Representations of first order function types as terminal coalgebras

Thorsten Altenkirch
txa@cs.nott.ac.uk

School of Computer Science and Information Technology
University of Nottingham, UK

Abstract. We show that function types which have only initial algebras for regular functors in the domains, i.e. first order function types, can be represented by terminal coalgebras for certain nested functors. The representation exploits properties of ω^{op}-limits and local ω-colimits.

1 Introduction

The work presented here is inspired by discussions the author had some years ago with Healfdene Goguen in Edinburgh on the question *Can function types be represented inductively?* or maybe more appropriately: *Can function types be represented algebraically?*.

In programming and type theory the universe of types can be divided as follows:

- function types (cartesian closure)
- algebraic types
  - inductive types (initial algebras)
  - coinductive types (terminal coalgebras)

In programming the difference between inductive and coinductive types is often obliterated because one is mainly interested in the collection of partial objects of a certain type. Inspired by Occam's razor it would be interesting if we could explain one class of types by another. Here we try to reduce function types to algebraic types.

The first simple observation is that function spaces can be eliminated using products if the domain is finite. Here we show that function spaces $A \to B$ can be eliminated using coinductive types if the domain A is defined inductively. It is interesting to note that ordinary coinductive types are sufficient only for functions over linear inductive types (i.e. where the signature functor has the form $T(X) = A_1 \times X + A_0$) but in general we need to construct functors defined by terminal coalgebras in categories of endofunctors. Those correspond to nested or nested datatypes which have been the subject of recent work [BM98,AR99,Bla00].

1.1 Examples

We give some examples which are instances of our general construction , proposition 8. We use the usual syntax for products and coproducts and $\mu X.F(X)$ to denote initial algebras and $\nu X.F(X)$ for terminal coalgebras. See section 2 for the details. The isomorphisms stated below exist under the condition that the ambient category has the properties introduced later, see section 3. A category which satisfies these properties is the category of sets whose cardinality is less or equal $\aleph_1$. Note that this category is not cartesian closed.

Natural Numbers Natural numbers are given by $\mathrm{Nat} = \mu X.1 + X$, i.e. they are the initial algebra of the functor[1] $T(X) = 1 + X$.We have that

$$(\mu X.1 + X) \to B \simeq \nu Y.B \times Y,$$

where $\nu Y.B \times Y$ is the terminal coalgebra of the functor $T'(X) = B \times X$ — this is the type of streams or infinite lists over B.

Using the previous isomorphism we obtain a representation of countable ordinals using only algebraic types. The type $\mathrm{Ord} = \mu X.1 + X + (\mathrm{Nat} \to X)$ can be represented as

$$\mathrm{Ord} \simeq \mu X.1 + X + \nu Y.X \times Y$$

Note, however, that there is no representation for functions over Ord and hence there seems to be no representation of the next number class using only coinductive types.

Lists We assume as given a type A for which we already know how to construct function spaces. Then lists over A are given by $\mathrm{List}(A) = \mu X.1 + A \times X$ and we have

$$(\mu X.1 + A \times X) \to B \simeq \nu Y.B \times (A \to Y)$$

The right hand side defines B-labelled, A-branching non-wellfounded trees. Combining this with the first case we obtain a representation for functions over lists of natural numbers:

$$\begin{aligned} \mathrm{List}(\mathrm{Nat}) \to B &\simeq \nu Y.B \times \mathrm{Nat} \to Y \\ &\simeq \nu Y.B \times (\nu Z.Y \times Z) \end{aligned}$$

Binary Trees By considering functions over binary trees $\mathrm{BTree} = \mu X.1 + X \times X$ we leave the realm of linear inductive types. The type of functions over trees is given by:

$$(\mu X.1 + X \times X) \to B \simeq (\nu F.\Lambda X.X \times F(F(X)))(B)$$

[1] We only give the effect on objects since the morphism part of the functor can be derived from the fact that all the operations we use are functorial in their arguments, i.e. T can be extended on morphisms by $T(f) = 1 + f$.

Here the right hand side is read as the terminal coalgebra over the endofunctor $H(F) = \Lambda X.X \times F(F(X))$ on the category of endofunctors. There seem to be two ways to extend H to morphisms, i.e. given a natural transformation $\alpha \in F \to G$

$$H_1(\alpha)_A = \alpha_A \times (G(\alpha_A) \circ \alpha_{FA})$$
$$H_2(\alpha)_A = \alpha_A \times (\alpha_{GA} \circ F(\alpha_A))$$

However, it is easy to see that the naturality of α implies $H_1(\alpha) = H_2(\alpha)$.

The type $\nu F.\Lambda X.X \times F(F(X))$ has a straightforward representation in a functional programming language like Haskell which allows nested datatypes. A variation of this type, namely $\mu F.\Lambda X.1 + X \times F(F(X))$, is used in [BM98] as an example for nested datatypes under the name *Bushes*. We can represent $\nu F.\Lambda X.X \times F(F(X))$ as

```
data BTfun x  = Case x (BTfun (BTfun x))
```

Here we consider only total elements of a type which entails that we have to differentiate between inductive and coinductive interpretations of recursively defined type. We interpret `BTfun` coinductively, which is sensible since the inductive interpretation is empty.

We assume that binary trees `BT` are defined as

```
data BT = L | Sp BT BT
```

This time we interpret the type inductively!

The two parts of an isomorphism which we call `lamBT` and `appBT` can be programmed in Haskell [2] :

```
lamBT :: (BT → a) → BTfun a
lamBT f = Case (f L) (lamBT (λ t → lamBT (λ u → f (Sp t u))))

appBT :: BTfun a → BT → a
appBT (Case a f) t = case t of L → a
                               Sp tl tr → appBT (appBT f tl) tr
```

Since we use polymorphic recursion it is essential to give the types of the two functions explicitly.

Finite Trees As a last example we shall consider functions over finitely branching trees which are interesting because they are defined using *interleaving* inductive types, i.e.

$$\begin{aligned}\text{FTree} &= \mu X.\text{List}(X) \\ &= \mu X.\mu Y.1 + X \times Y\end{aligned}$$

The function type over finite trees can be represented as follows:

$$(\mu X.\mu Y.1 + X \times Y) \to B \simeq (\nu F.\nu G.\Lambda Z.Z \times F(G(Z)))(B)$$

[2] We take the liberty of writing λ for \ and $\to$ for `->`.

1.2 Related work

After having completed most of the technical work presented in this paper it was brought to our attention that Ralf Hinze had already discovered what amounts to essentially the same translation in the context of generic functional programming [Hin00b,Hin00a]. His motivation was of a more practical nature: he was looking for a generic representation of memo functions. One of the anonymous referees pointed out that this construction was anticipated in a note by Geraint Jones [Jon98].

The present paper can be viewed as providing a more detailed categorical underpinning of Hinze's construction. In some regards however, our approaches differ fundamentally:

- We adopt a categorical perspective in which functions are *total*, in that exponentiation is right adjoint to products, where Hinze deals with partial functions (and hence a monoidally closed structure).
- As a consequence of this we differentiate between inductive and coinductive types. It also means that we cannot use fixpoint induction (as suggested by Hinze) but have to rely on using ω-limits and colimits explicitly.
- We show the existence of the exponentials whereas Hinze only shows that they are isomorphic to already existing ones.
- Hinze's programs require 2nd order impredicative polymorphism whereas our construction takes place in a predicative framework (compare also section 6).

1.3 Acknowledgments

I would like to thank Healfdene Goguen for inspiring this line of research, Thomas Streicher for valuable help on categorical questions, Peter Hancock for interesting email discussions and for pointing out Ralf Hinze's work to me, and Roland Backhouse for discussions on the use of fusion in this context. Dirk Pattinson provided valuable feedback on a draft version. I would also like to thank the anonymous referees whose comments I tried to incorporate to the best of my abilities.

2 Preliminaries

We work with respect to some ambient category $\mathbb{C}$ whose properties we shall make precise below. We assume that $\mathbb{C}$ is bicartesian, i.e. has all finite products (written $\mathbf{1}, A \times B$) and finite coproducts (written $\mathbf{0}, A+B$). We write $!_A \in A \to \mathbf{1}$ and $?_A \in \mathbf{0} \to A$ for the universal maps. Notationally, we use $\to$ for homsets and $\Rightarrow$ for exponentials. We do not assume that $\mathbb{C}$ is cartesian closed.

We assume the existence of ω^{op}-limits and ω-colimits. Here ω stands for the posetal category $(\omega, \leq)$ and by ω^{op}-completeness we mean that limits of all functors $F \in \omega^{\mathrm{op}} \to \mathbb{C}$ exist, i.e.

$$A \to \lim(F) \simeq \Delta(A) \dot{\to} F$$

where $\Delta(A) = \Lambda X.A$ is the constant functor. We write $\pi_F \in \Delta(\lim(F)) \to F$ for the projection and for the back direction of the isomorphism: $\text{prod}_F(\alpha) \in A \to \lim(F)$ given $\alpha \in \Delta(A) \dot{\to} F$.

Dually, by ω-cocompleteness we mean that all colimits of functors $F \in \omega \to \mathbb{C}$ exist, i.e.

$$\text{colim}(F) \to A \simeq F \dot{\to} \Delta(A)$$

We write $\text{inj}_F \in F \dot{\to} \Delta(\text{colim}(F))$ for the injection and for the inverse $\text{case}(\alpha) \in \text{colim}(F) \to A$ given $\alpha \in F \dot{\to} \Delta(A)$.

A functor $T \in \mathbb{C} \to \mathbb{C}$ is called ω^{op}-continuous (ω-cocontinuous) if it preserves all ω^{op}-limits (ω-colimits) up to isomorphism. We write

$$\Psi_T \in T(\lim(F)) \simeq \lim(T \circ F)$$
$$\Psi^T \in T(\text{colim}(F)) \simeq \text{colim}(T \circ F)$$

It is easy to see that coproducts preserve colimits and products preserve limits and hence the appropriate operations on functors are (co-)continuous. We will later identify the precise circumstances under which products preserve colimits.

Given an endofunctor $T \in \mathbb{C} \to \mathbb{C}$ the category of T-algebras has as objects $(A \in \mathbb{C}, f \in T(A) \to A)$ and morphisms $h \in (A, f) \to (B, g)$ are given by $h \in A \to B$ s.t. $g \circ T(h) = h \circ f$. We denote the initial T-algebra by $(\mu T, \text{in}_T \in T(\mu T) \to \mu T)$. Given any T-algebra (A, f) the unique morphism (often called a *catamorphism*) is written as $\text{fold}_T(f) \in \mu T \to A$. Dually, the category of T-coalgebras has as objects $(A \in \mathbb{C}, f \in A \to T(A))$ and morphisms $h \in (A, f) \to (B, g)$ are given by $h \in A \to B$ s.t. $g \circ h = T(h) \circ f$. The terminal T-coalgebra is written as $(\nu T, \text{out}_T \in \nu T \to T(\nu T))$ and given a coalgebra (A, f) the unique morphism (often called *anamorphism*) is written $\text{unfold}_T(f) \in A \to \nu T$.

For completeness we review some material from [Ada74,PS78] Given an endofunctor $T \in \mathbb{C} \to \mathbb{C}$ and $i \in \omega$ we write $T^i \in \mathbb{C} \to \mathbb{C}$ for the ith iteration of T. We define $\text{Chain}_T \in \omega \to \mathbb{C}$ and $\text{Chain}^T \in \omega^{\text{op}} \to \mathbb{C}$:

$$\text{Chain}_T(i) = T^i(\mathbf{1})$$
$$\text{Chain}_T(i \leq j) = T^i(!_{\text{Chain}_T(j-i)})$$
$$\text{Chain}^T(i) = T^i(\mathbf{0})$$
$$\text{Chain}^T(i \geq j) = T^j(?_{\text{Chain}^T(i-j)})$$

Proposition 1 (Adamek,Plotkin-Smyth). *Given a ω^{op}-complete and ω-cocomplete category $\mathbb{C}$ and an an endofunctor $T \in \mathbb{C} \to \mathbb{C}$ we have that:*

1. *If T is ω-cocontinuous then the initial algebra exists and*

$$mu(T) \simeq \text{colim}(\text{Chain}^T).$$

2. *If T is ω-continuous then the terminal coalgebra exists and*

$$\nu(T) \simeq \lim(\text{Chain}_T)$$

3 Locality

Since we do not assume that our ambient category is closed we have to be more precise w.r.t coproducts, colimits and initial algebras. We require that all those concepts exist locally. Given an object Γ which corresponds to a type context the local category wrt. Γ has the same objects as $\mathbb{C}$ and as morphisms $f \in \Gamma \times A \to B$. The local identity is just the projection $\pi_2 \in \Gamma \times A \to A$ and composition of $f \in \Gamma \times A \to B$ and $g \in \Gamma \times B \to C$ is given by $g \circ (1, f) \in \Gamma \times A \to C$. We say that X are local , if X exists in all local categories and coincide with global X.

A local functor is given by a function on objects and a natural transformation:

$$\mathrm{st}^T \in (\Gamma \times A \to B) \dot{\to} (\Gamma \times T(A) \to T(B))$$

natural in Γ which preserves local identity and composition:

$$\begin{aligned} \mathrm{st}^T(\pi_2) &= \pi_2 \\ \mathrm{st}^T(g \circ (1, f)) &= \mathrm{st}^T(g) \circ (1, \mathrm{st}^T(f)) \\ &\quad \text{where } f \in \Gamma \times B \to C \text{ and } g \in \Gamma \times A \to B. \end{aligned}$$

Alternatively this can be formalized via a natural transformation

$$\theta^T_{\Gamma,A} \in \Gamma \times T(A) \dot{\to} T(\Gamma \times A)$$

subject to the appropriate conditions but this can easily be seen to be equivalent.

Traditionally, local functors are called strong [CS92]. We diverge from this use because we want to apply the idea of locality also to other concepts like colimits and coalgebras and here the word strong is already used to signal that the uniqueness condition holds.

Proposition 2.

1. *Products are local.*
2. *ω-limits are local.*
3. *Terminal coalgebras of local functors are local.*

Proof. (Sketch): 3. Given a local T-coalgebra $f \in \Gamma \times A \to T(A)$ the local unfold is given by

$$\begin{aligned} &\mathrm{unfold}^*_T(f) \in \Gamma \times A \to \nu T \\ &\mathrm{unfold}^*_T(f) = \mathrm{unfold}_T(\theta^T_{\Gamma,A} \circ (1, f)) \end{aligned}$$

However, the same does not hold for coproducts, colimits or initial algebras. E.g. coproducts are not local in $\mathrm{CPO}_\perp$. This asymmetry is caused by the fact that the notion of local morphisms is not self dual.

Local coproducts are given by the following families of isomorphisms:

$$\begin{aligned} \Gamma \times \mathbf{0} &\to X \simeq \mathbf{1} \\ \Gamma \times (A + B) &\to X \simeq (\Gamma \times A \to X) \times (\Gamma \times B \to X) \end{aligned}$$

natural in Γ. Given a functor $F \in \omega^{\mathrm{op}} \to \mathbb{C}$ (not necessary local) the ω-colimit is local if the following family of isomorphisms exist:

$$\Gamma \times \mathrm{colim}(F) \to C \simeq \Delta(\Gamma) \times F \dot{\to} \Delta(C)$$

We say that a functor is locally cocontinuous if it preserves local colimits and again it is easy to see that local coproducts preserve local colimits. In the special case of local ω-colimits we also have

Proposition 3. *Products preserve local ω-colimits:*

$$\mathrm{colim}(F) \times \mathrm{colim}(G) \simeq \mathrm{colim}(F \times G)$$

Proof. (Sketch) For simplicity we only consider the case of $\Gamma = \mathbf{1}$. Using locality we show that

$$\mathrm{colim}(F) \times \mathrm{colim}(G) \to A \simeq (\Lambda(i,j) \in \omega \times \omega . F(i) \times G(j)) \dot{\to} \Delta(A)$$

Using the fact that either $i \leq j$ or $j \leq i$ we can show that the right hand side is isomorphic to

$$(\Lambda i \in \omega . F(i) \times G(i)) \dot{\to} \Delta(A)$$

Assuming that T is a local endofunctor, a local T-algebra with respect to Γ is given by $f \in \Gamma \times T(A) \to A$ and given another local algebra $g : \Gamma \times T(B) \to B$ then a morphism $h \in f \to g$ is given by $h \in \Gamma \times A \to B$ s.t. $h \circ (1, f) = g \circ (1, \mathrm{st}(h))$. Saying that an initial algebra $(\mu T, \mathrm{in}_T \in T(\mu T) \to \mu T)$ is local means that $\mathrm{in}_T \circ \pi_2 \in \Gamma \times T(\mu T) \to \mu T$ is an initial local T-algebra.

Definition 1. *We call a category $\mathbb{C}$ locally ω-bicomplete if the following conditions hold:*

1. $\mathbb{C}$ has all finite products.
2. $\mathbb{C}$ is ω^{op}-complete, i.e. it has all ω^{op}-limits.
3. $\mathbb{C}$ has all local finite coproducts.
4. $\mathbb{C}$ is locally ω-cocomplete, i.e. it has all local ω-colimits.

We assume that the ambient category $\mathbb{C}$ is locally ω-complete. We note that the initial algebra representation theorem can be localized:

Proposition 4. *Given a cocontinuous local endofunctor T: Then in the presence of local ω-colimits the representation of proposition 1 gives rise to a local initial algebra.*

Finally, we remark that the reason that the question of locality has so far got only very little attention in programming language theory is because here the ambient category is usually assumed to be cartesian closed and we have:

Proposition 5. *Assuming that our ambient category $\mathbb{C}$ is cartesian closed we have*

1. *Coproducts are local.*
2. *ω-colimits are local.*
3. *Initial algebras of local functors are local.*

Proof. (Sketch): 3. Let

$$\begin{aligned} &\overline{\mathrm{app}}_{\Gamma,A} \in \Gamma \times (\Gamma \Rightarrow A) \to A \\ &\overline{\lambda}_{\Gamma,A}(f) \in B \to (\Gamma \Rightarrow A) \qquad\qquad \text{given } f \in \Gamma \times B \to A. \end{aligned}$$

be twisted versions of the usual morphisms. Now, given $f \in \Gamma \times T(A) \to A$ we define

$$\begin{aligned} \mathrm{fold}^*_T(f) &= \overline{\mathrm{app}}(\mathrm{fold}_T(\overline{\lambda}(f \circ \mathrm{st}(\overline{\mathrm{app}})))) \\ &\in \Gamma \times \mu(T) \to A \end{aligned}$$

4 The μ-ν property

We shall now establish the main technical lemma of this paper which relates function spaces whose domains are initial algebras to terminal coalgebras. We say that an object $A \in \mathbb{C}$ is exponentiable if for all $B \in \mathbb{C}$: $A \Rightarrow B$ exists and there is an isomorphism

$$\Gamma \times A \to B \simeq \Gamma \to A \Rightarrow B$$

which is natural in Γ. We define $\mathbb{C}^*$ as the full subcategory of exponentiable objects.

Given a functor $F \in \omega^{\mathrm{op}} \to \mathbb{C}^*$ and an object $C \in \mathbb{C}$ we define

$$\begin{aligned} F \Rightarrow C &\in \omega \to \mathbb{C} \\ (F \Rightarrow C)(i) &= F(i) \Rightarrow C \end{aligned}$$

Note that $F \Rightarrow C \neq F \Rightarrow \Delta(C)$.

Lemma 1.

$$\Gamma \times \mathrm{colim}(F) \to C \simeq \Gamma \to \lim(F \Rightarrow C)$$

natural in Γ.

Proof. Straightforward unfolding of definitions.

Note that local colimits are essential here. We also know that limits in functor categories can be calculated pointwise:

Lemma 2. *Let $F \in \omega \to (\mathbb{C} \Rightarrow \mathbb{C})$ then we have*

$$\lim(F)(C) \simeq \lim(\Lambda i.F(i, C))$$

natural in C.

Lemma 3. *Given an ω-cocontinuous local functor $F \in \mathbb{C}^* \to \mathbb{C}^*$ which preserves exponentiability and an ω-continuous functor $G \in (\mathbb{C} \Rightarrow \mathbb{C}) \to (\mathbb{C} \Rightarrow \mathbb{C})$ s.t. for all exponentiable objects A*

$$F(A) \Rightarrow B \simeq G(\Lambda X.A \Rightarrow X)(B) \tag{H}$$

which is natural in B then we have:

1. *For all $i \in \omega$:*
$$\mathrm{Chain}^F(i) \Rightarrow C \simeq \mathrm{Chain}_G(i)(C)$$
natural in C.
2. $$\Gamma \times \mu(F) \to C \simeq \Gamma \to (\nu G)(C)$$
natural in Γ, C.

Proof.

1. By induction over i:
   0
$$\begin{aligned}
\mathrm{Chain}^F(0) \Rightarrow C &= \mathbf{0} \Rightarrow C \\
&\simeq \mathbf{1} && \text{Since } \mathbf{0} \text{ is local.} \\
&= \mathrm{Chain}_G(0)(C) && \text{Since } \mathrm{Chain}_G(0) = \Delta(\mathbf{1}).
\end{aligned}$$
   $i+1$
$$\begin{aligned}
\mathrm{Chain}^F(i+1) \Rightarrow C &= F(\mathrm{Chain}^F(i)) \Rightarrow C \\
&\simeq G(\Lambda X.\mathrm{Chain}^F(i) \Rightarrow X)(C) && \text{(H)} \\
&\simeq G(\mathrm{Chain}_G(i))(C) && \text{ind.hyp.} \\
&\simeq \mathrm{Chain}_G(i+1)(C)
\end{aligned}$$
2.
$$\begin{aligned}
\Gamma \times \mu(F) \to C &\simeq \Gamma \times \mathrm{colim}(\mathrm{Chain}^F) \to C && \text{by prop. 4.} \\
&\simeq \Gamma \to \lim(\mathrm{Chain}^F \Rightarrow C) && \text{by lemma 1.} \\
&\simeq \Gamma \to \lim(\mathrm{Chain}_G)(C) && \text{by 1.} \\
&\simeq \Gamma \to \nu(G)(C) && \text{by prop. 1.}
\end{aligned}$$

5 The representation theorem

We will now establish that function spaces whose domain is an inductive regular type can be isomorphically represented by coinductive nested types.

The set of inductive regular functors of arity n: $\mathcal{IND}_n \subseteq \mathbb{C}^n \to \mathbb{C}$ is inductively defined by the following rules:

$$\frac{0 \leq i < n}{\Lambda \boldsymbol{X}.X_i \in \mathcal{IND}_n} \qquad \frac{}{\Lambda \boldsymbol{X}.\mathbf{0}, \Lambda \boldsymbol{X}.\mathbf{1} \in \mathcal{IND}_n}$$

$$\frac{F, G \in \mathcal{IND}_n}{\Lambda \boldsymbol{X}.F(\boldsymbol{X}) + G(\boldsymbol{X}), \Lambda \boldsymbol{X}.F(\boldsymbol{X}) \times G(\boldsymbol{X}) \in \mathcal{IND}_n}$$

$$\frac{F \in \mathcal{IND}_{n+1}}{\Lambda \boldsymbol{X}.\mu Y.F(\boldsymbol{X}, Y) \in \mathcal{IND}_n}$$

An inductive regular type is just an inductive regular functor of arity 0.

Proposition 6. *All inductive regular functors are local and locally ω-cocontinuous.*

Proof. (Sketch): By induction over the structure of $\mathcal{IND}$. Locality simply follows from the fact that we use local coproducts and colimits and that projections and products are local anyway.

ω-cocontinuity follows from the fact that local coproducts preserve colimits and local initial algebras preserve local colimits since they correspond to local colimits (proposition 4). The case of products is covered by proposition 3.

We now define the set of coinductive nested functors of arity n: $\mathcal{COIND}_n \subseteq (\mathbb{C} \Rightarrow \mathbb{C})^n \to \mathbb{C} \Rightarrow \mathbb{C}$ inductively:

$$\frac{0 \leq i < n}{\Lambda \boldsymbol{F}.F_i \in \mathcal{COIND}_n} \qquad \frac{}{\Lambda \boldsymbol{F}, X.\mathbf{1}, \Lambda \boldsymbol{F}, \Lambda X.X \in \mathcal{COIND}_n}$$

$$\frac{G, H \in \mathcal{COIND}_n}{\Lambda \boldsymbol{F}, X.G(\boldsymbol{F}, X) \times H(\boldsymbol{F}, X), \Lambda \boldsymbol{F}.G(\boldsymbol{F}) \circ G(\boldsymbol{F}) \in \mathcal{COIND}_0}$$

$$\frac{G \in \mathcal{COIND}_{n+1}}{\Lambda \boldsymbol{F}.\nu H.G(\boldsymbol{F}, H) \in \mathcal{COIND}_n}$$

A coinductive nested type is a coinductive nested functor of arity 0 applied to any type (i.e. definable object).

Proposition 7. *All coinductive nested functors are ω^{op}-continuous.*

Proof. (Sketch): Follows from the fact that products and limits preserve limits.

We now assign to every inductive regular functor $F \in \mathcal{IND}_n$ a coinductive nested type $\hat{F} \in \mathcal{COIND}_n$ which represents the function space in the sense made precise below.

$$\begin{array}{ll}
F(\boldsymbol{X}) = X_i & \hat{F}(\boldsymbol{H}) = H_i \\
F(\boldsymbol{X}) = \mathbf{0} & \hat{F}(\boldsymbol{H}) = \Lambda X.\mathbf{1} \\
F(\boldsymbol{X}) = F_1(\boldsymbol{X}) + F_2(\boldsymbol{X}) & \hat{F}(\boldsymbol{H}) = \Lambda X.\hat{F}_1(\boldsymbol{H}) \times \hat{F}_2(\boldsymbol{H}) \\
F(\boldsymbol{X}) = \mathbf{1} & \hat{F}(\boldsymbol{H}) = \Lambda X.X \\
F(\boldsymbol{X}) = F_1(\boldsymbol{X}) \times F_2(\boldsymbol{X}) & \hat{F}(\boldsymbol{H}) = \hat{F}_1(\boldsymbol{H}) \circ \hat{F}_2(\boldsymbol{H}) \\
F(\boldsymbol{X}) = \mu Y.F'(\boldsymbol{X}, Y) & \hat{F}(\boldsymbol{H}) = \nu G.\hat{F}'(\boldsymbol{H}, G)
\end{array}$$

Proposition 8. *Given $F \in \mathcal{IND}_n$ and $\boldsymbol{A} \in \mathbb{C}^*$ define $H_i = \Lambda X.A_i \Rightarrow X$. We have that*

$$\Gamma \times F(\boldsymbol{A}) \to B \simeq \Gamma \to \hat{F}(\boldsymbol{H}, B)$$

which is natural in B

Proof. By induction over the structure of F:

$F((X)) = X_i$

$$\begin{aligned}
\Gamma \times F(\boldsymbol{A}) \to B &= \Gamma \times A_i \to B \\
&\simeq \Gamma \to A_i \Rightarrow B \\
&= \Gamma \to \hat{F}(\boldsymbol{H}, B)
\end{aligned}$$

$F(\boldsymbol{X}) = \mathbf{0}$

$$\begin{aligned}
\Gamma \times F(\boldsymbol{A}) \to B &= \Gamma \times 0 \to B \\
&\simeq \Gamma \to \mathbf{1} && \text{strong initial object} \\
&= \Gamma \to \hat{F}(\boldsymbol{H}, B)
\end{aligned}$$

$F(\boldsymbol{X}) = F_1(\boldsymbol{X}) + F_2(\boldsymbol{X})$

$$\begin{aligned}
\Gamma \times F(\boldsymbol{A}) \to B &= \Gamma \times F_1(\boldsymbol{A}) + F_2(\boldsymbol{A}) \to B \\
&\simeq (\Gamma \times F_1(\boldsymbol{A}) \to B) \times (\Gamma \times F_2(\boldsymbol{A}) \to B) && \text{strong coproducts} \\
&\simeq (\Gamma \to \hat{F}_1(\boldsymbol{H}, B)) \times (\Gamma \to \hat{F}_2(\boldsymbol{H}, B)) && \text{ind.hyp.} \\
&\simeq \Gamma \to \hat{F}_1(\boldsymbol{H}, B) \times \hat{F}_2(\boldsymbol{H}, B) \\
&= \Gamma \to \hat{F}(\boldsymbol{H}, B)
\end{aligned}$$

$F(\boldsymbol{X}) = \mathbf{1}$

$$\begin{aligned}
\Gamma \times F(\boldsymbol{A}) \to B &= \Gamma \times \mathbf{1} \to B \\
&\simeq \Gamma \to B \\
&= \Gamma \to \hat{F}(\boldsymbol{H}, B)
\end{aligned}$$

$F(\boldsymbol{X}) = F_1(\boldsymbol{X}) \times F_2(\boldsymbol{X})$

$$\begin{aligned}
\Gamma \times F(\boldsymbol{A}) \to B &= \Gamma \times F_1(\boldsymbol{A}) \times F_2(\boldsymbol{A}) \to B \\
&\simeq \Gamma \times F_1(\boldsymbol{A}) \to F_2(\boldsymbol{A}) \Rightarrow B \\
&\simeq \Gamma \to F_1(\boldsymbol{A}) \Rightarrow F_2(\boldsymbol{A}) \Rightarrow B \\
&\simeq \Gamma \to \hat{F}_1(\boldsymbol{H}, F_2(\boldsymbol{A})) \Rightarrow B) && \text{ind.hyp}(F_1) \\
&\simeq \Gamma \to \hat{F}_1(\boldsymbol{H}, \hat{F}_2(\boldsymbol{H}, B)) && \text{ind.hyp}(F_2) \\
&= \Gamma \to (\hat{F}_1(\boldsymbol{H}) \circ \hat{F}_2(\boldsymbol{H}))(B) \\
&= \Gamma \to \hat{F}(\boldsymbol{H}, B)
\end{aligned}$$

$F(\boldsymbol{X}) = \mu Y.F'(\boldsymbol{X}, Y)$

$$\begin{aligned}
\Gamma \times F(\boldsymbol{A}) \to B &= \Gamma \times \mu Y.F'(\boldsymbol{A}, Y) \to B \\
&\simeq \Gamma \to (\nu G.\hat{F}'(\boldsymbol{H}, G))(B) && (*) \\
&= \Gamma \to \hat{F}(\boldsymbol{H}, B)
\end{aligned}$$

To justify (*) we apply lemma 3,2. to $\Lambda Y.F'(\boldsymbol{A}, Y)$ and $\Lambda G.\hat{F}'(\boldsymbol{H}, G)$. Preservation of exponentials and (H) follows from the ind.hyp.

Corollary 1. *Every function space $A \Rightarrow B$ where $A \in \mathcal{IND}_0$ is an inductive regular type can be represented as a coinductive nested type $\hat{A}(B)$.*

6 Using Fusion ?

Roland Backhouse remarked that the central lemma 3 could be proven using the fusion theorem of [BBvGvdW96], pp.76:

Proposition 9 (Fusion). *Given a left adjoint functor $F \in \mathbb{C} \to \mathbb{D}$ and functors $G \in \mathbb{C} \to \mathbb{C}$ and $H \in \mathbb{D} \to \mathbb{D}$ s.t.*

$$F \circ G \simeq H \circ F$$

then

$$F(\mu(G)) \simeq \mu(H)$$

Using

$$\begin{aligned}
F \in \mathbb{C} &\Rightarrow (\mathbb{C} \Rightarrow \mathbb{C})^{\mathrm{op}} \\
F(X) &= \Lambda Y.X \Rightarrow Y
\end{aligned}$$

we may obtain lemma 3 as a corollary (w.o. requiring that the functors involved are continuous or cocontinuous) if we can show that F has a right adjoint. This right adjoint can be written as

$$\begin{aligned}
F^{\#} &\in (\mathbb{C} \Rightarrow \mathbb{C})^{\mathrm{op}} \to \mathbb{C} \\
F^{\#}(G) &= G \dot{\to} \lambda X.X
\end{aligned}$$

This requires that there is an internal representation of $G \dot{\rightarrow} \lambda X.X$ which depends on impredicative quantification as present in the Calculus of Constructions.

There is a very close connection between the construction sketched above and the Haskell programs (section 1.1). The programs seem not to use impredicative quantification explicitly because this is hidden by polymorphic recursion. However, if we attempt to present e.g. `appBT` using categorical combinators (e.g. `fold`) there seems to be no way to avoid impredicative polymorphism (which also has the consequence that this cannot be encoded in the current Haskell type system).

This also raises the question whether explaining polymorphic recursion which arises naturally when using nested types does in some natural cases require impredicative polymorphism. The specific case considered here shows that impredicativity can be avoided by using ω-completeness properties. It may be the case that similar explanations can be found for all sensible applications of polymorphic recursion.

7 Further work

There is a certain asymmetry in our construction: we construct function types of regular (inductive) types using nested (coinductive) types. It seems natural to ask what happens if we look at nested inductive types in the domain. It seems reasonable to look at functors definable in a simply typed language where type constructors like $\times$ or μ are just basic constants. The construction presented here can be generalized to this case (which we may call higher dimensional nested types). We plan to present details of this in a forthcoming paper.

In the current form our result is not applicable to categories of constructive functions like ω-Set. However, it seems likely that our result still holds when moving to an appropriate internal notion of limits and colimits.

The categorical features used here, e.g. initial and terminal algebras but no function types can be syntactically encoded in a calculus which for obvious reasons does not deserve the name λ-calculus. We believe that this calculus deserves further investigation because it represents the algorithms which can be defined using only algebraic types. It would be interesting to determine the precise proof-theoretic strength of this calculus which almost certainly exceeds that of first order arithmetic.

References

[Ada74] J. Adamek. Free algebras and automata realizations in the language of categories. *Comment. Math. Univ. Carolinae*, 15:589–602, 1974.

[AR99] T. Altenkirch and B. Reus. Monadic presentations of lambda terms using generalized inductive types. In *Computer Science Logic*, 1999.

[BBvGvdW96] R. Backhouse, R. Bijsterveld, R. van Geldrop, and J. van der Woude. Category theory as coherently constructive lattice theory. available from `http://www.cs.nott.ac.uk/~rcb/papers/papers.html`, December 1996. Working Document.

[Bla00] P. Blampied. *Structured recursion for non-uniform data-types*. PhD thesis, School of Computer Science and IT at the University of Nottingham, UK, 2000.

[BM98] R. Bird and L. Meertens. Nested datatypes. In J. Jeuring, editor, *Mathematics of Program Construction*, number 1422 in LNCS, pages 52 – 67. Springer Verlag, 1998.

[CS92] J. R. B. Cockett and D. Spencer. Strong categorical datatypes I. In R. A. G. Seely, editor, *Proceedings Intl. Summer Category Theory Meeting, Montréal, Québec, 23–30 June 1991*, volume 13 of *Canadian Mathematical Society Conf. Proceedings*. American Mathematical Society, 1992.

[Hin00a] R. Hinze. Generalizing generalized tries. *Journal of Functional Programming*, 2000.

[Hin00b] R. Hinze. Memo functions, polytypically! In Johan Jeuring, editor, *Proceedings of the Second Workshop on Generic Programming, WGP 2000*, 2000.

[Jon98] G. Jones. Tabulation for type hackers. Available from `ftp://ftp.comlab.ox.ac.uk/`, 1998.

[PS78] G. D. Plotkin and M. B. Smyth. The category-theoretic solution of recursive domain equations. *SIAM Journal on Computing*, 11, 1978.