Computer Science
University of Manchester

Encodings In Polymorphism, Revisited

Yuxi Fu

Technical Report UMCS–93–6–4

Encodings In Polymorphism, Revisited

Yuxi Fu

Department of Computer Science
University of Manchester
Oxford Rd., Manchester, U.K.

Abstract

We consider encodings in polymorphism with finite product types. These encodings are given in terms of I-algebras. They have the property that all canonical terms (ground terms) are normal terms. We transplant the proof of a well-known result to our setting and show why weak recursion is admissible. We also show how to carry out the dual encodings using the existential quantifier.

Technical Reports issued by the Department of Computer Science, Manchester University, are available by anonymous ftp from `m1.cs.man.ac.uk` (130.88.13.4) in the directory `/pub/TR`. The files are stored as PostScript, in compressed form, with the report number as filename. Alternatively, reports are available by post from The Computer Library, Department of Computer Science, The University, Oxford Road, Manchester M13 9PL, U.K.

The idea of coding up some well-known free structures in the second order lambda calculus has been around for quite a long time, although the first recorded result is as late as [1]. The criterion for a successful encoding, if possible, of an algebraic structure is that it should produce a model that is isomorphic to the term model of the structure. The main result in *loc.cit.* is that the first order free structures can be faithfully coded in polymorphism. Besides, the coding procedure is effective. Many questions remain unsolved in this area.

The emergence of recent interest in the subject is due to the much studied relationship between polymorphism on one hand and functional languages on the other. Perhaps even more fascinating is the question of parametricity where one's goal is to find models in which the interpretations of the encodings are initial among the models of the corresponding structures. See [5, 12, 18, 6] for some results on the topic. There has also been attempt to carry out the similar construction in dependent calculi [3, 17]. But so far there has been no good result. We are not sure to what extent this is due to our ignorance of the free structures of generalized algebras [2].

From another point of view, the coding technique belongs to internal type theory. For such a technique to work at all, the ambient type theory must be extensional, or equivalently the basic type constructors must be functorial. A well-known example is the encoding of inductive types via W-types, cf. [4, 9]. This point is extremely important if we are designing a language that admits the introduction of new types, hence new type constructors. Then in order for the system to work properly, the user-defined types must be extensional.

This is largely a summary paper. But there are some improvements and generalizations. We will show how inductive types can be coded in polymorphism via weakly initial I-algebras which are more suitable than weakly initial T-algebras. Next we show why weak recursion is admissible in the polymorphic encodings. Finally, we transplant the proof in [1] to our setting. The dual encodings for coinductive types are also mentioned.

1 Functors in Polymorphism

The second order lambda calculus $\mathcal{F}_2$, the second order λ-calculus, we use has extensional finite product types. The reasons for this requirement are twofold: 1. We want simple recursion, albeit a weak one. This can be obtained via induction only if the extensional product types are available. 2. The usual encoding for inductive types has a deficiency. For example both $\forall X.X \to (X \to X) \to X$ and $\forall X.(X \to X) \to X \to X$ code the type of natural numbers $\mathbf{N}$. But the canonical term $\Lambda X.\lambda f{:}X \to X.\lambda x{:}X.fx$ is not normal. We can do away with this problem by using product types as in $\forall X.X \wedge (X \to X) \to X$ where $\wedge$ is the product operator.

We say a type variable X occurs *positively* (*negatively*) in a type expression $F(X)$ if (i) $F(X)$ is a constant type or (ii) X occurs positively (negatively) in both $F_1(X)$ and $F_2(X)$ when $F(X) = F_1(X) \wedge F_2(X)$ or (iii) X occurs negatively (positively) in $F_1(X)$ and positively (negatively) in $F_2(X)$ when $F(X) = F_1(X) \to F_2(X)$ or (iv) X occurs positively (negatively) in $G(X,Y)$ when $F(X) = \forall Y.G(X,Y)$. A positive or negative type constructor in a non-dependent typed calculus can be regarded as a functor on the category of terms and contexts. Such functors are put to effective use in [19, 16, 7] to code up inductive and coinductive data types in polymorphism. The technique used here is closely related to the encodings of pure free varieties as carefully studied in [1], cf. also [8].

Let $\mathbf{C}$ be the category whose objects are closed types. A morphism from A to B is the equivalence class of closed terms $f : A \to B$. Suppose $F(X_1, \cdots, X_n)$ (hereafter we often abbreviate $X_1, \cdots, X_n$ to $\vec{X}$) is a type expression in $\mathcal{F}_2$. Then F defines a functor from $\mathbf{C}^\star \times \cdots \times \mathbf{C}^\star$ to $\mathbf{C}$. The i-th component of $\mathbf{C}^\star \times \cdots \times \mathbf{C}^\star$ is $\mathbf{C}$ if X_i occurs positively in $F(\vec{X})$ and is $\mathbf{C}^{op}$ if X_i occurs negatively in $F(\vec{X})$. The term $F(\vec{X})$ defines the object part of a functor. The morphism part F° of the functor should be a term of the following form

$$F^\circ \quad : \quad \forall \vec{X}.\forall \vec{Y}.M_1 \to \cdots \to M_n \to F(\vec{X}) \to F(\vec{Y})$$

where M_i is either $X_i \to Y_i$ or $Y_i \to X_i$ depending on whether X_i occurs positively or negatively in $F(\vec{X})$. By abuse of notation, we will write $\vec{X} \to \vec{Y}$ for $M_1 \to \cdots \to M_n$ and $\lambda \vec{f} : \vec{X} \to \vec{Y}$ for $\lambda f_1{:}M_1. \cdots \lambda f_n{:}M_n$. We now define F° inductively on the structure of $F(\vec{X})$ (we omit the negative case):

- $F(\vec{X}) = A$, a constant type. $F^\circ \stackrel{\text{def}}{=} \Lambda \vec{X}.\Lambda \vec{Y}.\lambda \vec{f}{:}\vec{X} \to \vec{Y}.\lambda a{:}A.a$.
- $F(\vec{X}) = X_i$. $F^\circ \stackrel{\text{def}}{=} \Lambda \vec{X}.\Lambda \vec{Y}.\lambda \vec{f}{:}\vec{X} \to \vec{Y}.\lambda x_i{:}X_i.f_i x_i$.
- $F(\vec{X}) = F_1(\vec{X}) \wedge \cdots \wedge F_n(\vec{X})$. Let

  $$F^\circ \stackrel{\text{def}}{=} \Lambda \vec{X}.\Lambda \vec{Y}.\lambda \vec{f}{:}\vec{X} \to \vec{Y}.\lambda h{:}F_1(\vec{X}) \wedge \cdots \wedge F_n(\vec{X}).\langle (F_1^\circ \vec{X}\vec{Y}\vec{f}(\pi_1 h), \cdots, F_n^\circ \vec{X}\vec{Y}\vec{f}(\pi_n h)\rangle.$$

- $F(\vec{X}) = F_1(\vec{X}) \to F_2(\vec{X})$. By induction hypothesis, we define

  $$F^\circ \stackrel{\text{def}}{=} \Lambda \vec{X}.\Lambda \vec{Y}.\lambda \vec{f}{:}\vec{X} \to \vec{Y}.\lambda h{:}F_1(\vec{X}) \to F_2(\vec{X}).\lambda y{:}F_1(\vec{Y}).F_2^\circ \vec{X}\vec{Y}\vec{f}(h(F_1^\circ \vec{X}\vec{Y}\vec{f}y)).$$

  The term so defined preserves identities:

  $$\begin{aligned} & F^\circ \vec{X}\vec{X}(\lambda x_1{:}X_1.x_1) \cdots (\lambda x_n{:}X_n.x_n) \\ = \; & \lambda h{:}F_1(\vec{X}) \to F_2(\vec{X}).\lambda y{:}F_1(\vec{X}).F_2^\circ \vec{X}\vec{X}(\lambda x_1{:}X_1.x_1) \cdots (\lambda x_n{:}X_n.x_n) \\ & (h(F_1^\circ \vec{X}\vec{X}(\lambda x_1{:}X_1.x_1) \cdots (\lambda x_n{:}X_n.x_n)y)) \\ = \; & \lambda h{:}F_1(\vec{X}) \to F_2(\vec{X}).\lambda y{:}F_1(\vec{X}).F_2^\circ \vec{X}\vec{X}(\lambda x_1{:}X_1.x_1) \cdots (\lambda x_n{:}X_n.x_n)(hy) \\ = \; & \lambda h{:}F_1(\vec{X}) \to F_2(\vec{X}).\lambda y{:}F_1(\vec{X}).hy \\ = \; & \lambda h{:}F_1(\vec{X}) \to F_2(\vec{X}).h \end{aligned}$$

  and compositions:

  $$\begin{aligned} & (F^\circ \vec{Y}\vec{Z}\vec{g}) \circ (F^\circ \vec{X}\vec{Y}\vec{f}) \\ = \; & \lambda h{:}F_1(\vec{X}) \to F_2(\vec{X}).(F^\circ \vec{Y}\vec{Z}\vec{g})((F^\circ \vec{X}\vec{Y}\vec{f})h) \end{aligned}$$

$$
\begin{aligned}
&= \lambda h{:}F_1(\vec{X}) \to F_2(\vec{X}).(F^\circ \vec{Y}\vec{Z}\vec{g})(\lambda y{:}F_1(\vec{Y}).(F_2^\circ \vec{X}\vec{Y}\vec{f})\cdot(h\cdot((F_1^\circ \vec{X}\vec{Y}\vec{f})\cdot y))) \\
&= \lambda h{:}F_1(\vec{X}) \to F_2(\vec{X}).\lambda z{:}F_1(\vec{Z}).(F_2^\circ \vec{Y}\vec{Z}\vec{g})\{(\lambda y{:}F_1(\vec{Y}).(F_2^\circ \vec{X}\vec{Y}\vec{f})\cdot(h\cdot((F_1^\circ \vec{X}\vec{Y}\vec{f})\cdot y))) \\
&\quad ((F_1^\circ \vec{Y}\vec{Z}\vec{g})\cdot z)\} \\
&= \lambda h{:}F_1(\vec{X}) \to F_2(\vec{X}).\lambda z{:}F_1(\vec{Z}).(F_2^\circ \vec{Y}\vec{Z}\vec{g})\{(F_2^\circ \vec{X}\vec{Y}\vec{f})\cdot(h\cdot((F_1^\circ \vec{X}\vec{Y}\vec{f})\cdot((F_1^\circ \vec{Y}\vec{Z}\vec{g})\cdot z)))\} \\
&= \lambda h{:}F_1(\vec{X}) \to F_2(\vec{X}).\lambda z{:}F_1(\vec{Z}).(F_2^\circ \vec{X}\vec{Z}(\vec{g}\circ\vec{f}))\cdot(h((F_1^\circ \vec{X}\vec{Z}(\vec{g}\circ\vec{f}))z)) \\
&= F^\circ \vec{X}\vec{Z}(\vec{g}\circ\vec{f}).
\end{aligned}
$$

- $F(\vec{X}) = \forall Z.G(\vec{X}, Z)$. $F^\circ \stackrel{\text{def}}{=} \Lambda\vec{X}.\Lambda\vec{Y}.\lambda\vec{f}{:}\vec{X} \to \vec{Y}.\lambda h{:}\forall Z.G(\vec{X}, Z).\Lambda Z.G^\circ \vec{X}\vec{Y}\vec{f}(hZ)$. In the above definition, Z is treated as a constant in G°. It can be readily verified that the functor so defined preserves identities and compositions.

Occasionally, we write $F^\circ \vec{f}$ for $F^\circ \vec{X}\vec{Y}\vec{f}$. Finally we remark that extensional rules are essential to the functoriality of F°.

The set of reduction rules is standard:

$$
\begin{array}{rcl}
(\Lambda X.f)\cdot A & \stackrel{\beta_t}{\rightsquigarrow} & f[A/X] \\
\Lambda X.fX & \stackrel{\eta_t}{\rightsquigarrow} & f \\
(\lambda x{:}A.f)\cdot a & \stackrel{\beta}{\rightsquigarrow} & f[a/x] \\
\lambda x{:}A.fx & \stackrel{\eta}{\rightsquigarrow} & f \\
\pi_i\langle a_1, a_2\rangle & \stackrel{\sigma}{\rightsquigarrow} & a_i \\
\langle \pi_1 t, \pi_2 t\rangle & \stackrel{\delta}{\rightsquigarrow} & t \\
u & \stackrel{!}{\rightsquigarrow} & \star
\end{array}
$$

where u is of type $\mathbf{1}$, the unit type. It is well-known that this set of reduction rules is strongly normalizing but not confluent. We should point out that the nonconfluency does not have any effect on the results in section 2.4.

2 Inductive Types in Polymorphism

There are two ways to have inductive types in polymorphism. One is to postulate rules for these types in $\mathcal{F}_2$. This external approach is adopted in [13, 10, 11, 14]. The other approach is to internally code up the inductive types in $\mathcal{F}_2$. The internal method is weaker but by no means less interesting.

2.1 Weakly Initial I-algebras in Polymorphism

In [10, 11] the author studies a categorical language based on the following construction (and its dual) that generates new data types. Let $P(X)$ be an n-tuple $[P_1(X), \cdots, P_n(X)]$

where X occurs positively in $P_1(X), \cdots, P_n(X)$. We may declare a new type as follows:

$$\begin{array}{rcl} \mu X.P(X) & = & \textbf{sum } X \textbf{ with constructors} \\ & & \quad \textbf{in}_1 : P_1(X) \longrightarrow X; \\ & & \quad \cdots \\ & & \quad \textbf{in}_n : P_n(X) \longrightarrow X \\ & & \textbf{end}. \end{array}$$

The reason we call it a *sum*-construction is that, categorically, the meaning of a μ-type is best captured by requiring the functor $[\![P_1(X)]\!] + \cdots + [\![P_n(X)]\!]$ to be initial, where $[\![P_1(X)]\!], \ldots, [\![P_n(X)]\!]$ are interpretations of $P_1(X), \ldots, P_n(X)$ on a suitable category. Our task is to code up this construction in polymorphism as faithfully as we can. This is done in [19] by coding the corresponding initial T-algebras. Our view is that the I-algebras introduced in [15] are most suitable to the description of the encodings of Hagino's categorical data types. Suppose $P_1(X), \cdots, P_n(X)$ are n type expressions in the second order λ-calculus with the free variable X occurring positively in all of them.

Definition 2.1 *Let $F_1, \ldots, F_n$ be functors on a category* $\mathbf{D}$. *A* $[F_1, \ldots, F_n]$-I-algebra *consists of an object A and morphisms $F_i(A) \xrightarrow{in_i} A$ for $i = 1, \ldots, n$. A homomorphism from a $[F_1, \ldots, F_n]$-I-algebra $\langle A, f_1, \ldots, f_n\rangle$ to another $[F_1, \ldots, F_n]$-I-algebra $\langle B, g_1, \ldots, g_n\rangle$ is a morphism $A \xrightarrow{h} B$ such that $h \circ f_i = g_i \circ F_i(h)$ for $i = 1 \cdots n$. The* initial $[F_1, \ldots, F_n]$-I*-algebra is the initial object in the category of $[F_1, \ldots, F_n]$-I-algebras. A* weakly initial $[F_1, \ldots, F_n]$-I*-algebra is a weakly initial object in this category.*

We now define some useful expressions:

$$\begin{array}{rcl} \langle P(X)\rangle & \stackrel{\text{def}}{=} & (P_1(X) \to X) \wedge \cdots \wedge (P_n(X) \to X) \\ \mu X.P(X) & \stackrel{\text{def}}{=} & \forall X.\langle P(X)\rangle \to X \\ \epsilon_P & \stackrel{\text{def}}{=} & \Lambda X.\lambda p{:}\langle P(X)\rangle.\lambda w{:}\mu X.P(X).wXp : \forall X.\langle P(X)\rangle \to \mu X.P(X) \to X \\ \mathsf{In}_i & \stackrel{\text{def}}{=} & \lambda t{:}P_i(\mu X.P(X)).\Lambda X.\lambda p{:}\langle P(X)\rangle.(\pi_i p)(P_i^{\circ} \mu X(\epsilon_P X p)t) : P_i(\mu X.P(X)) \to \mu X.P(X). \end{array}$$

In the above definition, $\pi_i p$ is p if $\langle P(X)\rangle$ has only one component. We will abbreviate $\mu X.P(X)$ to μ when there is no confusion.

Proposition 2.2 $\langle \mu X.P(X), \mathsf{In}_1, \cdots, \mathsf{In}_n\rangle$ *is a weakly initial* $[P_1(X), \cdots, P_n(X)]$-I*-algebra.*

Proof: Suppose $\langle C, c_1, \cdots, c_n\rangle$ is a $[P_1(X), \cdots, P_n(X)]$-I-algebra. We want to show that there is a term $\mathsf{Itr} : \mu X.P(X) \to C$ such that the diagram

$$\begin{tikzcd} P_i(\mu X.P(X)) \arrow[r, "P_i^{\circ}(\mathsf{Itr})"] \arrow[d, "\mathsf{In}_i"'] & P_i(C) \arrow[d, "c_i"] \\ \mu X.P(X) \arrow[r, "\mathsf{Itr}"'] & C \end{tikzcd}$$

commutes for all $i = 1, \cdots, n$. Take

$$\mathsf{Itr} \stackrel{\text{def}}{=} \epsilon_P C\langle c_1, \cdots, c_n\rangle = \lambda w{:}\mu X.P(X).wC\langle c_1, \cdots, c_n\rangle.$$

$$\begin{aligned}
\mathsf{Itr} \circ \mathsf{In}_i &= \lambda t{:}P_i(\mu).\mathsf{Itr}(\mathsf{In}_i t) \\
&= \lambda t{:}P_i(\mu).\mathsf{Itr}(\Lambda X.\lambda p{:}\langle P(X)\rangle.(\pi_i p)(P_i^\circ \mu X(\epsilon_P X p)t) \\
&= \lambda t{:}P_i(\mu).(\Lambda X.\lambda p{:}\langle P(X)\rangle.(\pi_i p)(P_i^\circ \mu X(\epsilon_P X p)t)C\langle c_1, \cdots, c_n\rangle \\
&= \lambda t{:}P_i(\mu).\pi_i\langle c_1, \cdots, c_n\rangle(P_i^\circ \mu C(\epsilon_P C\langle c_1, \cdots, c_n\rangle)t) \\
&= \lambda t{:}P_i(\mu).c_i(P_i^\circ \mu C(\epsilon_P C\langle c_1, \cdots, c_n\rangle)t) \\
&= \lambda t{:}P_i(\mu).c_i(P_i^\circ \mu C\mathsf{Itr}\cdot t) \\
&= c_i \circ (P_i^\circ \mu C\mathsf{Itr}).
\end{aligned}$$

$\mathsf{In}_1, \ldots, \mathsf{In}_n$ are constructors; Itr the iterator. ■

The n equations are precisely the induction scheme. Recursions are also definable due to the presence of extensional product types. To some extent, this is the best we can get. Suppose $C = A \to A$. Let c_i be $\lambda x{:}P_i(X).\lambda a{:}A.a$. Clearly the term $\lambda x{:}\mu X.P(X).\lambda a{:}A.a$ is a morphism from $\langle \mu X.P(X), \mathsf{In}_1, \cdots, \mathsf{In}_n\rangle$ to $\langle C, c_1, \cdots, c_n\rangle$. There are many cases in which one can show that $\mathsf{Itr} \neq \lambda x{:}\mu X.P(X).\lambda a{:}A.a$. The question remains if Itr is unique when C is not inhabited by a closed term.

Example 2.3 In the case of natural numbers, we see that $P_1(X) = \mathbf{1}$, $P_2(X) = X$. Let $\mathbf{N}$ be $\mu X.P(X) = \forall X.(\mathbf{1} \to X) \wedge (X \to X) \to X$, zero be $\mathsf{In}_1(\star)$ and $\mathsf{succ}(t)$ be $\mathsf{In}_2(t)$. Then

$$\begin{aligned}
\mathsf{N} &= \Lambda X.(\mathbf{1} \to X) \wedge (X \to X) \to X \\
\mathsf{zero} &= \Lambda X.\lambda p{:}(\mathbf{1} \to X) \wedge (X \to X).(\pi_1 p) \star \\
\mathsf{succ}(t) &= \Lambda X.\lambda p{:}(\mathbf{1} \to X) \wedge (X \to X).(\pi_2 p)(tXp) \\
\mathsf{Itr}(\mathsf{zero}) &= c_1(\star) \\
\mathsf{Itr}(\mathsf{succ}(t)) &= c_2(\mathsf{Itr}(t)).
\end{aligned}$$

It clearly conforms to the standard encoding. ■

Example 2.4 The type of lists of A is defined in Hagino's notation as follows:

$$\begin{aligned}
\mathrm{List}(A) \;=\; & \textbf{sum } X \textbf{ with constructors} \\
& \quad \textbf{nil} : \mathbf{1} \longrightarrow X; \\
& \quad \textbf{con} : A \times X \longrightarrow X \\
& \textbf{end}.
\end{aligned}$$

It is routine to check that

$$\begin{aligned}
\mathsf{List}(A) &= \Lambda X.(\mathbf{1} \to X) \wedge (A \times X \to X) \to X \\
\mathsf{nil} &= \Lambda X.\lambda p{:}(\mathbf{1} \to X) \wedge (A \times X \to X).(\pi_1 p) \star \\
\mathsf{con}(a, l) &= \Lambda X.\lambda p{:}(\mathbf{1} \to X) \wedge (A \times X \to X).(\pi_2 p)\langle \pi_1 l, (\pi_2 l)Cp\rangle \\
\mathsf{Itr}(\mathsf{nil}) &= c_1(\star) \\
\mathsf{Itr}(\mathsf{con}(a, l)) &= c_2\langle a, \mathsf{Itr}(l)\rangle.
\end{aligned}$$

This encoding gives rise to a term algebra, as we will soon see. ■

2.2 Coding Mutually Dependent Inductive Types

The data types *tree* and *forest* are defined in terms of each other. That is an example of mutually dependent inductive data types. We use the following syntax:

$$
\begin{array}{rl}
\langle \mu_1, \cdots, \mu_n \rangle = & \textbf{sum } (X_1, \cdots, X_n) \textbf{ with constructors} \\
& \quad \mathbf{in}_{1,1} : P_{1,1}(X_1, \cdots, X_n) \longrightarrow X_1, \\
& \quad \cdots \\
& \quad \mathbf{in}_{1,n_1} : P_{1,n_1}(X_1, \cdots, X_n) \longrightarrow X_1; \\
& \quad \cdots \\
& \quad \mathbf{in}_{n,1} : P_{n,1}(X_1, \cdots, X_n) \longrightarrow X_n, \\
& \quad \cdots \\
& \quad \mathbf{in}_{n,n_n} : P_{n,n_n}(X_1, \cdots, X_n) \longrightarrow X_n \\
& \textbf{end}.
\end{array}
$$

Example 2.5 The data type of trees and forests are defined as follows:

$$
\begin{array}{rl}
\langle T, F \rangle = & \textbf{sum } (X_1, X_2) \textbf{ with constructors} \\
& \quad \mathbf{span} : A \times X_2 \longrightarrow X_1; \\
& \quad \mathbf{join} : X_1 \times X_2 \longrightarrow X_2, \\
& \quad \mathbf{empty} : \mathbf{1} \longrightarrow X_2 \\
& \textbf{end}.
\end{array}
$$

It says that a tree consists of a root and a forest; a forest is either empty or a tree together with a forest. The structural recursion enjoys three equations:

$$
\begin{array}{rcl}
\mathsf{Itr}_F(\mathsf{empty}) & = & c_1(\star) \\
\mathsf{Itr}_F(\mathsf{join}(t, f)) & = & c_2(\mathsf{Itr}_T(t), \mathsf{Itr}_F(f)) \\
\mathsf{Itr}_T(\mathsf{span}(a, f)) & = & c_3(a, \mathsf{Itr}_F(f)).
\end{array}
$$

The intuition should be clear. ■

Definition 2.6 *Suppose we have a group of functors*

$$
\begin{array}{rcl}
F_{1,1}(X_1, \cdots, X_n) & : & \mathbf{C}_1 \times \cdots \times \mathbf{C}_n \longrightarrow \mathbf{C}_1 \\
& & \cdots \\
F_{1,n_1}(X_1, \cdots, X_n) & : & \mathbf{C}_1 \times \cdots \times \mathbf{C}_n \longrightarrow \mathbf{C}_1 \\
& & \cdots \\
F_{n,1}(X_1, \cdots, X_n) & : & \mathbf{C}_1 \times \cdots \times \mathbf{C}_n \longrightarrow \mathbf{C}_n \\
& & \cdots \\
F_{n,n_n}(X_1, \cdots, X_n) & : & \mathbf{C}_1 \times \cdots \times \mathbf{C}_n \longrightarrow \mathbf{C}_n
\end{array}
$$

covariant in all their domains. An I^n-algebra *is a tuple* $\langle(\mu_1,\cdots,\mu_n),\mathsf{In}_{1,1},\cdots,\mathsf{In}_{n,n_n}\rangle$ *where*

$$\mathsf{In}_{i,j} : F_{i,j}(\mu_1,\cdots,\mu_n) \longrightarrow \mu_i$$

for all possible i *and* j*. An* initial *I^n-algebra is such that if* $\langle(C_1,\cdots,C_n),c_{1,1},\cdots,c_{n,n_n}\rangle$ *is any other I^n-algebra, then there is a unique tuple* $\langle\mu_1 \xrightarrow{\mathsf{Itr}_1} C_1,\cdots,\mu_n \xrightarrow{\mathsf{Itr}_n} C_n\rangle$ *such that*

$$\begin{tikzcd}
F_{i,j}(\mu_1,\cdots,\mu_n) \arrow[r, "F_{i,j}(\mathsf{Itr}_1,\cdots,\mathsf{Itr}_n)"] \arrow[d, "\mathsf{In}_{i,j}"'] & F_{i,j}(C_1,\cdots,C_n) \arrow[d, "c_{i,j}"] \\
\mu_i \arrow[r, "\mathsf{Itr}_i"'] & C_i
\end{tikzcd}$$

commutes for all possible i,j. If we drop the uniqueness requirement, we get a weakly initial *I^n-algebra.*

Now suppose we have following type constructors with all its type variables occurring positively:

$$\begin{array}{ccc}
P_{1,1}(\vec{X}) & \cdots & P_{1,n_1}(\vec{X}) \\
\cdots & & \cdots \\
P_{n,1}(\vec{X}) & \cdots & P_{n,n_n}(\vec{X}).
\end{array}$$

As before we define μ_i and $\mathsf{In}_{i,j}$ as follows:

$$\begin{array}{rcl}
\mathcal{P}(\vec{X}) & \stackrel{\text{def}}{=} & (P_{1,1}(\vec{X})\rightarrow X_1)\wedge\cdots\wedge(P_{1,n_1}(\vec{X})\rightarrow X_1)\wedge\cdots\wedge(P_{n,1}(\vec{X})\rightarrow X_n)\wedge\cdots\wedge(P_{n,n_n}(\vec{X})\rightarrow X_n) \\
\mu_i\vec{X}.\mathcal{P}(\vec{X}) & \stackrel{\text{def}}{=} & \forall\vec{X}.\mathcal{P}(\vec{X})\rightarrow X_i \\
\epsilon^i_{\mathcal{P}} & \stackrel{\text{def}}{=} & \Lambda\vec{X}.\lambda p{:}\mathcal{P}(\vec{X}).\lambda w_i{:}\mu_i.w_i\vec{X}p : \forall\vec{X}.\mathcal{P}(\vec{X})\rightarrow\mu_i\rightarrow X_i \\
\mathsf{In}_{i,j} & \stackrel{\text{def}}{=} & \lambda t{:}P_{i,j}(\vec{\mu}).\Lambda\vec{X}.\lambda p{:}\mathcal{P}(\vec{X}).(\pi_{i,j}p)(\mathcal{P}^{\circ}_{i,j}\vec{\mu}\vec{X}(\epsilon^1_{\mathcal{P}}\vec{X}p)\cdots(\epsilon^n_{\mathcal{P}}\vec{X}p)t) : P_{i,j}(\mu\vec{X}.\vec{\mathcal{P}}(\vec{X}))\rightarrow\mu_i\vec{X}.\mathcal{P}(\vec{X})
\end{array}$$

Proposition 2.7 $\langle(\mu_1,\cdots,\mu_n),\mathsf{In}_{1,1},\cdots,\mathsf{In}_{n,n_n}\rangle$ *is a weakly initial I^n-algebra.*

Proof: Let $\langle(C_1,\cdots,C_n),c_{1,1},\cdots,c_{n,n_n}\rangle$ be another I^n-algebra. We define

$$\mathsf{Itr}_i \stackrel{\text{def}}{=} \epsilon^i_{\mathcal{P}}\vec{C}\langle c_{1,1}\cdots c_{n,n_n}\rangle = \lambda w_i{:}\mu_i.w_i\vec{C}\langle c_{1,1}\cdots c_{n,n_n}\rangle : \mu_i\rightarrow C_i$$

for $i = 1\cdots n$. We must check that the computation rule holds:

$$\begin{array}{rcl}
\mathsf{Itr}_i\circ\mathsf{In}_{i,j} & = & \lambda x_i{:}P_{i,j}(\vec{\mu}).\mathsf{Itr}_i(\mathsf{In}_{i,j}(x_i)) \\
 & = & \lambda x_i{:}P_{i,j}(\vec{\mu}).\mathsf{Itr}_i(\Lambda\vec{X}.\lambda p{:}\mathcal{P}(\vec{X}).\cdots)\vec{C}\langle c_{1,1},\cdots,c_{n,n_n}\rangle \\
 & = & \lambda x_i{:}P_{i,j}(\vec{\mu}).\pi_{i,j}\langle c_{1,1},\cdots,c_{n,n_n}\rangle((P^{\circ}_{i,j}\vec{\mu}\vec{C}(\epsilon^1_{\mathcal{P}}\vec{C}\vec{c})\cdots(\epsilon^n_{\mathcal{P}}\vec{C}\vec{c}))x_i) \\
 & = & \lambda x_i{:}P_{i,j}(\vec{\mu}).c_{i,j}(P^{\circ}_{i,j}\vec{\mu}\vec{C}\vec{\mathsf{Itr}}x_i) \\
 & = & c_{i,j}\circ P^{\circ}_{i,j}(\mathsf{Itr}_1,\cdots,\mathsf{Itr}_n).
\end{array}$$

This completes the proof. ∎

Example 2.8 Continuing the example of trees and forests, we see that

$$\begin{array}{rcl}
\mathcal{P}(X,Y) & = & (A\wedge Y\to X)\wedge(\mathbf{1}\to Y)\wedge(X\wedge Y\to Y)\\
\mathsf{T} & = & \forall X.\forall Y.\mathcal{P}(X,Y)\to X\\
\mathsf{F} & = & \forall X.\forall Y.\mathcal{P}(X,Y)\to Y\\
\mathsf{span}(a,f) & = & \Lambda X.\Lambda Y.\lambda p{:}\mathcal{P}(X,Y).(\pi_1)\langle a, fXYp\rangle\\
\mathsf{empty} & = & \Lambda X.\Lambda Y.\lambda p{:}\mathcal{P}(X,Y).(\pi_2)(\star)\\
\mathsf{join}(t,f) & = & \Lambda X.\Lambda Y.\lambda p{:}\mathcal{P}(X,Y).(\pi_3)\langle fXYp, fXYp\rangle
\end{array}$$

Suppose $\langle (C_1,C_2), c_1, c_2, c_3\rangle$ *is another I-algebra.*

$$\begin{array}{rcl}
\mathsf{Itr}_{\mathsf{T}}(\mathsf{span}(a,f)) & = & c_3\langle a, fC_1C_2\langle c_1,c_2,c_3\rangle\rangle\\
\mathsf{Itr}_{\mathsf{F}}(\mathsf{empty}) & = & c_1(\star)\\
\mathsf{Itr}_{\mathsf{F}}(\mathsf{join}(t,f)) & = & c_2\langle tC_1C_2\langle c_1,c_2,c_3\rangle, fC_1C_2\langle c_1,c_2,c_3\rangle\rangle.
\end{array}$$

They are exactly what one would expect. ∎

2.3 Weak Recursion

Lemma 2.9 *Let* $P(\vec{X})$ *be a type in* $\mathcal{F}_2$ *with free variables among* $X_1,\cdots,X_n$ *and without quantification. Then every term of* $P(\mathbf{1},\cdots,\mathbf{1})$*, not necessary closed, is equal to a unique closed normal term* $!_{P(\mathbf{1},\cdots,\mathbf{1})}$.

Proof: 1. $P(\vec{X})$ is the unit type. By the extensional rule for the unit type. 2. $P(\vec{X}) = P_1(\vec{X})\wedge P_2(\vec{X})$. By the surjective pairing rule. 3. $P(\vec{X}) = P_1(\vec{X})\to p_2(\vec{X})$. Suppose t is of type $P(\mathbf{1},\cdots,\mathbf{1})$. Then $t = \lambda x{:}P_1(\mathbf{1},\cdots,\mathbf{1}).tx = \lambda x{:}P_1(\mathbf{1},\cdots,\mathbf{1}).!_{P_2(\mathbf{1},\cdots,\mathbf{1})} = !_{P(\mathbf{1},\cdots,\mathbf{1})}$. This completes the proof. ∎

Let $t_1:\mu_1,\cdots,t_n:\mu_n$. We have

$$P^{\circ}_{i,j}(\lambda x{:}\mathbf{1}.t_1,\cdots,\lambda x{:}\mathbf{1}.t_n) : P_{i,j}(\vec{\mathbf{1}})\to P_{i,j}(\vec{\mu}).$$

By abuse of notation we will write $P^{\circ}_{i,j}(t_1,\cdots,t_n)$ instead.

The equation in the proof of proposition 2.7 implies

$$\mathsf{Itr}_i\circ\mathsf{In}_{i,j}\circ P^{\circ}_{i,j}(t_1,\cdots,t_n) = c_{i,j}\circ P^{\circ}_{i,j}(\mathsf{Itr}_1(t_1),\cdots,\mathsf{Itr}_n(t_n)).$$

Definition 2.10 *A term* $t:\mu_i$ *is said to be a* ground term *of type* μ_i *if it is (i)* $\mathsf{In}_{i,j}(\star)$ *when* $P_{i,j}(\vec{X}) = \mathbf{1}$*; (ii)* $\mathsf{In}_{i,j}(P^{\circ}_{i,j}(t_1,\cdots,t_n)\diamond)$ *where* $\diamond$ *is a closed term of type* $P_{i,j}(\vec{\mathbf{1}})$ *and* $t_1,\cdots,t_n$ *are ground terms.*

Suppose we have terms $f_{i,j} : P_{i,j}(\mu_1\wedge C_1,\cdots,\mu_n\wedge C_n)\to C_i$ for all possible i and j. We can define a weak form of recursion by extracting the second components of the iterators shown in the diagram:

$$\begin{tikzcd}
P_{i,j}(\mu_1, \cdots, \mu_n) \arrow[r, "P^\circ_{i,j}(\langle ?_1, \mathsf{Rec}_1\rangle, \cdots, \langle ?_n, \mathsf{Rec}_n\rangle)"] \arrow[d, "\mathsf{In}_{i,j}"'] & P_{i,j}(\mu_1 \wedge C_1, \cdots, \mu_n \wedge C_n) \arrow[d, "\langle P^\circ_{i,j}(\vec{\pi_1}); \mathsf{In}_{i,j}, f_{i,j}\rangle"] \\
\mu_i \arrow[r, "\mathsf{Itr}_i = \langle ?_i, \mathsf{Rec}_i\rangle"'] & \mu_i \wedge C_i.
\end{tikzcd}$$

It is the consequence of the weakness of the initiality that we cannot prove $?_i = Id_{\mu_i}$. So we only have

$$\mathsf{Rec}_i \circ \mathsf{In}_{i,j} = f_{i,j} \circ P^\circ_{i,j}(\langle ?_1, \mathsf{Rec}_1\rangle, \cdots, \langle ?_n, \mathsf{Rec}_n\rangle),$$

which is not quite what we would like to have. However

Proposition 2.11 $\mathsf{Rec}_i \circ \mathsf{In}_{i,j} \circ P^\circ_{i,j}(t_1, \cdots, t_n) = f_{i,j} \circ P^\circ_{i,j}(\langle t_1, \mathsf{Rec}_1(t_1)\rangle, \cdots, \langle t_n, \mathsf{Rec}_n(t_n)\rangle)$ *if* $t_1 : \mu_1, \cdots, t_n : \mu_n$ *are all ground terms.*

Proof: Notice that $?_i \circ \mathsf{In}_{i,j} = \mathsf{In}_{i,j} \circ P^\circ_{i,j}(?_1, \cdots, ?_n)$ and that if $P_{i,j}(\vec{\mu})$ is $\mathbf{1}$,

$$\begin{tikzcd}
\mathbf{1} \arrow[r, "Id_{\mathbf{1}}"] \arrow[d, "\mathsf{In}_{i,j}"'] & \mathbf{1} \arrow[d, "\langle \mathsf{In}_{i,j}, f_{i,j}\rangle"] \\
\mu_i \arrow[r, "\langle ?_i, \mathsf{Rec}_i\rangle"'] & \mu_i \wedge C_i
\end{tikzcd}$$

commutes and therefore $?_i(\mathsf{In}_{i,j}(\star)) = \mathsf{In}_{i,j}(\star)$. It follows that

$$\begin{array}{rcl}
?_i(t_i) & = & ?_i(\mathsf{In}_{g,h}(P^\circ_{g,h}(s_1, \cdots, s_n)\diamond)) \\
& = & \mathsf{In}_{g,h}(P^\circ_{g,h}(?_1(s_1), \cdots, ?_n(s_n))\diamond) \\
& = & \cdots \\
& = & \mathsf{In}_{g,h}(\cdots, ?_l(\mathsf{In}_{l,q}(\star)), \cdots) \\
& = & \mathsf{In}_{g,h}(\cdots, \mathsf{In}_{l,q}(\star), \cdots) \\
& = & \cdots \\
& = & \mathsf{In}_{g,h}(P^\circ_{g,h}(s_1, \cdots, s_n)\diamond) \\
& = & t_i.
\end{array}$$

Remember our convention and that $\langle ?, \mathsf{Rec}_i\rangle \circ (\lambda x{:}\mathbf{1}.t_i) = \lambda x{:}\mathbf{1}.\langle ?t_i, \mathsf{Rec}_i(t_i)\rangle$,

$$\begin{array}{rcl}
\mathsf{Rec}_i \circ \mathsf{In}_{i,j} \circ P^\circ_{i,j}(t_1, \cdots, t_n) & = & f_{i,j} \circ P^\circ_{i,j}(\langle ?_1, \mathsf{Rec}_1\rangle, \cdots, \langle ?_n, \mathsf{Rec}_n\rangle) \circ P^\circ_{i,j}(t_1, \cdots, t_n) \\
& = & f_{i,j} \circ P^\circ_{i,j}(\langle ?_1(t_1), \mathsf{Rec}_1(t_1)\rangle, \cdots, \langle ?_n(t_n), \mathsf{Rec}_n(t_n)\rangle) \\
& = & f_{i,j} \circ P^\circ_{i,j}(\langle t_1, \mathsf{Rec}_1(t_1)\rangle, \cdots, \langle t_n, \mathsf{Rec}_n(t_n)\rangle).
\end{array}$$

This completes the proof. ■

In the case of first order free pure structures, the above equation looks much more familiar:

$$\mathsf{Rec}_i(\mathsf{In}_{i,j}(t_1, \cdots, t_n)) = f_{i,j}(t_1, \mathsf{Rec}_1(t_1), \cdots, t_n, \mathsf{Rec}_n(t_n)).$$

2.4 The First Order Free Structures

A first order signature contains no higher order functions like $f : (s_1 \to s_2) \times s \to s$. By "pure" we mean there is no equational theory. Given a first order finite signature $Sig = \langle s_1, \cdots, s_n, f_{11}, \cdots, f_{nn_n} \rangle$, its free structure (term model) can be perfectly coded by a weakly initial I^n-algebra in polymorphism.

Now for each pair i, j,

$$P_{i,j}(\vec{X}) = X_{i_1} \wedge \cdots \wedge X_{i_m}.$$

It is easily verified that

$$P^{\circ}_{i,j} = \Lambda\vec{X}.\Lambda\vec{Y}.\lambda f{:}\vec{X} \to \vec{Y}.\lambda h{:}X_{i_1} \wedge \cdots \wedge X_{i_m}.\langle f_{i_1}(\pi_1 h), \cdots, f_{i_m}(\pi_m h)\rangle.$$

Therefore

$$\begin{aligned} \mathsf{In}_{i,j} &= \lambda t{:}P_{i,j}(\mu).\Lambda\vec{X}.\lambda p{:}\mathcal{P}(\vec{X}).(\pi_{i,j}p)\langle \epsilon^{i_1}_{\mathcal{P}}\vec{X}p(\pi_1 t), \cdots, \epsilon^{i_m}_{\mathcal{P}}\vec{X}p(\pi_m t)\rangle \\ &= \lambda t{:}P_{i,j}(\mu).\Lambda\vec{X}.\lambda p{:}\mathcal{P}(\vec{X}).(\pi_{i,j}p)\langle (\pi_1 t)\vec{X}p, \cdots, (\pi_m t)\vec{X}p\rangle. \end{aligned}$$

Or equivalently

$$\mathsf{In}_{i,j}(t_1, \cdots, t_m) = \Lambda\vec{X}.\lambda p{:}\mathcal{P}(\vec{X}).(\pi_{i,j}p)\langle t_1\vec{X}p, \cdots, t_m\vec{X}p\rangle \qquad (1)$$

where $t_1, \cdots, t_m$ are terms of types $\mu_1, \cdots, \mu_m$ respectively and are themselves of the form $\mathsf{In}\_(\_)$. This corresponds precisely to the so-called Λ-programmes in [1]. The only difference is that here we use product types. In *loc.cit.*, elements in a free structure are represented by *equivalence classes* of Λ-terms; here they are coded by the uniformly constructed *ground terms*. For this encoding to work, product types are necessary. Lemma 2.13 would fail if the only logical operators we use are $\forall$ and $\to$. By definition 2.10, a term of type μ_i is a *ground term* if (i) it is $\Lambda\vec{X}.\lambda p : \mathcal{P}(\vec{X}).(\pi_{i,j}p)\star$ when $\pi_{i,j}p : \mathbf{1} \to X_i$ or (ii) it is $\Lambda\vec{X}.\lambda p : \mathcal{P}(\vec{X}).(\pi_{i,j}p)\langle e_1, \cdots, e_m\rangle$ when $\pi_{i,j}p$ is of type $\mu_{i_1} \wedge \cdots \wedge \mu_{i_m} \to \mu_i$; and $e_1 : \mu_{i_1}, \cdots, e_m : \mu_{i_m}$ are ground terms. One easily sees that the terms generated by the equation 1 are exactly the ground terms of type μ_i $(i = 1, \ldots, n)$ and these ground terms are normal.

Proposition 2.12 *There are exact correspondences between*
1. sort s_i and type μ_i;
2. function $f_{i,j}$ and the term $\mathsf{In}_{i,j}$;
3. elements of the free structure and ground terms.

Proof: Easy. ■

Lemma 2.13 *The closed normal terms of $\forall\vec{X}.\mathcal{P}(\vec{X}) \to X_i$ are precisely the ground terms.*

Proof: The proof is adapted from [1]. Let $t : \mu_i$ be a closed normal term. We are going to work out the structure of $t\vec{X}p$ where p is a variable of type $\mathcal{P}(\vec{X})$. We first introduce the notion of degrees. Suppose A is a type. The degree of A is (i) 0 if A is either a constant type or a type variable (ii) the maximum of the degrees of its components when A is a product type (iii) $1 + max\{$the degree of A_1, the degree of $A_2\}$ when $A = A_1 \to A_2$ (iv) 1+the degree of $B(X)$ when $A = \forall X.B(X)$. The degree of an object is the degree of its type. In the following proof, we do not regard a type variable as a subterm.

1. $t\vec{X}p$ must be normal and the only variables it contains are $X_1, \ldots, X_n$ and p.

2. $t\vec{X}p$ cannot be of introduction form as X_i, the type of $t\vec{X}p$, is a type variable.

3. If s is a subterm of $t\vec{X}p$ and is abstraction-free, i.e. s contains neither Λ nor λ, then the degree of s is at most one. s must be either $\star$ whose degree is 0, or p whose degree is 1, or a tuple, or $\pi s'$ or $s_1 s_2$. In the third and forth cases we resort to the induction hypothesis. The degree of $s_1 s_2$ must be zero as by induction hypothesis the degree of s_1 is no more than 1.

4. $t\vec{X}p$ does not contain any abstraction. Suppose otherwise. Take the left-most such subterm t'. In view of 1 and 2, t' can only be a subterm of an application $f' \cdot t'$ or a component of a tuple t''. The former case is impossible because by hypothesis f' is abstraction-free and therefore by 3 the degree of f' is less than 1. In the latter case, t'' cannot occur in $\pi t''$, nor can it occur in any abstraction because t' is the left-most such term. t'' cannot occur in $g \cdot t''$ because of 3. The only case left is that t'' is a component of another tuple. But then we can repeat the argument to show that is impossible.

5. It follows from 3 and 4 that $t\vec{X}p$ does not contain any subterm that is of degree more than 1.

6. If $f \cdot a$ is a subterm of $t\vec{X}p$, then f must be either πp or p and therefore the type of $f \cdot a$ is a variable. (i) f is a variable, then it can only be p; (ii) f cannot be an application because of 5; (iii) f cannot be an abstraction due to 4; (iv) f is πb; there are these cases: (a) b cannot be an application by induction hypothesis; (b) b cannot be a tuple by 1; (c) b cannot be $\pi' b'$ because by induction hypothesis on 7, $\pi' b'$ is of an arrow type; (d) b must be p;

7. If πa is a subterm of $t\vec{X}p$, then it must be πp and therefore of an arrow type. (i) a cannot be an application by induction hypothesis on 6; (ii) it cannot be $\pi' a'$ by induction hypothesis; (iii) nor can it be a tuple by 1; (iv) therefore it must be p.

8. From 4, 6 and 7, we can infer that if a subterm of $t\vec{X}p$ has a product type of degree zero, then it must be a tuple.

9. We conclude that $t\vec{X}p$ must be $f \cdot a$. By 6, f must be πp or p. So the type of a is either $\mathbf{1}$ or $X_{i_1} \wedge \cdots \wedge X_{i_m}$(say) or a type variable. In the first case $a = \star$ by the extensional reduction rule for $\mathbf{1}$. In the second case, we infer from 8 that a must be a tuple $\langle t_{i_1}, \cdots, t_{i_m} \rangle$. But then $\Lambda \vec{X}.\lambda p{:}\mathcal{P}(\vec{X}).t_{i_j} : \forall \vec{X}.\mathcal{P}(\vec{X}) \to X_{i_j}$ for $i = 1, \cdots, n$ and we can repeat the whole argument for each t_{i_j}. The third case is simpler than the second case.

So t must be a ground term. ∎

Theorem 2.14 *Let Sig be a first order finite signature. Then the above polymorphic encoding gives rise to a term model of Sig.*

The theorem cannot be extended to higher order cases for at least two reasons: (i) the method cannot cope with cases where a sort occurs negatively as in $f : (s \rightarrow s) \rightarrow s$; (ii) even though we pretend we can treat negative appearance, our method produces more terms than wanted; for instance, suppose we have a signature $Sig = \langle s_1, s_2, f_0 : s_1, f_1 : s_1 \rightarrow s_1, f_2 : (s_1 \rightarrow s_1) \rightarrow s_2 \rangle$; in the term model, either of s_1 and s_2 has one element; but in the representation in $\mathcal{F}_2$, the type that represents s_2 has two elements.

3 Coinductive Types in Polymorphism

The coinductive types are not as familiar as the inductive ones. The purpose of this section is to carry out the constructions dual to the ones given insection 2. First let's refresh our memory on the rules about the extensional existential quantification.

Formation

$$\frac{\Delta, X \vdash P(X)}{\Delta \vdash \exists X.P(X)}$$

Introduction

$$\frac{\Delta, X \vdash P(X) \quad \Delta \vdash A \quad \Gamma \vdash_\Delta p : P(A)}{\Gamma \vdash_\Delta \delta(A, p) : \exists X.P(X)}$$

Elimination

$$\frac{\Gamma \vdash_\Delta t : \exists X.P(X) \quad \Delta \vdash C \quad \Gamma, x : P(X) \vdash_{\Delta, X} q : C}{\Gamma \vdash_\Delta \mathbf{E}(t, (X, x).q) : C}$$

Computation

$$\frac{\Delta, X \vdash P(X) \quad \Delta \vdash A \quad \Gamma \vdash_\Delta p : P(A) \quad \Delta \vdash C \quad \Gamma, x : P(X) \vdash_{\Delta, X} q : C}{\Gamma \vdash_\Delta \mathbf{E}(\delta(A, p), (X, x).q) = q[A/X, p/x]} \beta$$

$$\frac{\Gamma \vdash_\Delta t : \exists X.P(X) \quad \Delta \vdash C \quad \Gamma, z : \exists X.P(X) \vdash_\Delta q : C}{\Gamma \vdash_\Delta \mathbf{E}(t, (X, x).q[\delta(X, x)/z]) = q[t/z] : C} \eta$$

3.1 Weakly Terminal I-algebras in Polymorphism

The dual to Hagino's previous construction is this:

$$\begin{array}{rcl} \nu X.Q(X) & = & \textbf{product } X \textbf{ with destructors} \\ & & \quad \mathbf{out}_1 : X \longrightarrow Q_1(X); \\ & & \quad \ldots \\ & & \quad \mathbf{out}_n : X \longrightarrow Q_n(X) \\ & & \textbf{end}. \end{array}$$

Again we require that X occurs positively in all $Q_i(X)$ for $i = 1, \cdots, n$. Now the natural thing to do to code the above construction in polymorphism is to explore the duality of $\exists$ and $\forall$. In [19], the author uses the representation of $\exists$ by $\forall$. So a kind of 'double encoding' is present. In this paper we try to code up coinductive types by $\exists$ directly:

$$
\begin{aligned}
[Q(X)] &\stackrel{\text{def}}{=} (X \to Q_1(X)) \wedge \cdots \wedge (X \to Q_n(X)) \\
\nu X.Q(X) &\stackrel{\text{def}}{=} \exists X.[Q(X)] \wedge X \\
\eta_Q &\stackrel{\text{def}}{=} \Lambda X.\lambda q{:}[Q(X)].\lambda x{:}X.\delta(X, \langle q, x\rangle) : \forall X.[Q(X)] \to X \to \nu X.Q(X) \\
\mathsf{out}_i &\stackrel{\text{def}}{=} \lambda t{:}\nu X.Q(X).\mathbf{E}(t, (X, q).Q_i^\circ X \nu(\eta_Q X(\pi_1 q))(\pi_i(\pi_1 q)(\pi_2 q))) : (\nu X.Q(X)) \to Q_i(\nu X.Q(X)).
\end{aligned}
$$

The last term is obtained using the elimination rule for the existential quantifier:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{t : \nu, q : [Q(X)] \wedge X \vdash q : [Q(X)] \wedge X}{t : \nu, q : [Q(X)] \wedge X \vdash \pi_1 q : [Q(X)] \quad t : \nu, q : [Q(X)] \wedge X \vdash \pi_2 q : X}}{t : \nu, q : [Q(X)] \wedge X \vdash \pi_i(\pi_1 q)(\pi_2 q) : Q_i(X)}}{t : \nu, q : [Q(X)] \wedge X \vdash Q_i^\circ X \nu(\eta_Q X(\pi_1 q))(\pi_i(\pi_1 q)(\pi_2 q)) : Q_i(\nu)}}{t : \nu \vdash \mathbf{E}(t, (X, q).Q_i^\circ X \nu(\eta_Q X(\pi_1 q))(\pi_i(\pi_1 q)(\pi_2 q))) : Q_i(\nu)}}{\vdash \lambda t{:}\nu.\mathbf{E}(t, (X, q).Q_i^\circ X \nu(\eta_Q X(\pi_1 q))(\pi_i(\pi_1 q)(\pi_2 q))) : \nu \to Q_i(\nu)} .
$$

Proposition 3.1 *$\langle \nu X.Q(X), \mathsf{out}_1, \cdots, \mathsf{out}_n\rangle$ is a weakly terminal $[Q_1(X), \cdots, Q_n(X)]$-I-algebra.*

Proof: Given a $[Q_1(X), \cdots, Q_n(X)]$-I-algebra $\langle D, d_1, \cdots, d_n\rangle$, we need to show that there exists a term Coltr: $Q_i(D) \to Q_i(\nu X.Q(X))$ such that

$$
\begin{tikzcd}
D \arrow[r, "\mathsf{Coltr}"] \arrow[d, "d_i"'] & \nu X.Q(X) \arrow[d, "\mathsf{out}_i"] \\
Q_i(D) \arrow[r, "Q_i^\circ(\mathsf{Coltr})"'] & Q_i(\nu X.Q(X))
\end{tikzcd}
$$

commutes for all $i = 1, \cdots, n$. Define Coltr to be $\lambda x : D.\delta(D, \langle\langle d_1, \ldots, d_n\rangle, x\rangle)$. The calculation is simple:

$$
\begin{aligned}
\mathsf{out}_i \circ \mathsf{Coltr} &= \lambda x{:}D.\mathsf{out}_i(\mathsf{Coltr}(x)) \\
&= \lambda x{:}D.\mathsf{out}_i(\delta(D, \langle\langle d_1, \ldots, d_n\rangle, x\rangle)) \\
&= \lambda x{:}D.\mathbf{E}(\delta(D, \langle\langle d_1, \ldots, d_n\rangle, x\rangle), (X, q).Q_i^\circ X(\nu X.Q(X))(\eta_Q X(\pi_1 q))(\pi_i(\pi_1 q)(\pi_2 q))) \\
&= \lambda x{:}D.Q_i^\circ D(\nu X.Q(X))(\eta_Q D\langle d_1, \ldots, d_n\rangle)(d_i x) \\
&= \lambda x{:}D.Q_i^\circ D(\nu X.Q(X))(\lambda x{:}D.\delta(D, \langle\langle d_1, \ldots, d_n\rangle, x\rangle))(d_i x) \\
&= Q_i^\circ(\mathsf{Coltr}) \circ d_i.
\end{aligned}
$$

This completes the proof. ∎

Example 3.2 The data type of streams is defined as follows:

$$
\begin{array}{rcl}
\mathrm{Stream}(A) & = & \textbf{product } X \textbf{ with destructors} \\
 & & \quad \textbf{head} : X \longrightarrow A; \\
 & & \quad \textbf{tail} : X \longrightarrow X \\
 & & \textbf{end}.
\end{array}
$$

It corresponds to the following type and terms in $\mathcal{F}_2$:

$$
\begin{array}{rcl}
\nu X.Q(X) & \stackrel{\mathrm{def}}{=} & \exists X.(X \to A) \wedge (X \to X) \wedge X \\
\mathsf{head}(s) & \stackrel{\mathrm{def}}{=} & \mathbf{E}(s, (X, q).\pi_1(\pi_1 q)(\pi_2 q)) \\
\mathsf{tail}(s) & \stackrel{\mathrm{def}}{=} & \mathbf{E}(s, (X, q).\delta(X, \langle \pi_1 q, \pi_2(\pi_1 q)(\pi_2 q)\rangle)).
\end{array}
$$

One should compare this example to the encoding via $\forall$. ■

3.2 Coding Mutually Dependent Coinductive Types

We can also code up the mutually dependent coinductive types in polymorphism. The encodings are:

$$
\begin{array}{rcl}
\mathcal{Q}(\vec{X}) & \stackrel{\mathrm{def}}{=} & (X_1 \to Q_{11}(\vec{X})) \wedge \cdots \wedge (X_1 \to Q_{1n_1}(\vec{X})) \wedge \cdots \wedge (X_n \to Q_{n1}(\vec{X})) \wedge \cdots \wedge (X_n \to Q_{nn_n}(\vec{X})) \\
\nu_i \vec{X}.\mathcal{Q}(\vec{X}) & \stackrel{\mathrm{def}}{=} & \exists \vec{X}.\mathcal{Q}(\vec{X}) \wedge X_i \\
\eta^i_{\mathcal{Q}} & \stackrel{\mathrm{def}}{=} & \Lambda \vec{X}.\lambda q{:}\mathcal{Q}(\vec{X}).\lambda x{:}X_i.\delta(X_1, \delta(X_2, \ldots, \delta(X_n, \langle q, x\rangle)\cdots)) : \forall \vec{X}.\mathcal{Q}(\vec{X}) \to X_i \to \nu_i \vec{X}.\mathcal{Q}(\vec{X}) \\
\mathsf{out}_{ij} & \stackrel{\mathrm{def}}{=} & \lambda t{:}\nu_i \vec{X}.\mathcal{Q}(\vec{X}).\mathbf{E}(t, (\vec{X}, q).Q^{\circ}_{ij}\vec{X}\vec{\nu}(\eta^1_{\mathcal{Q}}\vec{X}(\pi_1 q))\cdots(\eta^n_{\mathcal{Q}}\vec{X}(\pi_1 q))(\pi_{ij}(\pi_1 q)(\pi_2 q))) : \nu_i \to Q_{ij}(\nu_i).
\end{array}
$$

Abbreviate $Q^{\circ}_{ij}\vec{X}\vec{\nu}(\eta^1_{\mathcal{Q}}\vec{X}(\pi_1 q))\cdots(\eta^n_{\mathcal{Q}}\vec{X}(\pi_1 q))(\pi_{ij}(\pi_1 q)(\pi_2 q))$ to ϕ_{ij}, the right hand side of the last definition stands for

$$
\mathbf{E}(t, (X_1, t_1).\mathbf{E}(t_1, (X_2, t_2).\cdots \mathbf{E}(t_{n-1}, (X_n, q).\phi_{ij})\cdots))
$$

which is obtained by the following derivations:

$$
\frac{t_{n-1} : \exists X_n.\mathcal{Q}(\vec{X}) \wedge X_i, q : \mathcal{Q}(\vec{X}) \wedge X_i \vdash \phi_{ij} : Q_{ij}(\nu_i)}{t_{n-1} : \exists X_n.\mathcal{Q}(\vec{X}) \wedge X_i \vdash \mathbf{E}(t_{n-1}, (X_n, q).\phi_{ij}) : Q_{ij}(\nu_i)}
$$

$$
\frac{t_{n-2} : \exists X_{n-1}\exists X_n.\mathcal{Q}(\vec{X}) \wedge X_i, t_{n-1} : \exists X_n.\mathcal{Q}(\vec{X}) \wedge X_i \vdash \mathbf{E}(t_{n-1}, (X_n, q).\phi_{ij}) : Q_{ij}(\nu_i)}{t_{n-2} : \exists X_{n-1}\exists X_n.\mathcal{Q}(\vec{X}) \wedge X_i \vdash \mathbf{E}(t_{n-2}, (X_{n-1}, t_{n-1}).\mathbf{E}(t_{n-1}, (X_n, q).\phi_{ij})) : Q_{ij}(\nu_i)}
$$

$$
\vdots
$$

$$
\frac{t : \exists X_1.\cdots \exists X_n.\mathcal{Q}(\vec{X}) \wedge X_i, t_1 : \exists X_2.\cdots \exists X_n.\mathcal{Q}(\vec{X}) \wedge X_i \vdash \mathbf{E}(t_1, (X_2, t_2).\cdots \mathbf{E}(t_{n-1}, (X_n, q).\phi_{ij})\cdots)}{t : \nu_i \vdash \mathbf{E}(t, (X_1, t_1).\mathbf{E}(t_1, (X_2, t_2).\cdots \mathbf{E}(t_{n-1}, (X_n, q).\phi_{ij})\cdots)) : Q_{ij}(\nu_i)}.
$$

Proposition 3.3 $\langle(\nu_1 \vec{X}.\mathcal{Q}(\vec{X}), \ldots, \nu_n \vec{X}.\mathcal{Q}(\vec{X})), \mathsf{out}_{11}, \ldots, \mathsf{out}_{nn_n}\rangle$ *is a weakly terminal* I^n*-algebra.*

Proof: Given an I^n-algebra $\langle (D_1,\ldots,D_n), d_{11},\ldots,d_{nn_n}\rangle$. Let

$$\mathsf{Coltr}_i \stackrel{\text{def}}{=} \lambda x{:}D_i.\delta(D_1,\delta(D_2,\ldots,\delta(D_n,\langle\langle d_{11},\ldots,d_{nn_n}\rangle,x\rangle)\cdots)).$$

We want to show that the following diagram commutes.

$$\begin{tikzcd} D_i \arrow[r, "\mathsf{Coltr}_i"] \arrow[d, "d_{ij}"'] & \nu_i\vec{X}.\mathcal{Q}(\vec{X}) \arrow[d, "\mathsf{out}_{ij}"] \\ Q_{ij}(D_i) \arrow[r, "Q^{\circ}_{ij}(\mathsf{Coltr})"'] & Q_{ij}(\nu_i\vec{X}.\mathcal{Q}(\vec{X})) \end{tikzcd}$$

This can be verified as follows:

$$\begin{aligned}
\mathsf{out}_{ij}\circ\mathsf{Coltr}_i &= \lambda x{:}D_i.\mathsf{out}_{ij}(\mathsf{Coltr}_i(x)) \\
&= \lambda x{:}D_i.\mathsf{out}_{ij}(\delta(D_1,\delta(D_2,\ldots,\delta(D_n,\langle\langle d_{11},\ldots,d_{nn_n}\rangle,x\rangle)\cdots))) \\
&= \lambda x{:}D_i.\mathbf{E}(\delta(D_1,\delta(D_2,\ldots,\delta(D_n,\langle\langle d_{11},\ldots,d_{nn_n}\rangle,x\rangle)\cdots)), \\
&\quad (\vec{X},q).Q^{\circ}_{ij}\vec{X}\vec{\nu}(\eta^1_{\mathcal{Q}}\vec{X}(\pi_1 q))\cdots(\eta^n_{\mathcal{Q}}\vec{X}(\pi_1 q))(\pi_{ij}(\pi_1 q)(\pi_2 q))) \\
&= \lambda x{:}D_i.Q^{\circ}_{ij}\vec{D}\vec{\nu}(\eta^1_{\mathcal{Q}}D_1\langle d_{11},\ldots,d_{nn_n}\rangle)\cdots(\eta^n_{\mathcal{Q}}D_n\langle d_{11},\ldots,d_{nn_n}\rangle)(d_{ij}x) \\
&= \lambda x{:}D_i.Q^{\circ}_{ij}\vec{D}\vec{\nu}(\lambda x{:}D_1.\delta(D_1,\delta(D_2,\ldots,\delta(D_n,\langle\langle d_{11},\ldots,d_{nn_n}\rangle,x\rangle)\cdots)))\cdots \\
&\quad (\lambda x{:}D_n.\delta(D_1,\delta(D_2,\ldots,\delta(D_n,\langle\langle d_{11},\ldots,d_{nn_n}\rangle,x\rangle)\cdots))) \\
&= Q^{\circ}_{ij}(\mathsf{Coltr}_1,\ldots,\mathsf{Coltr}_n)\circ d_{ij}
\end{aligned}$$

This completes the proof. ■

We should mention that these dual encodings are far less satisfactory than those in the inductive case. It remains to be seen if they are of any use.

4 Acknowledgement

The material in this paper is taken from the author's Ph.D thesis ([7]). My supervisor David Rydeheard has been very helpful all the way over the last two years. My thanks to him.

During the preparation of this paper, the author was supported by the following person, organizations and project: Yu-Gang Bau, the British Council, the State Education Commission of China and the CLICS-II project.

References

[1] C. Böhm and A. Berarducci. Automatic synthesis of typed λ-program on term algebras. *Theoretical Computer Science*, 39:135–154, 1985.

[2] J. Cartmell. Generalized algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209–243, 1986.

[3] T. Coquand and G. Huet. Constructions: A higher order proof system for mechanizing mathematics. In *EUROCAL85, LNCS203*, pages 151–184. Springer-Verlag, 1985.

[4] P. Dybjer. Inductively defined sets in Martin-Löf's set theory. In *Proceedings of the Workshop on General Logic*, Edinburgh, 1988. ECS-LFCS-88-52.

[5] P. Freyd. POLYNAT in PER. In J. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, pages 67–68, Boulder, June 89. A.M.S.

[6] P. Freyd, E. Robinson, and G. Rosolini. *Dinaturality for free*, 1992.

[7] Yuxi Fu. *Some Semantic Issues In Type Theory.* PhD thesis, Department of Computer Science, University of Manchester, May 1992.

[8] J. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1989.

[9] H. Goguen and Z. Luo. Inductive data types: Well-ordering types revisited. Technical report, LFCS, University of Edinburgh, April 1992.

[10] T. Hagino. *A Categorical Programming Language.* PhD thesis, LFCS, University of Edinburgh, 1987.

[11] T. Hagino. A typed lambda calculus with categorical type constructions. In D. Pitt, A. Poigné, and D. Rydeheard, editors, *Category Theory and Computer Science, LNCS 283*, pages 140–157. Springer-Verlag, 1987.

[12] M. Hyland, E. Robinson, and G. Rosolini. Algebraic types in per models. In M. Main et al, editor, *Mathematical Foundations of Programming Semantics, LNCS 442*, pages 333–350. Springer-Verlag, 1990.

[13] N. Mendler. *Inductive Definition in Type Theory.* PhD thesis, Cornell University, 1987.

[14] N. Mendler. Inductive types and type constraints in the second-order lambda calculus. *Annals of Pure and Applied Logic*, 51:159–172, 1991.

[15] N. Mendler. Predicative type universes and primitive recursion. In *Proceedings of the Sixth Symposium on Logic in Computer Science*, pages 173–184. IEEE Computer Science Press, July 1991.

[16] C. Ore. The extended calculus of constructions (ECC) with inductive types. *Information and Computation*, 99:231–264, August 1992.

[17] F. Pfenning and C. Paulin-Mohring. Inductively defined types in the calculus of constructions. In M. Main et al., editor, *Mathematical Foundations of Programming Semantics, LNCS442*, pages 209–228. Springer-Verlag, 1991.

[18] W. Phoa. Two results on set-theoretic polymorphism. In D. Pitt, P. Curien, Abramsky, A. Pitts, A. Poigné, and D. Rydeheard, editors, *Category Theory and Computer Science, LNCS 530*, pages 219–235, Paris, September 1991. Springer Verlag.

[19] G. Wraith. A note on categorical datatypes. In D. Pitt, Rydeheard, P. Dybier, A. Pitts, and A. Poigné, editors, *Category Theory and Computer Science, LNCS 389*, pages 213–223, Manchester, September 1989. Springer Verlag.