow F(A\times X)$ satisfying some elementary coherence conditions. This latter formulation does not require exponentials, relies only on the presence of an **X**–action over an **X**–strong category, and thereby provides a first–order viewpoint of strength.

The 2-category of **X**–strong categories is not finitely complete. It particularly lacks many standard constructions including the Eilenberg–Moore construction. Thankfully, the suggestion — attributed to Plotkin by Moggi — that strength can also be equivalently framed in terms of fibrations using projections to **X**–objects as display maps can be fully realized. The equivalence can be portrayed as an embedding of **X**–strong categories into the 2–category of **X**–indexed categories or split fibrations over **X**. This embedding can be used to give a coherent declarative implementation of parametrized datatypes-with-strength and their constructors/destructors in the style of Hagino.

The initiality and finality conditions associated with the strong datatype construction in the split fibration setting generate a strongly normalizing weak-head categorical combinator reduction system. The reduction rules permit both "eager" usage of initial datatypes and "lazy" usage of final datatypes to achieve reasonable computing strength. An **X**-action over a strong datatype can be computed in terms of programmed state transformations over its component datatypes. These reduction rules form the abstract engine for computation within the **Charity** categorical programming system.

1 Introduction

In many mathematical instances ordinary functors and natural transformations do not provide sufficient structure to successfully capture a particular phenomenon. A solution usually entails the enrichment of type-forming functors and natural transformations. When the phenomena are programs, we recognize a critical structure-capture requirement: the calculation of a functor should be doable at the same level as the calculation of maps. This has been reinforced by Moggi [12, 13, 14] by his demonstration of the direct relation of strong monads to the semantics of computation.

For cartesian closed categories, this causes us to focus on the obvious enrichment over the internal exponentiation. In this setting an endofunctor $F : \mathbf{X} \longrightarrow \mathbf{X}$ is enriched if it is represented as a natural transformation

$$f_{A,B} : A \Rightarrow B \longrightarrow F(A) \Rightarrow F(B)$$

which behaves correctly with respect to the composition as internally given. This natural transformation provides the means for building a "strength" for the ordinary functor via the following equivalence:

$$\frac{X \xrightarrow{curry(i_{A\times X})} A \Rightarrow (A \times X) \xrightarrow{f_{A,A\times X}} F(A) \Rightarrow F(A \times X)}{F(A) \times X \xrightarrow{\theta_{A,X}} F(A \times X)}.$$

Conversely, one may formally define the notion of a functor with strength and show that the notion also implies enrichment by

$$\frac{F(A) \times (A \Rightarrow B) \xrightarrow{\theta_{A,A\Rightarrow B}} F(A \times (A \Rightarrow B)) \xrightarrow{F(ev)} F(B)}{A \Rightarrow B \xrightarrow{f_{A,B}} F(A) \Rightarrow F(B)}.$$

These equivalences make redundant the requirement that exponentiation actually be present. We therefore simply consider functors with a given strength. This paper fully develops the consequences of following such a computational direction. Section 2 provides a definition of $\mathbf{X}$–strong categories which serves, we believe, as the appropriate categorical foundation of strength. The definition departs from previous usage, i.e. as enrichment, by adopting the generality of a non-closed setting while concentrating on the product as the source and effect of a tensor action.

Datatypes are built with limits and colimits, yet the resulting 2-category of $\mathbf{X}$–strong categories disappoints us by lacking arbitrary finite limits. In particular, the category of F-algebras of an endofunctor cannot be constructed. Thus there is no immediate guidance for obtaining the "correct" notion of a free monad or, more generally, a datatype. For example, the "correct" notion of a free monad on an endofunctor F is that it is the monad generated by the left adjoint to the underlying functor from the category of F–algebras.

Section 3 shows how our purview of strength can be moved to a complete setting. There we define and explain an embedding, attributed in [12, 13] to Gordon Plotkin, of the 2-category of $\mathbf{X}$-strong categories into the 2-category of split fibrations over $\mathbf{X}$. This latter category, by virtue of its completeness, has the algebraic constructions and the guidance we seek.

Section 4 explains the limit constructions of the category of algebras for a strong initial datatype and the category of coalgebras for a strong final datatype within fibrations. The limits are unraveled into the universal initiality and finality diagrams from which all our combinators and computation rules for strong datatypes are derived, as detailed in Section 5. The fibration-based datatype notion we propose is correct in terms of its context parametrization of both constructors and destructors, a feature neglected in other approaches but vital to practical programming practices.

2 X–strength

This section's purpose is to cast a foundational description of strength and then mold it into an elementary description that will be more widely recognized. The molding will be performed quite independently of cartesian closedness.

The starting point in generality is the notion of a **V**–tensored category **Y**. Here **V** is a category with an arbitrary tensor product, as described by Anders Kock [9], and **Y** is a category with a **V**–action on it given by a tensor preserving functor from **V** into *Endo*(**Y**). Our interest narrows to the special case of **V**'s tensor product being the (cartesian) product. We will hereafter designate such a category **V** as **X**. Furthermore, we demand that **Y** have its own tensor, viz. its product, and that the action, now an **X**–action, be representable by this tensor. The term **X**–*strong* will be used to describe such a category **Y**. It should be apparent that **X**–strong categories resemble K-algebras (for K a field) more closely than K-modules.

If **X** is a cartesian category (i.e. has a binary product and a terminal object) an **X**-*strong category* **Y** is a cartesian category **Y** with a bifunctor, called an **X**–*action*,

$$\_ \oslash \_ \quad : \quad \mathbf{Y} \times \mathbf{X} \longrightarrow \mathbf{Y}$$

that is compatible with the coherent natural isomorphisms

- *ass* : $Y \oslash (X_1 \times X_2) \longrightarrow (Y \oslash X_1) \oslash X_2$,
- *rid* : $(Y \oslash 1) \longrightarrow Y$,
- *rep* : $Y \oslash X \longrightarrow Y \times (1 \oslash X)$.

The first two natural isomorphisms are the standard ones and must satisfy the coherence conditions expected for any monoidal action. The last allows the action to be represented by the product of the category **Y** on which **X** acts.

Because these categories are cartesian we can simplify the general commuting requirements to the following four equations:

- $\mathit{ass}\,;\mathit{rid} = \mathit{id} \oslash p_0 \quad : \quad Y \oslash (X \times 1) \longrightarrow Y \oslash X$,
- $\mathit{rid} = \mathit{rep}\,; p_0 \quad : \quad (Y \oslash X) \oslash 1 \longrightarrow Y \oslash X$,
- $\mathit{ass}\,;(\mathit{rid} \oslash \mathit{id}) = \mathit{id} \oslash p_1 \quad : \quad Y \oslash (1 \times X) \longrightarrow Y \oslash X$,
- $\mathit{rid} \oslash \mathit{id} = \mathit{rep}\,; p_1 \quad : \quad (1 \oslash 1) \oslash X \longrightarrow 1 \oslash X$.

Notice that they are aimed at ensuring that the map *rid* behaves correctly.

Proposition 2.1 *A cartesian category* **Y** *with an* **X**–*action is* **X**–*strong if and only if there is a cartesian functor*

$$J : \mathbf{X} \longrightarrow \mathbf{Y}$$

and a natural isomorphism $r_{Y,X} : Y \oslash X \longrightarrow Y \times J(X)$.

Proof. If we are given J and the natural isomorphism r then by setting

$$rid = r_{Y,1} \,;\, p_0$$

$$ass = r_{Y,X_1\times X_2} \,;\, \langle\langle p_0, p_1 \,;\, J(p_0)\rangle, p_1 \,;\, J(p_1)\rangle \,;\, r^{-1}_{Y\times J(X_1),X_2} \,;\, r^{-1}_{Y,X_1} \oslash id$$

$$rep = r_{Y,X} \,;\, id \times \langle !, id\rangle \,;\, id \times r^{-1}_{1,X}$$

the (simplified) coherence requirements are straightforward to verify.

Conversely, given an $\mathbf{X}$–action we may choose $J(X) = 1 \oslash X$. Clearly J is a functor and r is definable as *rep*. It remains to show J is cartesian. Because $rid : 1 \oslash 1 \longrightarrow 1$ is an isomorphism it follows that $1 \oslash 1$ is final. To show that products are preserved we consider

$$1 \oslash A \xleftarrow{1 \oslash p_0} 1 \oslash (A \times B) \xrightarrow{1 \oslash p_1} 1 \oslash B$$

and show that it is naturally isomorphic to

$$1 \oslash A \xleftarrow{p_0} 1 \oslash A \times 1 \oslash B \xrightarrow{p_1} 1 \oslash B$$

This is accomplished with the following two diagrams:

$$\begin{tikzcd}
1 \oslash (X \times Y) \arrow[r, "id \oslash (id \times !)"'] \arrow[d, "ass"'] & 1 \oslash (X \times 1) \arrow[d, "ass"'] \arrow[dr, "id \oslash p_0"] & \\
(1 \oslash X) \oslash Y \arrow[r, "(id \oslash id) \oslash !"'] \arrow[d, "rep"'] & (1 \oslash X) \oslash 1 \arrow[r, "rid"'] \arrow[d, "rep"'] & 1 \oslash X \\
1 \oslash X \times 1 \oslash Y \arrow[r, "id \times id \oslash !"'] & 1 \oslash X \times 1 \oslash 1 \arrow[ur, "p_0"'] &
\end{tikzcd}$$

and

$$\begin{tikzcd}
1 \oslash (X \times Y) \arrow[r, "1 \oslash (! \times id)"'] \arrow[d, "ass"'] & 1 \oslash (1 \times Y) \arrow[d, "ass"'] \arrow[dr, "id \oslash p_1"] & \\
(1 \oslash X) \oslash Y \arrow[r, "(id \oslash !) \oslash i"'] \arrow[d, "rep"'] & (1 \oslash 1) \oslash Y \arrow[r, "rid \oslash i"'] \arrow[d, "rep"'] & 1 \oslash Y \\
1 \oslash X \times 1 \oslash Y \arrow[r, "id \oslash ! \times i"'] & 1 \oslash 1 \times 1 \oslash Y \arrow[ur, "p_1"'] &
\end{tikzcd}$$

in which the four identities are used in the triangles. $\square$

All the $\mathbf{X}$–strong categories of interest can be regarded as having a cartesian embedding J of $\mathbf{X}$ into it. The $\mathbf{X}$–action is, up to isomorphism, simply given by the product with the image of the objects of $\mathbf{X}$ under this embedding. This observation motivates our dropping the action notation $A \oslash X$ in favor of $A \times J(X)$ or just $A \times X$.

$\mathbf{X}$–*strong functors*, or simply *strong functors*, are functors between $\mathbf{X}$–strong categories equipped with a natural transformation

$$\theta^F_{Y,X} : F(Y) \oslash X \longrightarrow F(Y \oslash X),$$

called an $\mathbf{X}$–*strength*, or simply a *strength*, such that

$$\begin{tikzcd}
(F(A) \oslash B) \oslash C \arrow[d, "\theta^F_{A,B} \oslash id_C"'] & & F(A) \oslash (B \times C) \arrow[ll, "ass"'] \arrow[dd, "\theta^F_{A,B\times C}"] \\
F(A \oslash B) \oslash C \arrow[d, "\theta^F_{A \oslash B,C}"'] & & \\
F((A \oslash B) \oslash C) & & F(A \oslash (B \times C)) \arrow[ll, "F(ass)"]
\end{tikzcd}$$

and

$$\begin{tikzcd}
F(A) \oslash 1 \arrow[r, "\theta^F_{A,1}"] \arrow[dr, "rid"'] & F(A \oslash 1) \arrow[d, "F(rid)"] \\
& F(A)
\end{tikzcd}$$

commute.

A strong functor, as used by Moggi for cartesian closed categories, is simply an $\mathbf{X}$-strong endofunctor on $\mathbf{X}$ itself with the natural $\mathbf{X}$–action given by product. It is quite reasonable to discuss "strong functors" for non-cartesian closed categories: the list datatype on a distributive category [2] is a strong functor even though its setting is not closed.

An $\mathbf{X}$-*strong natural transformation* between strong functors $F, G : \mathbf{Y}_1 \longrightarrow \mathbf{Y}_2$ is a parameterized natural transformation

$$\alpha_{A,X} : F(A) \oslash X \longrightarrow G(A \oslash X) : \mathbf{Y}_1 \times \mathbf{X} \longrightarrow \mathbf{Y}_2$$

such that

$$\begin{tikzcd}
(F(A) \oslash X) \oslash Y \arrow[r, "\theta^F_{A,X} \oslash id_Y"] \arrow[d, "\alpha_{A,X} \oslash id_Y"'] & F(A \oslash X) \oslash Y \arrow[d, "\alpha_{A \oslash X,Y}"] \\
G(A \oslash X) \oslash Y \arrow[r, "\theta^G_{A \oslash X,Y}"'] & G((A \oslash X) \oslash Y)
\end{tikzcd}$$

commutes.

Let $\nu : X \xrightarrow{\cong} X_1 \times X_2$ be any decomposition of X into a product. Then the composition of two strong transformations $\alpha_{A,X}$ and $\beta_{A,X}$ is defined by the following sequence of maps:

$$\begin{tikzcd}
F(A) \oslash X \arrow[r, "id \oslash \nu"] \arrow[ddd, "(\alpha.\beta)_{A,X}"'] & F(A) \oslash (X_1 \times X_2) \arrow[r, "\text{ass}"] & (F(A) \oslash X_1) \oslash X_2 \arrow[d, "\alpha_{A,X_1} \oslash id"'] \\
 & & G(A \oslash X_1) \oslash X_2 \arrow[dd, "\beta_{A \oslash X_1, X_2}"'] \\
 & & \\
H(A \oslash X) & H(A \oslash (X_1 \times X_2)) \arrow[l, "H(id \oslash \nu^{-1})"'] & H((A \oslash X_1) \oslash X_2) \arrow[l, "H(\text{ass}^{-1})"']
\end{tikzcd}$$

The composition does not depend on the particular decomposition of X. That is, all decompositions result in the same parameterized composition. Furthermore, this composition straightforwardly commutes with the strengths in the required manner.

By noting that (1) the definition of strength implies $\theta^F_{A,1}$ is an isomorphism and (2) the definition of strong natural transformation trivially implies θ^F is strong and commutative with all other strong transformations, we see θ^F is actually the identity strong transformation on F under this composition. This establishes that **X**–strong categories, **X**–strong functors and **X**–strong transformations form a 2-category **Strong**(**X**).

This formulation of an **X**–strong transformation reveals the structural effect of making transformations strong. However, it fails to promote easy manipulation of these transformations. Notice that an ordinary transformation α gives rise to a related strong transformation provided

$$\begin{tikzcd}
F(A) \oslash X \arrow[r, "\theta^F_{A,X}"] \arrow[d, "\alpha_A \oslash id_X"'] & F(A \oslash X) \arrow[d, "\alpha_{A \oslash X}"] \\
G(A) \oslash X \arrow[r, "\theta^G_{A,X}"'] & G(A \oslash X)
\end{tikzcd}$$

commutes. The associated parameterized transformation is

$$\alpha_{A,X} \quad = \quad \alpha_A \oslash id_X \,;\theta^G_{A,X} \quad = \quad \theta^F_{A,X}\,;\alpha_{A\oslash X},$$

which is clearly strong. Conversely, a strong transformation is related to an ordinary natural transformation by

$$F(A) \xrightarrow{rid^{-1}} F(A)\oslash 1 \xrightarrow{\alpha_{A,1}} G(A\oslash 1) \xrightarrow{G(rid)} G(A)$$

and these associations are inverse. Thus we have reorganized our view of **X**–strong categories into a more accessible form:

Theorem 2.2 *The 2-category* $\mathbf{Strong}(\mathbf{X})$ *is isomorphic to the 2-category having as*

0-cells: *cartesian functors* $J : \mathbf{X} \longrightarrow \mathbf{Y}$,

1-cells: *pairs* (F, θ^F) *where* F *is a functor* $F : \mathbf{Y} \longrightarrow \mathbf{Y}'$ *and* θ^F *is a strength:*

$$\theta^F_{Y,X} : F(Y) \times J'(X) \longrightarrow F(Y \times J(X)),$$

satisfying

$$\begin{tikzcd}
(F(A)\times J'(X_1))\times J'(X_2) \arrow[d, "\theta^F_{A,X_1}\times id"'] & F(A)\times J'(X_1\times X_2) \arrow[l, "ass"'] \arrow[dd, "\theta^F_{A,X_1\times X_2}"] \\
F(A\times J(X_1))\times J'(X_2) \arrow[d, "\theta^F_{A\times J(X_1),X_2}"'] & \\
F((A\times J(X_1))\times J(X_2)) & F(A\times J(X_1\times X_2)) \arrow[l, "F(ass)"]
\end{tikzcd}$$

and

$$\begin{tikzcd}
F(A)\times J'(X) \arrow[rr, "\theta^F_{A,X}"] \arrow[dr, "p_0"'] & & F(A\times J(X)) \arrow[dl, "F(p_0)"] \\
& F(A) &
\end{tikzcd}$$

2-cells: *natural transformations* $\alpha : F \longrightarrow G$ *which satisfy :*

$$
\begin{tikzcd}
F(A) \times J'(X) \arrow[r, "\theta^F_{A,X}"] \arrow[d, "\alpha_A \times id"'] & F(A \times J(X)) \arrow[d, "\alpha_{A \times J(X)}"] \\
G(A) \times J'(X) \arrow[r, "\theta^G_{A,X}"'] & G(A \times J(X))
\end{tikzcd}
$$

Therefore an $\mathbf{X}$–strong category may be regarded without loss as a cartesian category with a cartesian functor from $\mathbf{X}$, an $\mathbf{X}$-strong functor as a functor with a strength, and an $\mathbf{X}$–strong transformation as a transformation which commutes with the strengths as shown above. We shall henceforth identify $\oslash$ and $\times$ and avoid the use of the relatively unfamiliar coherence maps for $\_ \oslash \_$.

3 Strength to Fibrations

A strong functor may equivalently be regarded as a morphism of fibrations [5] in which $\mathbf{X}$ is viewed as a category of contexts. Technically this is given by an embedding of the 2–category $\mathbf{Strong}(\mathbf{X})$ into the 2–category of split fibrations over $\mathbf{X}$, $\mathbf{SplFib}(\mathbf{X})$. The latter can, of course, be equivalently viewed as $\mathbf{X}$–indexed categories. This section derives the embedding; the next section attaches the importance of this embedding to our fabrication methodology for datatypes.

Given an $\mathbf{X}$–strong category $\mathbf{Y}$ we may form the category $P_{\mathbf{X}}(\mathbf{Y})$ to be be thought of as $\mathbf{Y}$–maps with context ¿from $\mathbf{X}$:

Objects: pairs of objects (Y, X) where $Y \in \mathbf{Y}$ and $X \in \mathbf{X}$,

Maps: pairs of maps $(f, x) : (Y, X) \longrightarrow (Y', X')$ where $f : Y \times X \longrightarrow Y'$ in $\mathbf{Y}$ and $x : X \longrightarrow X'$ in $\mathbf{X}$,

Identities: $(p_0, id) : (Y, X) \longrightarrow (Y, X)$,

Composition: $(f, h) ; (g, k) = (\langle f, p_1; h\rangle; g, h; k)$.

A map (f, h) uses h to propagate the context in the second component and f to compute new data values from both data values in Y and the available context:

$$(f, h) : (Y, X) \longrightarrow (Y', X') \ ; \ (y, x) \mapsto (f(y, x), h(x)).$$

Composing this map with (g, k) gives:

$$(f, h) ; (g, k) : (Y, X) \longrightarrow (Y'', X'') \ ; \ (y, x) \mapsto (g(f(y, x), h(x)), k(h(x))).$$

Thus g receives a propagated context and the computed value. This can be viewed as the Kleisli composition for the comonad of the $\mathbf{X}$–action (see Remark 3.4).

The objects of this category may be considered as second projections $p_1 : Y \times X \longrightarrow X$ and the maps as naturality squares. The category, seen as arrows, can be viewed as a fibration sitting over $\mathbf{X}$, i.e.

$$\delta_{\mathbf{Y}} : P_{\mathbf{X}}(\mathbf{Y}) \longrightarrow \mathbf{X}$$

where $(f, x) \mapsto x$. The second projections behave as display maps [21], i.e. context extractors. The map which carries $\mathbf{Y}$ to the fibration $\delta_{\mathbf{Y}}$ is the object map of a 2–functor from strong categories to split fibrations over $\mathbf{X}$, denoted $P_{\mathbf{X}}(\_)$. We are now at the starting point for discussing the correspondence of strength to fibrations attributed to Plotkin by Moggi. The remainder of this section is dedicated to proving:

Theorem 3.1 *The 2–functor* $P_{\mathbf{X}}(\_) : \mathbf{Strong}(\mathbf{X}) \longrightarrow \mathbf{SplFib}(\mathbf{X})$ *is a full and faithful embedding.*

Let $\mathbf{StrSplFib}(\mathbf{X})$ denote the 2–category of split fibrations of the form $\delta_{\mathbf{Y}}$. The 1-cells are *morphisms of fibrations* (functors $\tilde{F} : P_{\mathbf{X}}(\mathbf{Y}) \longrightarrow P_{\mathbf{X}}(\mathbf{Z})$ that commute with the fibration legs and preserve the splitting on the nose). The 2-cells are *natural transformations of morphisms of fibrations* ($\tilde{\alpha} : \tilde{F}_1 \longrightarrow \tilde{F}_2$ such that $\delta_{\mathbf{Y}} = \delta_{\mathbf{Z}} * \tilde{\alpha}$ (horizontal composite)). We will show that $P_{\mathbf{X}}(\_)$ factored through this full sub–2–category is an isomorphism.

First we assure ourselves that the functors δ are indeed fibrations that individually possess a splitting. The underlying cleavage is defined by

$$(x, (Y, X')) \mapsto (p_0, x) \quad .$$

This is illustrated by the diagram

$$\begin{tikzcd}
(Y, X) \arrow[r, "{(p_0, x)}"] \arrow[d, dotted, "\delta"'] & (Y, X') \arrow[d, dotted, "\delta"'] \\
X \arrow[r, "x"'] & X'
\end{tikzcd}$$

The fact that this cleavage yields cartesian arrows follows directly from the definitions and parallels the reasoning that

$$\begin{tikzcd}
Y \times X \arrow[r, "i \times x"] \arrow[d, "p_1"'] \arrow[dr, phantom, "pb"] & Y \times X' \arrow[d, "p_1"'] \\
X \arrow[r, "x"'] & X'
\end{tikzcd}$$

is a pullback. Because the $\mathbf{X}$–action canonically provides the products used in defining the maps in $P_{\mathbf{X}}(\mathbf{Y})$, the cleavage is closed to composition, i.e. it is a splitting.

Suppose $\tilde{F} : P_{\mathbf{X}}(\mathbf{Y}) \longrightarrow P_{\mathbf{X}}(\mathbf{Z})$ is a morphism of fibrations. On the objects it must take the simple form

$$\tilde{F}(A, X) = (F(A), X)$$

for some object function F due jointly to fixing the second coordinate and preserving splitting on the nose, thereby allowing the definition of F to be independent of X. On maps it must take the less precise form

$$\tilde{F}(f,x) = (\tilde{F}_0(f,x),x)$$

since the fibration morphism preserves fibers over morphisms as well.

In the fiber over 1 we may recapture an ordinary functor $F : \mathbf{Y} \longrightarrow \mathbf{Z}$ by setting for $x : Y \longrightarrow Y'$

$$F(x) = \langle id_{F(Y)}, !_{F(Y)} \rangle \,;\, \tilde{F}_0(p_0 \,;\, x \;,\; id_1) \quad .$$

To see that this is a functor note first that

$$F(id) = \langle id, ! \rangle \,;\, \tilde{F}_0(p_0, id_1) = \langle id, ! \rangle \,;\, p_0 = id$$

where $\tilde{F}_0(p_0, id_1) = p_0$ since $\tilde{F}$ preserves identities. For composition we exploit the fact

$$\tilde{F}(p_0 \,;\, x \,;\, y \;,\; id_1) = \tilde{F}(p_0 \,;\, x \;,\; id_1) \,;\, \tilde{F}(p_0 \,;\, y \;,\; id_1)$$

so that

$$\begin{aligned}
\langle id, ! \rangle \,;\, \tilde{F}_0(p_0 \,;\, x \,;\, y \;,\; id_1) &= \langle id, ! \rangle \,;\, \langle \tilde{F}_0(p_0 \,;\, x \;,\; id_1) \;,\; p_1 \,;\, id_1 \rangle \,;\, \tilde{F}_0(p_0 \,;\, y \;,\; id_1) \\
&= \langle id, ! \rangle \,;\, \tilde{F}_0(p_0 \,;\, x \;,\; id_1) \,;\, \langle id, ! \rangle \,;\, \tilde{F}_0(p_0 \,;\, y \;,\; id_1)
\end{aligned}$$

follows, showing composition is preserved.

Next we shall show that this functor has a strength given by $\theta^F = \tilde{F}_0(id, !)$, the first component of

$$\tilde{F}(id, !) \;:\; (F(A), X) \longrightarrow (F(A \times X), 1) \quad .$$

θ^F's naturality in each argument derives easily from the commutativity of the following square:

Lemma 3.2 *The diagram*

$$\begin{array}{ccc}
(F(A), X) & \xrightarrow{(\theta^F, !)} & (F(A \times X), 1) \\
\Big\downarrow{\scriptstyle (p_0 \,;\, F(a) \;,\; x)} & & \Big\downarrow{\scriptstyle (p_0 \,;\, F(a \times x) \;,\; id_1)} \\
(F(A'), X') & \xrightarrow[(\theta^F, !)]{} & (F(A' \times X'), 1)
\end{array}$$

commutes.

Proof. The lemma's diagram is the image in $P_{\mathbf{X}}(\mathbf{Z})$ of

$$\begin{tikzcd}
(A,X) \arrow[r, "{(id,!)}"] \arrow[d, "{(p_0\,;a\ ,\ x)}"'] & (A\times X,1) \arrow[d, "{(p_0\,;(a\times x)\ ,\ id_1)}"] \\
(A',X') \arrow[r, "{(id,!)}"'] & (A'\times X',1)
\end{tikzcd}$$

in $P_{\mathbf{X}}(\mathbf{Y})$ under $\tilde{F}$. To see this we observe that

$$\tilde{F}(p_0\,;a\ ,\ x) = \tilde{F}(p_0,x)\,;\tilde{F}(p_0\,;a\ ,\ id) = (p_0,x)\,;\tilde{F}(p_0\,;a\ ,\ id)$$

because the splitting over x, (p_0,x), must be taken by $\tilde{F}$ to the specified (i.e. same) splitting over x. Furthermore, any map $\tilde{F}(p_0\,;g\ ,\ id)$ is determined as the unique map over the splitting of $!\ :\ X \longrightarrow 1$ by applying $\tilde{F}$ to

$$\begin{tikzcd}
(A,X) \arrow[r, "{(p_0,!)}"] \arrow[d, dotted, "{(p_0\,;g\ ,\ id_X)}"'] & (A,1) \arrow[d, "{(p_0\,;g\ ,\ id_1)}"] \\
(A',X) \arrow[r, "{(p_0,!)}"'] & (A',1)
\end{tikzcd}$$

and observing that $(\tilde{F}_0(p_0\,;g\ ,\ id_X)\ ,\ id_X)$ is the unique solution in the resultant image

$$\begin{tikzcd}
(F(A),X) \arrow[r, "{(p_0,!)}"] \arrow[d, dotted] & (F(A),1) \arrow[d, "{(\tilde{F}_0(p_0\,;g\ ,\ id_1),id_1)}"] \\
(F(A'),X) \arrow[r, "{(p_0,!)}"'] & (F(A'),1)
\end{tikzcd}$$

Tracing the diagram's first coordinate yields $\tilde{F}_0(p_0\,;g\ ,\ id_X) = p_0\,;F(g)$, giving the commutativity of the lemma's diagram (and consequently the naturality of θ^F).

□

A more precise form for the $\tilde{F}$–image of an arbitrary map $(a,x)\ :\ (A,X) \longrightarrow (A',X')$. can now be derived:

Lemma 3.3

$$\tilde{F}(a,x) = (\theta^F\,;F(a)\ ,\ x).$$

Proof. A splitting can be factored out by $(a,x) = (a,id)\,;(p_0,x)$, leaving the problem of finding $\tilde{F}(a,id)$. But we note that

$$\begin{tikzcd}
(A,X) \arrow[r, "(id,!)"] \arrow[d, "(a,id)"'] & (A\times X,1) \arrow[d, "(p_0\,;a\ ,\ id_1)"] \\
(A',X) \arrow[r, "(p_0,!)"'] & (A',1)
\end{tikzcd}$$

commutes so $\tilde{F}(a,id)$ is the unique solution (by the familiar splitting argument) in the diagram's $\tilde{F}$-image (created via the proof of Lemma 3.2)

$$\begin{tikzcd}
(F(A),X) \arrow[r, "(\theta^F,!)"] \arrow[d, dotted] & (F(A\times X),1) \arrow[d, "(p_0\,;F(a)\ ,\ id_1)"] \\
(F(A'),X) \arrow[r, "(p_0,!)"'] & (F(A'),1)
\end{tikzcd}$$

which is clearly given by $(\theta^F\,;F(a)\ ,\ id_X)$.

$\square$

This observation is useful in showing how to reconstruct the functor $\tilde{F}$ from (F,θ^F). More immediately we see it provides

$$\begin{tikzcd}
F(A)\times 1 \arrow[rr, "\theta^F"] \arrow[dr, "p_0"'] & & F(A\times 1) \arrow[dl, "F(p_0)"] \\
& F(A) &
\end{tikzcd}$$

the first requirement for strength by examining the first coordinate of the image of

$$\begin{tikzcd}
(A,1) \arrow[rr, "(id_A,!)"] \arrow[dr, "(p_0,id)"'] & & (A\times 1,1) \arrow[dl, "(p_0\,;p_0\ ,\ id_1)"] \\
& (A,1) &
\end{tikzcd}$$

under $\tilde{F}$. For verifying associativity of the strength we can now track the first coordinate in the $\tilde{F}$–image of

$$\begin{tikzcd}
(A, X \times Y) \arrow[r, "(\mathrm{ass}\,;p_0\ ,\ p_1)"] \arrow[d, "(p_0, id)"'] & (A \times X, Y) \arrow[r, "(id, !)"] & ((A \times X) \times Y, 1) \\
(A, X \times Y) \arrow[rr, "(id, !)"'] & & (A \times (X \times Y), 1) \arrow[u, "(p_0\,;\mathrm{ass}\ ,\ id_1)"']
\end{tikzcd}$$

This boils down to distilling from the fibration morphism a strong functor. In similar fashion we can extract from a natural transformation $\tilde{\alpha} : \tilde{F}_1 \longrightarrow \tilde{F}_2$ of fibration morphisms a strong transformation α between the respective extracted functors. The (A, X)-component of $\tilde{\alpha}$ has the general form

$$\tilde{\alpha}^{A,X} = (\tilde{\alpha}_0^{A,X}, id_X)$$

due to the horizontal composition requirement. In the fiber over 1 the A-component of α is defined as

$$\alpha_A = \langle id, ! \rangle\,; \tilde{\alpha}_0^{A,1}$$

It is straightforward to verify the required transformational properties of α and to derive the relationship

$$\tilde{\alpha}^{A,X} = (p_0\,; \alpha_A\ ,\ id_X)$$

directly as the unique comparison over a splitting arrow in the following diagram:

$$\begin{tikzcd}
\tilde{F}(A, X) \arrow[r, "\tilde{F}(p_0, !)"] \arrow[d, dotted] & \tilde{F}(A, 1) \arrow[d, "(\tilde{\alpha}_0^{A,1}, id_1)"] \\
\tilde{G}(A, X) \arrow[r, "\tilde{G}(p_0, !)"'] & \tilde{G}(A, 1)
\end{tikzcd}$$

This extraction process preserves composition of ordinary functors and transformations as it clearly does so in the fiber over 1. We must also check that it preserves the strong compositions. For functors this fact is given by breaking the defining map for the strength of F into a composition

$$(\theta^F, !) = (id, !)\ ;\ (p_0\,; \theta_F\ ,\ id_1)$$

Under $\tilde{G}$ this yields in the first coordinate the Kock equation

$$\theta^{G \circ F} = \theta^G\,; G(\theta^F)$$

as desired. For natural transformations of morphisms of fibrations the strengthening of the extracted natural transformations is found in the first coordinate of

$$\begin{tikzcd}
\tilde{F}(A, X) \arrow[r, "\tilde{F}(id, !)"] \arrow[d, "\tilde{\alpha}^{A,X}"'] & \tilde{F}(A \times X, 1) \arrow[d, "\tilde{\alpha}^{A \times X, 1}"] \\
\tilde{G}(A, X) \arrow[r, "\tilde{G}(id, !)"'] & \tilde{G}(A \times X, 1)
\end{tikzcd}$$

This establishes a 2–functor

$$\mathbf{StrSplFib}(\mathbf{X}) \longrightarrow \mathbf{Strong}(\mathbf{X})$$

To show it is an isomorphism it suffices to show bijectivity on objects, functors, and transformations. By construction, bijectivity immediately holds for objects.

Given a strong functor (F, θ^F) we define an opposing lifting map that sends F to $\tilde{F}$ defined by:

$$[(A, X) \xrightarrow{(a,x)} (A', X')] \quad \mapsto \quad [(F(A), X) \xrightarrow{(\theta^F; F(a)\ ,\ x)} (F(A'), X')].$$

Note that if $\tilde{F}$ is a morphism of fibrations the strong functor recaptured from the fiber over 1 will be F since

$$\begin{aligned} \langle id, !\rangle\,;\tilde{F}_0(p_0\,;a\ ,\ id_1) &= \langle id, !\rangle\,;\theta^F\,;F(p_0\,;a) \\ &= \langle id, !\rangle\,;p_0\,;F(a) \\ &= F(a). \end{aligned}$$

So $F \longrightarrow \tilde{F} \longrightarrow F$ will be the identity. Conversely, Lemma 3.3 will tell us that the transition $\tilde{F} \longrightarrow F \longrightarrow \tilde{F}$ is also the identity. This means the 2–functor will be an isomorphism on functors.

$\tilde{F}$ is indeed a morphism of fibrations as it is a functor $P_{\mathbf{X}}(\mathbf{Y}) \longrightarrow P_{\mathbf{X}}(\mathbf{Z})$ by

$$\begin{aligned} \langle \theta^F\,;F(g)\ ,\ x\rangle\,;\theta^F\,;F(g') &= \langle \theta^F, p_1\rangle\,;(F(g)\times x)\,;\theta^F\,;F(g') \\ &= \langle \theta^F, p_1\rangle\,;\theta^F\,;F(g\times x)\,;F(g') \\ &= \langle id, p_1\rangle\,;(\theta^F\times id)\,;F(g\times x)\,;F(g') \\ &= (id\times\Delta)\,;ass\,;(\theta^F\times id)\,;F(g\times x)\,;F(g') \\ &= (id\times\Delta)\,;\theta^F\,;F(ass)\,;F(g\times x)\,;F(g') \\ &= \theta^F\,;F(id\times\Delta)\,;F(ass)\,;F(g\times x)\,;F(g') \\ &= \theta^F\,;F(\langle g\ ,\ p_1\,;x\rangle\,;g') \end{aligned}$$

and it preserves the splitting because $\theta^F\,;F(p_0) = p_0$.

Finally, we must show that strong transformations can be lifted back to transformations of fibrations. We now know that a strong transformation is determined by its underlying ordinary transformation. From this fact the process of extracting the strong transformation from a lifted strong transformation is clearly seen to behave as the identity.

If the lifting of the strong transformation extracted from a transformation of fibrations can also be proved to be the identity then we will have established an isomorphism of transformations. To confirm this, we first remark that a strong transformation α may be lifted by using the following diagram as a "naturality square":

$$\begin{tikzcd} (F(A), X) \arrow[r, "(\theta^F\,;F(a)\ ,\ x)"] \arrow[d, "(p_0\,;\alpha_A\ ,\ id_X)"'] & (F(A'), X') \arrow[d, "(p_0\,;\alpha_{A'}\ ,\ id_{X'})"] \\ (G(A), X) \arrow[r, "(\theta^G\,;G(a)\ ,\ x)"'] & (G(A'), X') \end{tikzcd}$$

Furthermore, we discovered above that transformations of fibrations also have this form. So the lifting of an extracted transformation is truly the identity. This completes the proof of Theorem 3.1.

We shall hereafter refer to the full and faithful embedding of $\mathbf{Strong}(\mathbf{X})$ into $\mathbf{SplFib}(\mathbf{X})$ that follows from this development as the *strong–fibration embedding.*

Remark 3.4 A very compact perspective of strength leverages the equivalence between the categories of split fibrations and of indexed categories. Because we can view the $\mathbf{X}$–action $\oslash$ of the $\mathbf{X}$–strong category $\mathbf{Y}$ as an ordinary product in $\mathbf{Y}$, each functor $\_ \oslash X$ for $X \in \mathbf{X}$ is a comonad of $\mathbf{Y}$. The unit and multiplier are:

$$\eta : \_ \oslash X \xrightarrow{p_0} \_$$

$$\mu : \_ \oslash X \xrightarrow{\langle id, p_1 \rangle} (\_ \oslash X) \oslash X$$

The Kleisli category of this comonad is also Lambek's polynomial category obtained by adjoining an element [10].

To each $X \in \mathbf{X}$ we may associate the Kleisli category $\mathbf{Y}_X$. Since any morphism $f : X_1 \to X_2$ in $\mathbf{X}$ clearly induces a morphism of the respective comonads, then it also induces a functor $f^* : \mathbf{Y}_{X_2} \to \mathbf{Y}_{X_1}$. Our association is consequently an indexed category $\mathcal{F} : \mathbf{X}^{op} \to \mathbf{Cat}$. The morphisms of fibrations in $\mathbf{StrSplFib}(\mathbf{X})$ restricted to the fiber over 1, i.e. the strong functors, can now be seen to be exactly the morphisms of the Grothendieck construction of the corresponding split fibration for $\mathcal{F}$.

4 Datatypes and Strength

Tatsuya Hagino's thesis [6] proposed a categorical programming language that introduced two sorts of datatype declarations: initial datatypes and final datatypes. Gavin Wraith [25] somewhat simplified the original system and declarations by assuming explicitly that the underlying setting was a bicartesian closed category. In our treatment we take another direction: assume that the underlying category is a cartesian category and replace the closed requirement by the assumption that the datatypes are strong. Because the strong functors of Section 3 are always covariant, we avoid the well-known nuisance of moving contravariant functors — such as exponentials and hom-functors — to a new venue, e.g. O-categories [19], where they can also act covariantly — the limit-colimit coincidence [17] — and thereby become eligible for computing ω-colimits as solutions of recursive domain equations for datatypes.

Factorizer versions of the Hagino "right" and "left" forms of datatype declaration done in a style similar to that suggested by Wraith are:

$$\begin{array}{lll} \mathbf{data} & C \longrightarrow R(A) & = \\ & d_1 : C \longrightarrow E_1(A, C) & \\ | & \quad \ldots & \\ | & d_n : C \longrightarrow E_n(A, C). & \end{array} \qquad \begin{array}{lll} \mathbf{data} & L(A) \longrightarrow C & = \\ & c_1 : E_1(A, C) \longrightarrow C & \\ | & \quad \ldots & \\ | & c_n : E_n(A, C) \longrightarrow C. & \end{array}$$

The first declaration asserts that any map to the new type $R(A)$ from a state type C is to be determined by a set of programmer-chosen maps from C to the types $E_1(A, C), ..., E_n(A, C)$. The second definition forms exactly the dual. The labels d_i and c_i name the collection of canonical operations (not to be confused with the programmer-chosen maps) that are universally associated with their respective datatypes and in one-to-one correspondence with the programmer-chosen maps in a particular declaration. Clearly the first definition declares a final datatype, the second an initial datatype.

The chosen parameter type variable A usually ranges over a power of the given category. However, we shall take the interpretation that it simply ranges over some category and assume the typing $E_i : \mathbf{A} \times \mathbf{C} \longrightarrow \mathbf{C}$. The object $R(A)$ comes equipped with canonical maps

$$d_i : R(A) \longrightarrow E_i(A, R(A))$$

which Wraith called **destructors** as they play the dual role to the canonical maps associated with $L(A)$

$$c_i : E_i(A, L(A)) \longrightarrow L(A)$$

which are traditionally called **constructors**. With respect to these maps the initial and final datatypes satisfy the following universal properties:

$$\begin{tikzcd} C \arrow[r, "g_i"] \arrow[d, dotted, "f_R(g)"'] & E_i(A, C) \arrow[d, dotted, "E_i(A, f_R(g))"] \\ R(A) \arrow[r, "d_i"'] & E_i(A, R(A)) \end{tikzcd} \qquad \begin{tikzcd} E_i(A, L(A)) \arrow[r, "c_i"] \arrow[d, dotted, "E_i(A, f_L(h))"'] & L(A) \arrow[d, dotted, "f_L(h)"] \\ E_i(A, C) \arrow[r, "h_i"'] & C \end{tikzcd}$$

Here g and h respectively represent the n-tuples of the g_i and h_i morphisms.

We shall denote the datatypes as (R, d) and (L, c), respectively, to emphasize the definitional consolidation of destructors and constructors with their respective datatypes. Additionally, these finality and initiality diagrams often constitute recursive definitions; they shall be referred to as the "recursion" diagrams for their datatypes.

Potentially the category of coalgebras for the final datatypes may be constructed as an **inserter** (see Kelly [8]), a 2–categorical limit, if it exists. The inserter resembles the comma category except that the functors giving the domain and codomain originate on the same object. In fact it is a lax equalizer: for general functors $G, H : \mathbf{A} \to \mathbf{B}$, the inserter is the universal pair (U, β) where $U : \mathbf{Insert}(G, H) \to \mathbf{A}$ is a functor and $\beta : G \circ U \to H \circ U$ is a natural transformation.

For example, in the 2-category **Cat** the objects of the inserter category $\mathbf{Insert}(G, H)$ are the pairs (A, b) where A is in $\mathbf{A}$ and $b : G(A) \to H(A)$. A morphism $(A_1, b_1) \to$

(A_2, b_2) corresponds to the existence of a commuting "naturality square" generated by a $\mathbf{A}$-morphism $f : A_1 \to A_2$:

$$\begin{tikzcd} G(A_1) \arrow[r, "G(f)"] \arrow[d, "b_1"'] & G(A_2) \arrow[d, "b_2"] \\ H(A_1) \arrow[r, "H(f)"'] & H(A_2) \end{tikzcd}$$

The inserter functor U is realized as the underlying (forgetful) functor from $\mathbf{Insert}(G, H)$. The required natural transformation β consequently arises with the components $\beta_{(A,b)} = b$.

In our final datatype situation this means that we seek an underlying functor to the category of origin

$$U_R : \mathbf{Insert}(\Delta_n \circ Pr_1, \langle E_1, ..., E_n \rangle) \longrightarrow \mathbf{A} \times \mathbf{C}$$

and a natural transformation α, regarded as the universal destructor of the final datatype, where

$$\begin{tikzcd} \mathbf{Insert}(\Delta_n \circ Pr_1, \langle E_1, ..., E_n \rangle) \arrow[r, shift left] \arrow[r, shift right] \arrow[r, phantom, "\Downarrow\alpha"] & \mathbf{C}^n \end{tikzcd}$$

in which $\alpha : \Delta_n \circ Pr_1 \circ U_R \longrightarrow \langle E_1, ..., E_n \rangle \circ U_R$. Similarly the opposite construction

$$(\mathbf{Insert}(\langle E_1, ..., E_n \rangle, \Delta_n \circ Pr_1), U_L, \alpha')$$

constructs the algebras for the initial datatype and its universal constructor.

To find what the universal properties expressed by the destructor squares translate to in this construction, we must take heed that these datatypes are *parametrized* by objects of $\mathbf{A}$. To naively demand that the underlying functor followed by the first projection, i.e. $Pr_0 \circ U_R$, have a right adjoint for extracting the final coalgebra and the associated destructor squares is inadequate: such an adjunction would permit the parameter to vary within it!

Parametrization is more precisely viewed in fibrational terms: it is elementary to show that $\mathcal{P} = Pr_0 \circ U_R : \mathbf{Insert}(\Delta_n \circ Pr_1, \langle E_1, ..., E_n \rangle) \longrightarrow \mathbf{A}$ is a cofibration where the co-cartesian arrows in the inserter category of the cofibration) are the naturality squares of the form $((A, C), \alpha) \to ((A', C), \alpha')$ generated by $\mathbf{A} \times \mathbf{C}$-morphisms $(h_{\mathbf{A}}, h_{\mathbf{C}}) : (A, C) \to (A', C)$ where $h_{\mathbf{C}} = id_C$.

The final coalgebra for each A-object can now be viewed as picking out final objects in each fiber of $\mathcal{P}$. This is formalized by asserting the existence of a *fibered* right adjoint R of $Pr_0 \circ U_R (= \mathcal{P})$ between the identity cofibration $\mathcal{I} : \mathbf{A} \to \mathbf{A}$ and $\mathcal{P}$:

$$
\begin{tikzcd}
\mathbf{A} \arrow[dr, "\mathcal{I}"'] \arrow[rr, shift right, "R"'] \arrow[rr, phantom, "\perp"] & & \mathbf{Insert}(\ldots) \arrow[ll, shift right, "\mathcal{P}"'] \arrow[dl, "\mathcal{P}"] \\
& \mathbf{A} &
\end{tikzcd}
$$

The unit and counit of this adjunction are required by definition to be vertical arrows, thereby allowing the restrictions of the functors $\mathcal{P}$ and R to the respective fibers over any object A to form an adjunction where the parameter is constant. Each fiber-restricted right adjoint of $\mathcal{P}$ does the selection of the final coalgebra for the corresponding base in $\mathbf{A}$ exactly according to the universal destructor square. Furthermore, this definition is slightly redundant: it is easy to show that the unit is vertical if and only if the counit is vertical. Since $\mathcal{I}$-verticals can only be identities, the fundamental requirement becomes having a right adjoint to $\mathcal{P}$ that possesses an identity counit.

Moreover, with R being a fibration morphism, this view of parametrization converts back to the general 2-categorical setting where we essentially insist $\mathbf{A}$ to be a reflective subcategory of $\mathbf{Insert}(\ldots)$ via an inclusion that is the right adjoint to $\mathcal{P}$. Under the expanded requirement, we can now let $\mathcal{P}$ act as a (fiber-wise) underlying functor in order to follow the "correct" notion for building final datatypes.

Likewise, the initial datatype algebra for each A-object can be "correctly" specified by noting that $\mathcal{Q} = Pr_0 \circ U_L : \mathbf{Insert}(\langle E_1, ..., E_n\rangle, \Delta_n \circ Pr_1) \longrightarrow \mathbf{A}$ is a fibration and providing a fibered left adjoint L that has an identity unit as follows:

$$
\begin{tikzcd}
\mathbf{Insert}(\ldots) \arrow[dr, "\mathcal{Q}"'] \arrow[rr, shift right, "\mathcal{Q}"'] \arrow[rr, phantom, "\perp"] & & \mathbf{A} \arrow[ll, shift right, "L"'] \arrow[dl, "\mathcal{I}"] \\
& \mathbf{A} &
\end{tikzcd}
$$

Analogously, here $\mathbf{A}$ can also be seen as a coreflective subcategory of $\mathbf{Insert}(\ldots)$ under the inclusion L.

The motivation for laying out such abstract formalization is that once datatype building can be understood in this way we may ship it into any 2–category with finite limits in order to install the correct notion in that setting. The most natural candidate for receiving shipment is, of course, the 2–category $\mathbf{Strong}(\mathbf{X})$. Alas, $\mathbf{Strong}(\mathbf{X})$ is not finitely complete and even fails to have inserters. This situation is rescued by the strong–fibration embedding. With it we can make datatypes using $\mathbf{Strong}(\mathbf{X})$ inside of $\mathbf{SplFib}(\mathbf{X})$ which, by its equivalence to the 2–category of $\mathbf{X}$–indexed categories, is finitely complete.

The abstract construction of a strong initial or strong final datatype using the strong–fibration embedding, i.e. respectively the initial object or the final object of an inserter, can be unwound and the computationally unimportant components of morphism pairs then removed. The resulting elementary recursion diagrams expressing the two relevant universal properties are displayed below. We express the collection of functions, $g_1, \ldots, g_n$ as "gs". For a *strong final datatype* the diagram is

$$\begin{tikzcd}
C \times X \arrow[rr, "\langle g_i, p_1 \rangle"] \arrow[dd, dotted, "\mathit{unfold}\{gs\}"'] & & E_i(A, C) \times X \arrow[d, "\theta_2^{E_i}"] \\
& & E_i(A, C \times X) \arrow[d, dotted, "E_i(A, \mathit{unfold}\{gs\})"] \\
R(A) \arrow[rr, "d_i"'] & & E_i(A, R(A))
\end{tikzcd}$$

in which $\theta_2^{E_i}$, as $\theta^{E_i} ; E_i(p_0, id_{C \times X})$ via the naturality of θ^{E_i}, is the strength applied in the second argument only. Similarly, the diagram for a *strong initial datatype* is

$$\begin{tikzcd}
E_i(A, L(A)) \times X \arrow[rr, "c_i \times id_X"] \arrow[d, "\langle \theta_2^{E_i}, p_1 \rangle"'] & & L(A) \times X \arrow[dd, dotted, "\mathit{fold}\{hs\}"] \\
E_i(A, L(A) \times X) \times X \arrow[d, dotted, "E_i(A, \mathit{fold}\{hs\}) \times id_X"'] & & \\
E_i(A, C) \times X \arrow[rr, "h_i"'] & & C
\end{tikzcd}$$

These diagrams are explored further in the next section, but we should note here the serendipitous inclusion of the context parameter X in the domain of each h_i and g_i, the components of actions — *fold* and *unfold*, respectively — that are in turn determined by these components.

It makes moral intuitional sense to allow the context to be processed with the state by the components of an action using that same context. This feature contrasts with other often-used examples, e.g. concerning lists and natural number objects, where this morality has not been kept.

An immediate corollary of the strong-fibration embedding is :

Lemma 4.1 *In the datatypes (R, d) and (L, c), R and L are strong functors.*

It is well-known that if a category has coproducts and admits the construct of final datatypes then the destructors form a product cone. Dually, if the category has products then the constructors of initial datatypes form coproduct cocones. For strong initial datatypes we have the even stronger result:

Lemma 4.2 *If (L, c) is an initial* $\mathbf{X}$*–strong datatype then c forms an* $\mathbf{X}$*–sum.*

Here we use the term $\mathbf{X}$–*sum* to indicate that the $\mathbf{X}$–action distributes over the coproduct, i.e. the $c_i \times id_X$ morphisms are embeddings.

Proof. Suppose $h_i : E_i(A, L(A)) \times X \longrightarrow C$ then any comparison map

$$h : L(A) \times X \longrightarrow C$$

with $(c_i \times id_X)\,; h = h_i$ satisfies the following recursion diagram

$$\begin{tikzcd}
E_i(A, L(A)) \times X \arrow[r, "c_i \times id_X"] \arrow[d, "\langle \theta_2^{E_i}, p_1 \rangle"'] & L(A) \times X \arrow[dd, "\langle p_0, h \rangle"] \\
E_i(A, L(A) \times X) \times X \arrow[d, "E_i(A, \langle p_0, h \rangle) \times id_X"'] & \\
E_i(A, L(A) \times C) \times X \arrow[r, "\nu_i"'] & L(A) \times C
\end{tikzcd}$$

where $\nu_i = \langle p_0\,; E_i(id_A, p_0)\,; c_i\ ,\ (E_i(id_A, p_0) \times id_X)\,; h_i \rangle$. Therefore it exists and is unique. $\square$

Many initial datatypes have been studied in first-order settings. For example, Leopoldo Román proposed various recursion schemes for the natural numbers in [18] including the one suggested by the above analysis. The first author studied the list datatype in [2] by using a slight variant of the list recursion diagram. For initial datatypes based on functors whose strengths are isomorphisms (these we call *linear functors*) the exact form of the recursion diagrams is not critical. As an example, the recursion suggested in [2] is correct — but morally "wrong" — and is shown by

$$\begin{tikzcd}
A \times (\mathit{list}(A) \times X) \arrow[r, "\mathrm{ass}\,;\,\mathit{cons} \times \mathit{id}_X"] \arrow[d, "\mathit{id}_A \times a"] & \mathit{list}(A) \times X \arrow[d, "a"] & 1 \times X \arrow[l, "\mathit{nil} \times \mathit{id}_X"'] \arrow[dl, "f"] \\
A \times C \arrow[r, "g"'] & C &
\end{tikzcd}$$

It is quickly seen to be equivalent to the morally correct version

$$\begin{tikzcd}
(A \times \mathit{list}(A)) \times X \arrow[r, "\mathit{cons} \times \mathit{id}_X"] \arrow[d, "\langle \mathrm{ass}^{-1}\,;\,\mathit{id}_A \times a\,,\,p_2 \rangle"] & \mathit{list}(A) \times X \arrow[d, "a"] & 1 \times X \arrow[l, "\mathit{nil} \times \mathit{id}_X"'] \arrow[dl, "f"] \\
(A \times C) \times X \arrow[r, "g'"'] & C &
\end{tikzcd}$$

that combines the context X with the state C for processing by the programmer-chosen maps f and g'.

For more complex datatypes based on functors whose strengths are not isomorphisms the equivalence between the datatypes implied by the diagrams above fails.

Choosing the correct diagram in these cases becomes critical, but the proper choice is made evident by our general construction.

A common asserted justification for stronger recursion diagrams are their validity in a cartesian closed category. This justification's weakness is revealed by its not confirming whether we have obtained all the strength it is natural to have.

Yet, admittedly, knowing that our strength-based diagrams are valid in a cartesian closed category certainly provides assurance:

Proposition 4.3 *In a cartesian closed category ordinary datatypes are necessarily strong datatypes.*

Proof. We shall demonstrate this for the initial datatypes. Suppose (L, c) is an ordinary initial datatype, i.e. it satisfies the simple algebra initiality property firstly-mentioned in this section.

Let $h_i : E_i(A, C) \times X \longrightarrow C$ be a collection of programmer-chosen maps. Define h_i^* as a particular transpose via

$$\frac{E_i(A, X \Rightarrow C) \times X \xrightarrow{\langle \theta_2^{E_i}, p_1 \rangle} E_i(A, (X \Rightarrow C) \times X) \times X \xrightarrow{E_i(id_A, eval) \times id_X \,;\, h_i} C}{E_i(A, X \Rightarrow C) \xrightarrow{h_i^*} X \Rightarrow C}$$

Using the h_i^* as programmer-chosen maps for the ordinary initial datatype L, we construct the diagram below to obtain a unique arrow $a : L(A) \longrightarrow X \Rightarrow C$ which gives commutativity in its top square. The bottom pentagon commutes by adjointness.

By rearranging the left vertical composite, the outermost polygon becomes the strong recursion diagram for $\mathit{fold}\{hs\} = (a \times id_X)\,;\mathit{eval}$. Since any *fold* map can be factored in the presence of cartesian closure into the form of the right vertical composite, the comparison map is unique. Thus (L, c) is strong as well.

$$\begin{tikzcd}
E_i(A, L(A)) \times X \arrow[r, "c_i \times id_X"] \arrow[d, "E_i(id_A, a) \times id_X"'] & L(A) \times X \arrow[d, "a \times id_X"] \\
E_i(A, X \Rightarrow C) \times X \arrow[r, "h_i^* \times id_X"] \arrow[d, "\langle \theta_2^{E_i}, p_1 \rangle"'] & (X \Rightarrow C) \times X \arrow[dd, "\mathit{eval}"] \\
E_i(A, (X \Rightarrow C) \times X) \times X \arrow[d, "E_i(id_A, \mathit{eval}) \times id_X"'] & \\
E_i(A, C) \times X \arrow[r, "h_i"'] & C
\end{tikzcd}$$

□

5 Datatypes to Combinator Reduction

We now transition from specifying strong datatypes to establishing a categorical combinator reduction system that is directly derivable from our brand of Hagino-Wraith style declarations. The strong initiality and strong finality diagrams of the previous section provide immediately the re-write rules for the $\mathit{fold}\{h_1, \ldots, h_n\}$ and $\mathit{unfold}\{g_1, \ldots, g_n\}$ actions by pasting the diagrams in the direction of decreasing data structure complexity, i.e. towards earlier declaration. These respective actions are uniquely built ¿from a programmer's specification of context-parameterized state transformers h_i coming from components of a datatype and of context-parametrized state de-transformers g_i going to components of a datatype. If components of a datatype are intuitively regarded both as its generalized "subterms" and as "instances" of the domains (codomains) of the transformers (de-transformers), the rewriting rules can be seen to correspond closely to folding and unfolding program transformations as first cataloged by Burstall and Darlington [1].

Thus the *fold* rewrite rules for the initial datatype L are easily seen to be:

$$c_i \times id_X \,;\, \mathit{fold}^L\{h_1,\ldots,h_n\} \quad\Longrightarrow\quad \langle \theta^{E_i}\{p_0, \mathit{fold}^L\{h_1,\ldots,h_n\}\}, p_1\rangle \,;\, h_i$$

and similarly the *unfold* rewrite rules for the final datatype R are:

$$\mathit{unfold}^R\{g_1,\ldots,g_n\} \,;\, d_i \quad\Longrightarrow\quad \langle g_i, p_1\rangle \,;\, \theta^{E_i}\{p_0, \mathit{unfold}^R\{g_1,\ldots,g_n\}\}$$

where $\theta^F\{f_1,\ldots,f_m\}$ will hereafter abbreviate $\theta^F \,;\, F(f_1,\ldots,f_m)$ for any strong datatype F. We call fold^L the L *fold factorizer* or informally the *fold from* L. Likewise we term unfold^R the R *unfold factorizer* or informally the *unfold to* R. These two factorizers can form a minimal utilizable set of categorical programming primitives within this setting, but such a severe constraint in expressiveness leads expectedly to over-complicated codes.

This problem is alleviated by defining two specializations, respectively, of both folding and unfolding that correspond to familiar and vital programming tools. The definitions leverage the fact that, although we are working in a cartesian setting, this setting is closed over *all* declarations of strong initial and final datatypes.

First, the respective analogies of case analysis and higher-order transformation for initial datatypes can be specified with the use of the available binary products, which can be considered as the strong final datatype *Prod* of parametric arity 2 where the component functors E_1 and E_2 are respectively $Pr_0 \circ Pr_0$ and $Pr_1 \circ Pr_0$:

(i) L *case factorizer* :

$$\begin{aligned}&\mathit{case}^L\{h_1,\ldots,h_n\} \equiv \\ &\quad \mathit{fold}^L\{\ldots, \langle p_0 \,;\, E_i(id_A, p_0) \,;\, c_i \;,\; (E_i(id_A,p_0)\times id_X) \,;\, h_i\rangle, \ldots\} \,;\, p_1\end{aligned}$$

(ii) L *map factorizer* :

$$\mathit{map}^L\{h_1,\ldots,h_n\} \equiv \mathit{fold}^L\{\ldots, \theta^{E_i}\{(h_1,\ldots,h_n), p_0\} \,;\, c_i, \ldots\}$$

Second, the analogies of record formation and higher-order transformation for final datatypes are definable by using the available binary sums, which can be treated as the strong initial datatype *Sum* of arity 2 with the same component functors as *Prod*:

(i) R *record factorizer* :

$$\begin{aligned}&\mathit{record}^R\{g_1,\ldots,g_n\} \equiv \\ &b_1 \times id_X \,;\, \mathit{unfold}^R\{\ldots, \langle \mathit{case}^{Sum}\{p_0 \,;\, d_i \;,\; g_i\} \,;\, E_i\{id_A, b_0\} \;,\; p_1\rangle, \ldots\}\end{aligned}$$

(ii) R *map factorizer* :

$$\begin{aligned}&\mathit{map}^R\{g_1,\ldots,g_n\} \equiv \\ &\quad \mathit{unfold}^R\{\ldots, d_i \times id_X \,;\, \langle \theta^{E_i}\{(g_1,\ldots,g_n), p_0\}, p_1\rangle, \ldots\}\end{aligned}$$

Each of the four specializations and their associated rewriting rules are justified below.

The L case factorizer is precisely the comparison map h of the diagram in the proof of Lemma 4.2. The rewrite rule derives directly from the $\mathbf{X}$–sum factoring property:

$$c_i \times id_X \, ; \mathrm{case}^L\{h_1, \ldots, h_n\} \quad \Longrightarrow \quad h_i$$

The L map factorizer is defined by the diagram below in which the the collection of passed functions, $h_1, \ldots, h_n$ is abbreviated as "hs":

$$\begin{tikzcd}
E_i(A, L(A)) \times X \arrow[rr, "c_i \times id_X"] \arrow[d, "\langle \theta_2^{E_i}, p_1 \rangle"] & & L(A) \times X \arrow[dd, dotted, "\theta^L"] \\
E_i(A, L(A) \times X) \times X \arrow[d, dotted, "E_i(id, \theta^L) \times id_X"] & & \\
E_i(A, L(A \times X)) \times X \arrow[r, "\theta_1^{E_i}"] \arrow[d, dotted, "E_i(id, L(hs)) \times id_X"] & E_i(A \times X, L(A \times X)) \arrow[r, "c_i"] \arrow[d, "E_i(hs, L(hs))"] & L(A \times X) \arrow[d, dotted, "L(hs)"] \\
E_i(A, L(B)) \times X \arrow[r, "\theta^{E_i}\{hs, p_0\}"'] & E_i(B, L(B)) \arrow[r, "c_i"'] & L(B)
\end{tikzcd}$$

The top hexagon defines L *strength* as a fold operator and the outermost polygon defines the L map factorizer as a fold operator. That is, $map^L\{hs\} = \theta^L\{hs\}$. The diagram's polygons are essentially pasted from upper-right-to-lower-left to derive the following rewrite rule:

$$c_i \times id_X \, ; \mathrm{map}^L\{h_1, \ldots, h_n\} \quad \Longrightarrow \quad \theta^{E_i}\{(h_1, \ldots, h_n), \mathrm{map}^L\{h_1, \ldots, h_n\}\} \, ; c_i$$

The R record factorizer can be defined in a manner dual to the definition of the L case factorizer:

$$\begin{tikzcd}
(R(A) + C) \times X \arrow[r, "\nu_i"] \arrow[dd, dotted, "\mathrm{case}^{\mathrm{Sum}}\{p_0, g\}"'] & E_i(A, R(A) + C) \times X \arrow[d, "\theta_2^{E_i}"] \\
& E_i(A, (R(A) + C) \times X) \arrow[d, dotted, "E_i(id_A, \mathrm{case}^{\mathrm{Sum}}\{p_0, g\})"] \\
R(A) \arrow[r, "d_i"'] & E_i(A, R(A))
\end{tikzcd}$$

where

$$\nu_i = \langle \mathit{case}^{Sum}\{p_0\,;d_i\ ,\ g_i\}\ ,\ p_1\rangle\,;E_i(id_A, b_0)\times id_X$$

and $g : C \times X \to R(A)$ is any comparison map such that $g\,;d_i = g_i$. Here $\mathit{record}^R\{g_1,\ldots,g_n\}$ is defined to be exactly g which is, in turn, defined by the unique definition of $\mathit{case}^{Sum}\{p_0, g\}$ as an unfold operator. By Lemma 4.2 g is uniquely specified. From the definition's factorization condition we have the rewrite rule:

$$\mathit{record}^R\{g_1,\ldots,g_n\}\,;d_i \quad\Longrightarrow\quad g_i$$

The duality is seen clearly by noting that the L case factorizer definition can now be expressed in terms of record factorizers, viz. by using the relation $\mathit{record}^{Prod}\{f_1, f_2\} = \langle f_1, f_2\rangle$ for f_1 and f_2 having a common domain.

Finally, the R map factorizer is defined by the following diagram in which the collection of passed functions $g_1,\ldots,g_n$ is abbreviated as "gs":

$$\begin{tikzcd}
R(A)\times X \arrow[r, "d_i\times id_X"] \arrow[ddd, dotted, "\theta^R"] & E_i(A,R(A))\times X \arrow[d, "\langle\theta_1^{E_i}, p_1\rangle"] & \\
& E_i(A\times X, R(A))\times X \arrow[r, "E_i(gs,id)\times id_X"] \arrow[d, "\theta_2^{E_i}"] & E_i(B,R(A))\times X \arrow[d, "\theta_2^{E_i}"'] \\
& E_i(A\times X, R(A)\times X) \arrow[r, "E_i(gs,id)"] \arrow[d, dotted, "E_i(id,\theta^R)"] & E_i(B, R(A)\times X) \arrow[dd, dotted, "E_i(id,\theta^R\{gs\})"'] \\
R(A\times X) \arrow[r, "d_i"'] \arrow[d, dotted, "R(gs)"] & E_i(A\times X, R(A\times X)) \arrow[dr, "E_i(gs,R(gs))"'] & \\
R(B) \arrow[rr, "d_i"'] & & E_i(B,R(B))
\end{tikzcd}$$

The top left hexagon defines R *strength* as an unfold operator and the outermost polygon defines the R map factorizer as an unfold operator. Again, this means $map^R\{gs\} = \theta^R\{gs\}$. The polygons of the diagram are pasted lower-left-to-upper-right to obtain the rewrite rule:

$$map^R\{g_1,\ldots,g_n\}\,;d_i \quad\Longrightarrow\quad d_i\times id_X\,;\theta^{E_i}\{(g_1,\ldots,g_n), map^R\{g_1,\ldots,g_n\}\}$$

Conjoining these six rewriting rules with the well-known basic rewriting rules for cartesian categories forms a strongly-normalizing polymorphic combinatory reduction system in which an expansive variety of eager initial and lazy final datatypes can be declared in a cartesian setting and their terms computed. This system forms the abstract machine backbone of the **Charity** categorical programming language implemented by Cockett and Fukushima [4].

To balance our discussions of the more familiar initial datatypes, we close this section with two concrete strong final datatype examples: infinite lists and colists, two forms of "lazy lists". Their utility often lies in reducing complexity in list manipulation. Final datatypes are often infinitary and correspond to greatest fixpoints of colimit-preserving functors (e.g. see Manes and Arbib [11]). Our first example, infinite lists, are declared with the component functors Pr_0 and Pr_1 and with the destructors that extract respectively the head and the (infinite) tail of an infinite list. Essentially we are solving the domain equation $\mathit{Inflist}(A) = A \times \mathit{Inflist}(A)$ maximally, i.e. finding all truly infinite lists of the parameter type A. The strong infinite list recursion diagram is presented below:

$$
\begin{tikzcd}
A \times X \arrow[d, "p_0"'] & C \times X \arrow[l, "\langle f, p_1 \rangle"'] \arrow[r, "\langle g, p_1 \rangle"] \arrow[d, "\mathit{unfold}^{\mathit{Inflist}}\{f,g\}"'] & C \times X \arrow[d, "\mathit{unfold}^{\mathit{Inflist}}\{f,g\}"'] \\
A & \mathit{Inflist}(A) \arrow[l, "\mathit{head}"] \arrow[r, "\mathit{tail}"'] & \mathit{Inflist}(A)
\end{tikzcd}
$$

The colist datatype includes the non-empty finite lists as well as the infinite lists. Here Pr_0 and $\mathit{Sum}\{1, C\}$ are the declaration functors and consequently we envision the equation $\mathit{Colist}(A) = A + (A \times \mathit{Colist}(A)) = A \times (1 + \mathit{Colist}(A))$. Recall that the second equality follows ¿from Lemma 4.2. Using like-named destructors, the strong colist recursion diagram reduces to

$$
\begin{tikzcd}
A \times X \arrow[d, "p_0"'] & C \times X \arrow[l, "\langle f, p_1 \rangle"'] \arrow[r, "\langle g, p_1 \rangle"] \arrow[d, "\mathit{unfold}^{\mathit{Colist}}\{f,g\}"'] & (1 + C) \times X \arrow[d, "\theta^{\mathit{Sum}}\{p_0, \mathit{unfold}^{\mathit{Colist}}\{f,g\}\}"'] \\
A & \mathit{Colist}(A) \arrow[l, "\mathit{head}"] \arrow[r, "\mathit{tail}"'] & 1 + \mathit{Colist}(A)
\end{tikzcd}
$$

The partiality of the colist *tail*, revealed by the appearance of an error type (1) in its codomain, contrasts with the totality of *head*.

6 Conclusions and Future Work

A forthcoming paper will describe a term logic with support for strong datatypes which serves as a categorical programming language. The translation of this term logic into the above datatype combinators will be explained and proven to be consistent and functionally complete.

Such a categorical programming language has been implemented: **Charity** is currently an operational testbed for categorical programming and uses a mixture of eager initial and lazy final datatypes (Cockett and Fukushima [4]). As a programming language it is remarkably useable (considering its formal nature and the fact that all programs strongly normalize). The arithmetic strength of the setting is weaker than most comparable systems (as there is no exponentiation) but does allow, for example, the definition of Ackermann's function.

Categorical computation remains relatively inefficient compared even to conventional functional languages, so techniques such as optimizing the translation of the term logic to combinators in order to speed up execution will be important. Recent intriguing results by Barry Jay [7] on the efficiency of iteration within categorical computation may help illuminate such optimizations. The discovery of additional specializations of the fold and unfold factorizers to outfit a categorical programmer's "toolkit" in the sense hinted by David Turner [22] would further this goal. Also, optimizing via program transformations similar to those developed by Malcolm [15, 16] for general Hagino datatypes may be feasible. Finally, the current research surge in functional programming analysis ¿from the perspectives of control, continuations, and partial evaluation might well be worthwhile in this setting.

Cockett's recent investigations [3] indicate that attractive abstract data structure specifications, such as those espoused by Walters [23], are highly applicable to this programming setting. Also, preliminary investigations have begun towards incorporating dependent types into the system which would yield further improvements in abstract data structure expressiveness.

7 Acknowledgements

Robin Cockett would like to acknowledge the support — financial and intellectual — that the Australian Research Council and the Sydney category seminar provided during the preparation of this work. He is grateful to Richard Wood for explaining his **V**–indexed categories [24] amid the Kangaroos and to Barry Jay for his burgeoning interest in this topic. And above all, he would like to acknowledge the constant contributions and support provided by Ross Street throughout his stay in Sydney.

Dwight Spencer wishes to thank the Formal Methods Group of the University of Calgary Computer Science Department for giving warm support and open collaboration to this author's pursuits in categorical computation. He gives particular thanks to Tom Fukushima for his sharp-eyed error-checking of this paper.

References

[1] R.M. Burstall and J. Darlington *A transformation system for developing recursive programs.* Journal of the Association for Computing Machinery, 24, No. 1 (1977), 44-67.

[2] J.R.B. Cockett, *List-arithmetic distributive categories: locoi.* Journal of Pure and Applied Algebra 66 (1990) 1-29.

[3] J.R.B. Cockett, *Conditional control is not quite categorical control*, IV Higher Order Workshop, Banff 1990, Graham Birtwistle (Ed.) Springer–Verlag, Workshops in Computing (1991) 190-217.

[4] J.R.B. Cockett and T. Fukushima, *About Charity*, University of Calgary (1991). Unpublished manuscript.

[5] J. W. Gray, *Fibred and cofibred categories.* Proceedings of the Conference on Categorical Algebra, La Jolla, 1965 (Springer, Berlin) 21-84.

[6] T. Hagino, *A categorical programming language.* Ph.D. Thesis, University of Edinburgh (1987).

[7] C.B. Jay, *Tail recursion via universal invariants*, University of Edinburgh Research Report Preprint (1991).

[8] G.M. Kelly, *Elementary observations on 2-categorical limits.* Bulletin of the Australian Mathematical Society 39 (1989) 301-317.

[9] A. Kock, *Strong functors and monoidal monads.* Archiv der Mathematik, 23 (1972) 113-120.

[10] J. Lambek and P.J. Scott, *Introduction to Higher Order Categorical Logic.* Cambridge University Press (1986).

[11] E. G. Manes and M. A. Arbib, *Parametrized datatypes do not need highly constrained parameters.* Information and Control 52 (1982) 139-158.

[12] E. Moggi, *Computational lambda-calculus and monads.* Proceedings of the 4^{th} Annual Symposium on Logic in Computer Science (1989) 14-23.

[13] E. Moggi, *An Abstract View of Programming Languages.* Lecture Notes, University of Edinburgh (June 1989).

[14] E. Moggi, *Notions of computation and monads.* Information and Control 93 (1991) 55-92.

[15] G. Malcolm, *Data Structures and Program Transformation.* Science of Computer Programming 14 (1990) 255-279.

[16] G. Malcolm, *Algebraic Data Types and Program Transformation.* Ph.D. Thesis, University of Groningen (1990).

[17] P. Panangaden, P. Mendler and M. Schwartzbach, *Recursively defined types in constructive type theory* in Resolution of Equations in Algebraic Structures. Academic Press (1989).

[18] L. Román, *On recursive principles in cartesian categories.* Preprint.

[19] M. Smyth and G. Plotkin, *The category theoretic solution of recursive domain equations.* SIAM Journal of Computing 11 (1982) 761-783.

[20] D. Spencer, *A survey of categorical computation: fixed points, partiality, combinators, ... control?* Bulletin of the European Association of Theoretical Computer Science 43 (February, 1991) 285-312.

[21] P. Taylor, *Recursive domains, indexed category theory and polymorphism*, Ph.D. Thesis, University of Cambridge (1986).

[22] D. Turner, *Duality and De Morgan Principles for Lists*, Computer Laboratory Research Report Preprint, University of Kent (1991).

[23] R.F.C. Walters, *Data types in distributive categories.* Bulletin of the Australian Mathematical Society 40 (1989) 79-82.

[24] R.J. Wood, **V**–*indexed categories.* in Springer Lecture Notes in Mathematics 661 (1978) 126-140.

[25] G.C. Wraith, *A note on categorical datatypes* in Springer Lecture Notes in Computer Science 389 (1989) 118-127.