Containers

Constructing Strictly Positive Types

Michael Abbott [a], Thorsten Altenkirch [b], Neil Ghani [a]

[a] *Department of Mathematics and Computer Science, University of Leicester*

[b] *School of Computer Science and Information Technology, Nottingham University*

Abstract

We introduce container functors as a representation of data types providing a new conceptual analysis of data structures and polymorphic functions. Our development exploits Type Theory as a convenient way to define constructions within locally cartesian closed categories. We show that container morphisms can be full and faithfully interpreted as polymorphic functions (i.e. natural transformations) and that in the presence of W-types all strictly positive types (including nested inductive and coinductive types) give rise to containers.

Key words: Type Theory, Category Theory, Container Functors, W-Types, Induction, Coinduction, Initial Algebras, Final Coalgebras.

1 Introduction

One of the strengths of modern functional programming languages like Haskell or CAML is that they support recursive datatypes such as lists and various forms of trees. When reasoning about functional programs we commonly restrict our view to total functions and view types as sets. David Turner called this *elementary strong functional programming* (Turner, 1996), though *total* might have been a better word. Not all recursive types make sense in this view, e.g. we can hardly understand $D \cong D \to D$ as a set. Moreover, even if we restrict ourselves to well behaved types like lists over A which are a solution to $\mathrm{List}A \cong 1 + A \times \mathrm{List}A$, in the total setting we have to decide which fixpoint we mean. There are two canonical choices:

finite lists correspond to the initial algebra of the signature functor. We denote this as $\mathrm{List}A = \mu X.\, 1 + A \times X$.

potentially infinite lists correspond to the terminal coalgebra of the signature functor. We denote this as $\mathrm{List}^{\infty} A = \nu X.\, 1 + A \times X$.

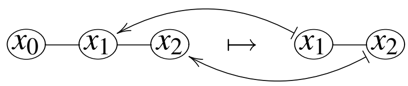
In this paper we investigate **strictly positive types**, these are types which can be formed using 0, 1, $+$, $\times$, $+$, $\rightarrow$, μ, ν with the restriction that types on the left side of the arrow have to be closed with respect to. type variables. Examples of strictly positive types are: natural numbers $\mathbb{N} \equiv \mu X.\, 1+X$, binary trees $\mathrm{BTree}A \equiv \mu X.\, A+X\times X$, streams $\mathrm{Stream}A \equiv \nu X.\, A\times X$, ordinal notations $\mathrm{Ord} \equiv \mu X.\, 1+X+\mathbb{N}\rightarrow X = \mu Y.\, 1+Y+(\mu X.\, 1+X)\rightarrow Y$, and Rose trees $\mathrm{RTree} \equiv \mu Y.\, \mathrm{List}Y = \mu Y.\, \mu X.\, 1+X\times Y$. Intuitively, these types can be understood as sets of trees (potentially infinitely branching), which have finite and infinite parts.

Our central insight is that all strictly positive types can be represented as **containers**, which can be viewed as a normal form for those types. A 1-ary container is given by a type of shapes S and a family of position types indexed by S: $s : S \vdash P\,s$, we write this as $(s : S \triangleright Ps)$ or just $(S \triangleright P)$. The extension of a container $[\![S \triangleright P]\!]$ is a functor, which on objects is given by $[\![S \triangleright P]\!]X = \sum s\!:\!S.\,(P\,s \rightarrow X)$, where $\sum$ is the infinite sum or dependent product. I.e. for a type X an element of $\sum s\!:\!S.\,(Ps \rightarrow X)$ is a pair (s, f) where $s : S$ is a shape and $f : Ps \rightarrow X$ is a function which assigns elements of X to all positions for that shape. E.g. List is represented by the container $(n : \mathbb{N} \triangleright \mathrm{Fin}\,n)$ where $\mathrm{Fin}\,n = \{0, 1, \ldots, n-1\}$, i.e. a list is given by the length (its shape) and a function which assigns elements to all the positions in the list. List^∞ is represented by $(n : \mathbb{N}^\infty \triangleright \mathrm{Fin}'\,n)$ where $\mathbb{N}^\infty \equiv \nu X.1+X$ are the conatural numbers which extend the usual natural numbers by an infinite element $\infty = 1+\infty$ and Fin' extends Fin by $\mathrm{Fin}\,\infty = \mathbb{N}$, that is the positions of an infinite list are the natural numbers. We show (corollary 5.1) that all strictly positive types can be represented as containers — we leave it as an exercise for the reader to work out the representation of rose trees.

Morphisms between functorial datatypes are polymorphic functions, in categorical terms natural transformations. We define morphisms between containers which represent polymorphic functions: given two containers $(S \triangleright P)$ and $(T \triangleright Q)$ a morphism is given by a pair (u, g) where

- $u : S \rightarrow Q$ is a function on shapes,
- $g : \prod s\!:\!S.\, Q(us) \rightarrow Ps$ is a function which assigns to every position in the target a position in the source.

Each container morphism gives rise to a natural transformation, for example the reverse function $\mathrm{rev}_A : \mathrm{List}A \rightarrow \mathrm{List}A$ is represented by $(\mathrm{id}\!:\!\mathbb{N} \rightarrow \mathbb{N}, r : \prod n\!:\!\mathbb{N}.\,(\mathrm{Fin}\,n \rightarrow \mathrm{Fin}\,n))$ where $rni = n-1-i$. The function on positions has to be defined contravariantly because we can always assign where an element of the target structure comes from but not vice versa. E.g. consider the tail function on lists which is represented by $(\lambda n.n \dot{-} 1,\ \lambda i.i+1)$ where $\dot{-}$ is cutoff subtraction. This can be visualised as tail of list



We show that every polymorphic function can be represented as a container morphism (theorem 3.5), i.e. the representation functor $[\![-]\!]$ is full and faithful.

One of the main applications of containers is **generic programming**: our representation gives a convenient way to program with or reason about datatypes and polymorphic functions. We have already exploited this fact in our work on derivatives of datatypes which uses containers to develop an important idiom in functional programming to support generic editing operations on datatypes (Abbott et al., 2003b, 2004b).

We use here the language of extensional Type Theory with W-types ($\mathbf{MLW^{ext}}$, see Aczel (1999)) as the internal language of locally cartesian closed categories with disjoint coproducts and initial algebras of the functors of the form $FX = \sum s : S.\,(Ps \to X)$, i.e. unary container functors. We show that W-types are sufficient to represent **all** strictly positive types allowing arbitrary nestings of μ and ν, corollary 5.1. Thus, we improve on our previous results in two ways:

- In Abbott et al. (2003a) we only considered μ and we required that the ambient category has infinite limits and colimits, which rules out many interesting examples (e.g. realisability categories).
- In Abbott et al. (2004a) we show that nested μ-types can be represented using W-types, but did not consider ν-types.

The extension to ν-types is non-trivial: it requires to show proposition 4.6 which is stronger than the corresponding proposition 6.1 in Abbott et al. (2004a) — we show that we have an initial solution and not just an isomorphism — and it requires the reduction of M-types (the dual of W-types) to M-types — we achieve this in proposition 4.5.

1.1 Related work

The term **container** is commonly used in programming to refer to a type (or its instances) which can be used to store data. Hoogendijk and de Moor (2000) develop a theory of container types using a relational categorical setting. We share many underlying intuitions and motivations but our framework is based on functions and inspired by intuitionistic Type Theory and it is not clear to use whether there is a more formal relation between the two approaches.

Our work is clearly related to the work of Joyal (1986) on species and analytical functors whose relevance for Computer Science has been recently noticed by Hasegawa (2002). Indeed, if we ignore the fact that analytical functors allow quotients of positions, i.e. if we consider normal functors, we get a concept which is equivalent to a container with a countable set of shapes and a finite set of positions. Hence containers can be considered as a generalisation of normal functors of arbitrary size.

Dyckhoff and Tholen (1987) and Johnstone (1991) introduce the notion of *partial product* which is roughly equivalent to our notion of a container functor, this is originally attributed to Pasynkov (1965) who worked on algebraic topology. However, they use this concept in a different setting and do not introduce morphisms of containers or show the closure properties we establish in this paper.

Dybjer (1997) has shown that non-nested inductive types can be encoded with W-Types. His construction is a sub-case of 5.1 only covering initial algebras of strictly positive functors without nested occurrences of μ or ν. Apart from extending this to nested uses of μ and ν our work also provides a detailed analysis of the categorical infrastructure needed to derive the result.

Recently Gambino and Hyland (2004) have put our results in a more general context and indeed their theorem 12 generalises our proposition 4.6 to dependently typed containers, which they call dependent polynomial functors. Similarly, their theorem 14 is closely related to our proposition 4.3. We also learnt from their work that this construction is related to the proof in Moerdijk and Palmgren (2000) that W-types localise to slice categories.

After learning about our proposition 4.8 that M-types are derivable from W-types, van den Berg and De Marchi (2004) have given an independent proof of this fact using a different methodology.

1.2 Plan of the paper

We review the type theoretic and corresponding categorical infrastructure in section 2. Then in section 3 we formally introduce the category of container and prove some basic properties such as the representation theorem and closure under polynomial operations. The core of the paper is section 4 where we show that container types are closed under μ and ν and that M-types are derivable from W-types. The fairly technical proofs of the theorems in this section are relegated to the appendix. We close with conclusion and discuss further work.

2 Background

2.1 The Categorical Semantics of Dependent Types

This paper can be read in two ways:

(1) as a construction within the extensional Type Theory $\mathbf{MLW^{ext}}$ (see Aczel, 1999) with finite types, W-types but no universes;
(2) as a construction in the internal language of locally cartesian closed categories with disjoint coproducts initial algebras of the functors of the form $FX = \sum s : S.(Ps \to X)$, i.e. W-types. We call those categories **Martin-Löf categories**

The key idea to this dual view is to regard an object $B \in \mathbb{C}/A$ as a *family* of objects of $\mathbb{C}$ indexed by elements of A, and to regard A as the *context* in which B regarded as a *type dependent on A* is defined. The details of this construction can be found in Streicher (1991), Hofmann (1994), Jacobs (1999) and Abbott (2003).

Elements of A will be represented by morphisms $f : U \to A$ in $\mathbb{C}$, and *substitution* of f for A in B is implemented by pulling back B along f to $f^*B \in \mathbb{C}/U$. We start to build the internal language by writing $a : A \vdash Ba$ to express B as a type *dependent* on values in A, and then the result of substitution of f is written as $u : U \vdash B(fu)$. We will treat Ba as an alias for B and $B(fu)$ as an alias for f^*B, and we'll write $a : A \vdash Ba$ or even just $A \vdash B$ for $B \in \mathbb{C}/A$—variables will occasionally be omitted from the internal language where practical for conciseness.

Note that substitution by pullback extends to a functor $f^* : \mathbb{C}/A \to \mathbb{C}/U$: for conciseness of presentation we will assume that substitution corresponds precisely to a choice of pullback, but for a more detailed treatment of the issues involved see Bénabou (1985), Hofmann (1994) and Abbott (2003).

Terms of type $a : A \vdash Ba$ correspond to *global sections* of B, which is to say morphisms $t : 1 \to B$ in $\mathbb{C}/A$. In the internal language we write $a : A \vdash ta : Ba$ for such a morphism in $\mathbb{C}$. We will occasionally write t for ta, again omitting a variable when it can be inferred. Given objects $a : A \vdash Ba$ and $a : A \vdash Ca$ we will write $a : A \vdash fa : Ba \to Ca$ for a morphism in $\mathbb{C}/A$, and similarly $a : A \vdash fa : Ba \cong Ca$ for an isomorphism.

The morphism in $\mathbb{C}$ associated with $B \in \mathbb{C}/A$ will be written as $\pi_B : \sum_A B \to A$ (the *display map* for B); the transformation $B \mapsto \sum_A B$ becomes a left adjoint functor $\sum_A \dashv \pi_B^*$, where pulling back along π_B plays the role of *weakening* with respect to a variable $b : Ba$ in context $a : A$. In the type theory we'll write $\sum_A B \in \mathbb{C}$ as $\sum a : A.Ba$, or more concisely $\sum_A B$, with elements $\Gamma \vdash (t,u) : \sum a : A.Ba$ corresponding to elements $\Gamma \vdash t : A$ and $\Gamma \vdash u : Bt$.

More generally, all of the constructions described here localise: given an arbitrary context $\Gamma \in \mathbb{C}$ and an object $A \in \mathbb{C}/\Gamma$ we can use the isomorphism $(\mathbb{C}/\Gamma)/A \cong \mathbb{C}/\sum_\Gamma A$ to interpret $\Gamma, a\!:\!A \vdash Ba$ both as a morphism $\pi_B : \sum_A B \to A$ in $\mathbb{C}/\Gamma$ and as $\pi_B : \sum_A B \to \sum_\Gamma A$ in $\mathbb{C}$, and $\sum$ extends to provide a left adjoint to every substitution functor. We will write $\Gamma, a\!:\!A, b\!:\!Ba \vdash C(a,b)$ or just $\Gamma, A, B \vdash C$ as a shorthand for $\Gamma, (a,b) : \sum_A B \vdash C(a,b)$. For non-dependent Σ-types we write $A \times B$.

Local cartesian closed structure on $\mathbb{C}$ allows right adjoints to weakening $\pi_A^* \dashv \prod_A$ to be constructed for every $\Gamma \vdash A$ with type expression $\Gamma \vdash \prod a\!:\!A.Bb$ for $\Gamma \vdash \prod_A B$ derived from $\Gamma, A \vdash B$. For non-dependent Π-types we use the notations $A \to B$ and also B^A.

Finally the *equality type* $a,b\!:\!A \vdash a = b$ is represented as an object of $\mathbb{C}/A \times A$ by the diagonal morphism $\delta_A : A \to A \times A$, and more generally $\Gamma, a, b\!:\!A \vdash a = b$. Note that we work in extensional Type Theory, hence there is no difference between definitional and propositional equality. We believe that our development could also be implemented in an intensional system like COQ (Huet et al., 2004) by using setoids (Hofmann, 1997).

For coproducts in the internal language to behave properly, in particular for containers to be closed under products, we require that $\mathbb{C}$ have *disjoint* coproducts: the pullback of distinct coprojections $A \xrightarrow{\kappa} A + B \xleftarrow{\kappa'} B$ into a coproduct is always the initial object 0. When this holds the functor $\mathbb{C}/A+B \to (\mathbb{C}/A) \times (\mathbb{C}/B)$ taking $A + B \vdash C$ to $(A \vdash \kappa^* C,\ B \vdash \kappa'^* C)$ is an equivalence: write $- \,\text{⨣}\, -$ for the inverse functor. Thus given $A \vdash B$ and $C \vdash D$ (with display maps π_B and π_D) we write $A + C \vdash B \,\text{⨣}\, D$ for their disjoint sum; this satisfies two identities: $\sum_{A+C}(B \,\text{⨣}\, D) \cong \sum_A B + \sum_C D$ and $\pi_{B \text{⨣} D} = \pi_B + \pi_D$ (modulo the preceding isomorphism).

For the development of finite coproducts it is actually sufficient to introduce only 0 and disjoint $\mathrm{Bool} = 1 + 1$. In the Type Theory disjointness corresponds to having a constant disjoint: $(\mathrm{true} = \mathrm{false}) \to 0$. Given this we can encode arbitrary coproducts as $A + B = \sum b : \mathrm{Bool}.\,(b = \mathrm{true} \to A) \times (b = \mathrm{false} \to B)$.

Given a (finite) index set I define $[\mathbb{C}^I, \mathbb{C}^J]$ to be the category of *fibred* functors and natural transformations $\mathbb{C}^I \to \mathbb{C}$ where the fibre of $\mathbb{C}^I$ over $\Gamma \in \mathbb{C}$ is the I-fold product $(\mathbb{C}/\Gamma)^I$. Of course, when $J = 1$ we will write this as $[\mathbb{C}^I, \mathbb{C}]$. We write $\prod_{i \in I} A_i$ and $\sum_{i \in I} A_i$ for finite products/coproducts indexed by a finite set I. The same extends to the disjoint finite sum of families where we write $\sum_{i \in I} A_i \vdash \text{⨣}_{i \in I} B_i$.

The following lemma collects together some useful identities which hold in any category considered in this paper.

Lemma 2.1 *For extensive locally cartesian closed $\mathbb{C}$ the following isomorphisms* *hold (*IC *stands for* intensional choice*,* Cu *for* Curry *and* DF *for* disjoint fibres*):*

$$\prod a:A.\ \sum b:Ba.\ C(a,b) \cong \sum f:\prod a:A.Ba.\ \prod a:A.\ C(a,fa) \tag{IC1}$$
$$\prod_{i\in I}\sum b:B_i.\ C_i b \cong \sum a:\prod_{i\in I}B_i.\ \prod_{i\in I}C_i(\pi_i a) \tag{IC2}$$
$$\prod a:A.\ C^{Ba} \cong \left(\sum a:A.\ Ba\right) \to C \tag{Cu1}$$
$$\prod_{i\in I}(B_i \to C) \cong \left(\sum_{i\in I}B_i\right) \to C \tag{Cu2}$$
$$\left(\coprod_{i\in I}B_i\right)(\kappa_i a) \cong B_i a \tag{DF1}$$
$$\sum_{i\in I}\sum a:A_i.\ C(\kappa_i a) \cong \sum a:\sum_{i\in I}A_i.\ Ca \tag{DF2}$$
□

2.2 W-types and M-types

In Martin-Löf's Type Theory (Martin-Löf, 1984; Nordström et al., 1990) the building block for inductive constructions is the W-type. Given a family of constructors $A \vdash B$ the type $\mathsf{W}a:A.Ba$ (or $\mathsf{W}_A B$) should be regarded as the type of "well founded trees" constructed by regarding each $a:A$ as a constructor of arity Ba.

The standard presentation of a W-type is through one type forming rule, an introduction rule and an elimination rule, together with an equation.

Definition 2.2 *A type system* has W-types *iff it has a type constructor*

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash \mathsf{W}_A B} \tag{W-type}$$

together with a constructor term

$$\Gamma,\ a:A,\ f:Ba \to \mathsf{W}_A B \vdash \mathrm{sup}(a,f):\mathsf{W}_A B \tag{sup}$$

and an elimination rule

$$\frac{\begin{array}{l}\Gamma,\ w:\mathsf{W}_A B \vdash Cw \\ \Gamma,\ a:A,\ f:Ba \to \mathsf{W}_A B,\ g:\prod b:B(a).C(fb) \vdash h(a,f,g):C(\mathrm{sup}(a,f))\end{array}}{\Gamma,\ w:\mathsf{W}_A B \vdash \mathrm{wrec}_h w:Cw} \tag{wrec}$$

satisfying the equation for variables $a:A$ and $f:(\mathsf{W}_A B)^{B(a)}$:

$$\mathrm{wrec}_h \mathrm{sup}(a,f)) = h(a,f,\mathrm{wrec}_h \cdot f) \ .$$

where $(\mathrm{wrec}_h \cdot f)b \equiv \mathrm{wrec}_h(fb)$; note that the first argument of this composition is a dependent function.

Note that the elimination rule together with equality types ensures that wrec_h is unique. It is easy to see that the rule (wrec) implies that each $\mathsf{W}_A B$ is an initial algebra for the functor $FX = \sum a\!:\!A.\,(Ba \to X)$ in each slice. Moerdijk and Palmgren (2000) show that the global version of W-types implies the existence of W-types in each slice.

We consider that this notion summarises the essence of Martin-Löf's Type Theory from a categorical perspective, hence the following definition.

Definition 2.3 *A* Martin-Löf category *is a locally cartesian closed category with disjoint coproducts and an initial algebra for every container functor (i.e. W-types).*

We know that W-types exist in toposes with natural numbers objects (Moerdijk and Palmgren, 2000, proposition 3.6) and in categories which are both locally cartesian closed and locally presentable (Abbott et al., 2003a, theorem 6.8). Moreover, W-types exist in models of Type Theory based on realisability such as the category of ω-sets.

Dually, we introduce M-types as the terminal coalgebras as the terminal coalgebras of the same collection of functors. There is no standard representation of M-types in Type Theory, indeed the elegant unification of primitive recursion and induction does not dualise easily. However, in extensional Type Theory we can simply state the equations implied by assuming that M-types are terminal coalgebras:

Definition 2.4 *A type system* has M-types *iff it has a type constructor*

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash \mathsf{M}_A B} \qquad \text{(M-type)}$$

together with a deconstructor term

$$\Gamma,\ m\!:\!\mathsf{M}_A B \vdash \mathrm{sup}^{-1} m : \sum a\!:\!A.\ (Ba \to \mathsf{M}_A B) \qquad \text{(supinv)}$$

and an unfolding rule

$$\frac{\begin{array}{c}\Gamma,\ c\!:\!C \vdash sc\!:\!A \\ \Gamma,\ c\!:\!C \vdash pc\!:\!B(sc) \to C\end{array}}{\Gamma,\ c\!:\!C \vdash \mathrm{unfold}_{s,p} c : \mathsf{M}_A B} \qquad \text{(mrec)}$$

satisfying the equation

$$\mathrm{sup}^{-1}(\mathrm{unfold}_{s,p} c) = (sc,\ \mathrm{unfold}_{s,p} \cdot pc) \ .$$

and the conditional equation

$$\frac{\begin{array}{l}\Gamma,\ c\!:\!C \vdash hc : M \\ \mathrm{sup}^{-1}(hc) = (sc,\ h \cdot pc)\end{array}}{hc = \mathrm{unfold}_{s,p} c} \qquad \text{(mrec-eta)}$$

Observe that we can define the inverse

$$\sup = \text{unfold}_{\pi,\pi'} : \left(\sum a\!:\!A.\ (Ba \to \mathsf{M}_A B)\right) \longrightarrow \mathsf{M}_A B$$

2.3 Strictly positive types

We have already discussed strictly positive types in the introduction, here is an inductive definition:

Definition 2.5 *A* strictly positive type (SPT) in n variables *(Abel and Altenkirch, 2000) is a type expression (with type variables $X_1, \ldots, X_n$) built up inductively according to the following rules:*

- *if K is a constant type (with no type variables) then K is a SPT;*
- *each type variable X_i is a SPT;*
- *if F, G are SPTs then so are $F+G$ and $F \times G$;*
- *if K is a constant type and F a SPT then $K \to F$ is a SPT;*
- *if F is a SPT in $n+1$ variables then $\mu X.\, F, \nu X.\, F$ is a SPT in n variables (for X any type variable).*

3 Basic Properties of Containers

We will now introduce the category of containers $\mathscr{G}$ equipped with its interpretation functor $[\![-]\!] : \mathscr{G} \to [\mathbb{C}, \mathbb{C}]$. When constructing fixed points it is also necessary to take account of containers with parameters, so we define $[\![-]\!] : \mathscr{G}_I \to [\mathbb{C}^I, \mathbb{C}]$ for each parameter index set I. For the purposes of this paper the index set n or I can be assumed to be a finite set, but in fact this makes little difference. Indeed, it is straightforward to generalise the development in this paper to the case where containers are parameterised by *internal* index objects $I \in \mathbb{C}$; when $\mathbb{C}$ has enough coproducts nothing is lost by doing this, since $\mathbb{C}^I \simeq \mathbb{C}/\sum_{i \in I} 1$. This generalisation will be important for future developments of this theory, but is not required in this paper.

This discussion leads to the following definition of a container in I parameters.

Definition 3.1 *Given an index set I define the* category of containers in I parameters *$\mathscr{G}_I$ as follows:*

- *Objects are pairs $(A \in \mathbb{C}, B \in (\mathbb{C}/A)^I)$; write this as $(A \triangleright B) \in \mathscr{G}_I$*
- *Morphisms $(A \triangleright B) \to (C \triangleright D)$ are pairs (u, f) for $u : A \to C$ in $\mathbb{C}$ and $f : (u^*)^I D \to B$ in $(\mathbb{C}/A)^I$.*

Thus a container in one parameter is just a family $A \vdash B$, while a container in n parameters consists of a single shape A together with a family of positions $A \vdash B_i$ for each $i \in n$.

A container $(A \rhd B) \in \mathcal{G}_I$ can be written using type theoretic notation as

$$\vdash A \qquad i\!:\!I, a\!:\!A \vdash B_i(a) \ .$$

A morphism $(u, f)\!:\!(A \rhd B) \to (C \rhd D)$ can be written in type theoretic notation as

$$u\!:\!A \longrightarrow C \qquad i\!:\!I, a\!:\!A \vdash f_i(a)\!:\!D_i(ua) \longrightarrow B_i(a) \ .$$

Finally, each $(A \rhd B) \in \mathcal{G}_I$, thought of as a syntactic presentation of a datatype, generates a fibred functor $[\![A \rhd B]\!]\!:\!\mathbb{C}^I \to \mathbb{C}$ which is its semantics.

Definition 3.2 *Define the* container extension functor $[\![-]\!]\!:\!\mathcal{G}_I \to [\mathbb{C}^I, \mathbb{C}]$ *as follows. Given $(A \rhd B) \in \mathcal{G}_I$ and $X \in \mathbb{C}^I$ define*

$$[\![A \ \rhd \ B]\!]X \equiv \sum a\!:\!A.\ \prod_{i \in I}(B_i(a) \to X_i) = \sum_A \prod_I X^B \ ,$$

and for $(u, f)\!:\!(A \rhd B) \to (C \rhd D)$ define $[\![u, f]\!]\!:\![\![A \rhd B]\!] \to [\![C \rhd D]\!]$ to be the natural transformation with components $[\![u, f]\!]_X\!:\![\![A \rhd B]\!]X \to [\![C \rhd D]\!]X$ defined thus:

$$(a, g)\!:\![\![A \ \rhd \ B]\!]X \vdash [\![u, f]\!]_X\,(a, g) \equiv (ua, (g_i \cdot f_i)_{i \in I}) \ .$$

An alternative inductive definition of containers can be shown to be equivalent to the system above: a container functor $F(\vec{X}, Y)$ in $n+1$ parameters can be thought of as a family of containers in one parameter indexed by the parameter $\vec{X}$. To be precise, we have the following proposition.

Proposition 3.3 *The following induction clauses define the containers with finite sets of parameters.*

(1) A container in 0 parameters is the same thing as an object of $\mathbb{C}$, which is its extension.
(2) There is a bijection between container functors in $n+1$ parameters and pairs (S, P) where S is a container functor $S\!:\!\mathbb{C}^n \to \mathbb{C}$ in n parameters and P is a family $S\vec{X} \vdash P\vec{X}$ for each $\vec{X} \in \mathbb{C}^n$ satisfying coherent isomorphisms $(P\vec{Y})((S\vec{f})s) \cong (P\vec{X})s$ for $\vec{f}\!:\!\vec{X} \to \vec{Y}$ and $s\!:\!S\vec{X}$. Define the extension of the pair (S, P) to be the functor $F_{S,P}(\vec{X}, Y) \equiv \sum s\!:\!S\vec{X}.\,((P\vec{X})s \to Y)$.

PROOF. Case 1 is obvious by inspection, so consider case 2.

Write a container in $n+1$ parameters as $(A \triangleright B, C)$ where B is an n-indexed family with extension $[\![A \triangleright B, C]\!](\vec{X}, Y) = \sum a\!:\!A.\, (\prod_{i \in I}(B_i a \to X_i)) \times (Ca \to Y)$. Given $(A \triangleright B, C)$ define $S \equiv [\![A \triangleright B]\!]$ and define $P\vec{X} \equiv C(S!_{\vec{X}})$ where $!_{\vec{X}} : \vec{X} \to 1$ in $\mathbb{C}^n$. Observe that $S!_{\vec{X}}$ is the projection $\pi : \sum_A \prod_I X^B \to A$, and so $(P\vec{X})(a, f) = Ca$ depends only on $a\!:\!A$. This gives us the construction of a pair (S, P) from an $n+1$ parameter container. Conversely, given a pair (S, P) we can reconstruct the container $(A \triangleright B, C)$ by taking $C \equiv P1$; indeed P is fully determined by $P1$.

To see that these two construction have the same extension:

$$
\begin{aligned}
[\![A \triangleright B, C]\!](\vec{X}, Y) &= \sum a\!:\!A.\, \left(\prod_{i \in I}(B_i a \to X_i)\right) \times (Ca \to Y) \\
&\cong \sum a\!:\!A.\, \sum f : \prod_{i \in I}(B_i a \to X_i).\, (Ca \to Y) \\
&\cong \sum s : S\vec{X}.\, (Ps \to Y) = F_{S,P}(\vec{X}, Y)
\end{aligned}
\qquad \square
$$

The following proposition follows from the construction of $[\![-]\!]$ as a type expression.

Proposition 3.4 *For each container $F \in \mathcal{G}_I$ and each container morphism $\alpha : F \to G$ the functor $[\![F]\!]$ and natural transformation $[\![\alpha]\!]$ are fibred over $\mathbb{C}$.* $\square$

By making essential use of the fact that the natural transformations in $[\mathbb{C}^I, \mathbb{C}]$ are fibred we can show that T is full and faithful.

Theorem 3.5 (Representation) *The functor $T : \mathcal{G}_I \to [\mathbb{C}^I, \mathbb{C}]$ is full and faithful.*

PROOF. To show that T is full and faithful it is sufficient to lift each natural transformation $\alpha : [\![A \triangleright B]\!] \to [\![C \triangleright D]\!]$ in $[\mathbb{C}^I, \mathbb{C}]$ to a map $(u_\alpha, f_\alpha) : (A \triangleright B) \to (C \triangleright D)$ in $\mathcal{G}_I$ and show this construction is inverse to T.

Given $\alpha : [\![A \triangleright B]\!] \to [\![C \triangleright D]\!]$ define $\ell \equiv (a, \mathrm{id}_{B(a)}) : [\![A \triangleright B]\!]B$ in the context $a\!:\!A$ — that is to say, construct $\ell : 1 \to [\![A \triangleright B]\!]B$ in the fibre $\mathbb{C}/A$. As the natural transformation α is fibred, it localises to $\alpha_B : [\![A \triangleright B]\!]B \to [\![C \triangleright D]\!]B$ in $\mathbb{C}/A$ and so we can compute $A \vdash \alpha_B \ell : [\![C \triangleright D]\!]B = \sum_C \prod_I B^D$; write this as $\alpha_B \ell = (u_\alpha, f_\alpha)$, where $u_\alpha a : C$ and $f_\alpha a : \prod_{i \in I}(D_i(u_\alpha(a)) \Rightarrow B_i(a))$ in context $a\!:\!A$.

Thus (u_α, f_α) can be understood as a morphism $(A \triangleright B) \to (C \triangleright D)$ in $\mathcal{G}_I$, so we have a construction $[\mathbb{C}^I, \mathbb{C}]([\![A \triangleright B]\!], [\![C \triangleright D]\!]) \to \mathcal{G}_I((A \triangleright B), (C \triangleright D))$; it remains to show that this is inverse to the action of the functor T.

For $\alpha = [\![u, f]\!]$, evaluate $\alpha_B \ell = (ua, \mathrm{id} \cdot f) = (u, f)$. In the opposite direction, to show that $\alpha = [\![u_\alpha, f_\alpha]\!]$, let $X \in \mathbb{C}^I$, $a\!:\!A$ and $g : \prod_{i \in I}(B_i(a) \Rightarrow X_i)$ be given, consider the diagram

$$\begin{tikzcd}
1 \arrow[r, "\ell"] \arrow[dr, "(u_\alpha a, f_\alpha a)"'] & \llbracket A \rhd B \rrbracket B \arrow[r, "\llbracket A \rhd B \rrbracket g"] \arrow[d, "\alpha_B"] & \llbracket A \rhd B \rrbracket X \arrow[d, "\alpha_X"] \\
& \llbracket C \rhd D \rrbracket B \arrow[r, "\llbracket C \rhd D \rrbracket g"'] & \llbracket C \rhd D \rrbracket X
\end{tikzcd} \qquad \text{in } \mathbb{C}/\llbracket A \rhd B \rrbracket X$$

and evaluate

$$\begin{aligned}
\alpha_X(a,g) &= \alpha_X((\llbracket A \rhd B \rrbracket g)\ell) = (\llbracket C \rhd D \rrbracket g)(\alpha_B \ell) = (\llbracket C \rhd D \rrbracket g)(u_\alpha a, f_\alpha a) \\
&= (u_\alpha a, g \cdot f_\alpha a) = \llbracket u_\alpha, f_\alpha \rrbracket_X (a,g) \quad .
\end{aligned}$$

This shows that $\alpha = \llbracket u_\alpha, f_\alpha \rrbracket$ as required. $\square$

This theorem gives a particularly simple analysis of polymorphic functions between container functors. For example, it is easy to observe that there are precisely n^m polymorphic functions $X^n \to X^m$: the data type X^n is the container $(1 \rhd n)$ and hence there is a bijection between polymorphic functions $X^n \to X^m$ and functions $m \to n$. Similarly, any polymorphic function $\mathrm{List}X \to \mathrm{List}X$ can be uniquely written as a function $u : \mathbb{N} \to \mathbb{N}$ together with for each natural number $n : \mathbb{N}$, a function $f_n : un \to n$.

It turns out that each $\mathscr{G}_I$ inherits products and coproducts from $\mathbb{C}$, and that T preserves them:

Proposition 3.6 *If $\mathbb{C}$ has products and coproducts then $\mathscr{G}_I$ has products and coproducts preserved by T.*

PROOF. Since T is full and faithful it we can reflect the construction products and coproducts along T.

Products. Let $(A_k \rhd B_k)_{k \in K}$ be a family of objects in $\mathscr{G}_I$ and compute

$$\begin{aligned}
\textstyle\prod_{k \in K} \llbracket A_k \rhd B_k \rrbracket X &= \textstyle\prod_{k \in K} \sum a : A_k .\ \prod_{i \in I} (B_{k,i}(a) \Rightarrow X_i) \\
&\cong \textstyle\sum a : \prod_{k \in K} A_k .\ \prod_{k \in K} \prod_{i \in I} (B_{k,i}(\pi_k a) \Rightarrow X_i) \\
&\cong \textstyle\sum a : \prod_{k \in K} A_k .\ \prod_{i \in I} \Big(\Big(\sum_{k \in K} B_{k,i}(\pi_k a) \Big) \Rightarrow X_i \Big) \\
&= \Big\llbracket \textstyle\prod_{k \in K} A_k \rhd \sum_{k \in K} (\pi_k^*)^I B_k \Big\rrbracket X
\end{aligned}$$

showing by reflection along T that

$$\textstyle\prod_{k \in K} (A_k \rhd B_k) \cong \Big(\prod_{k \in K} A_k \rhd \sum_{k \in K} (\pi_k^*)^I B_k \Big) \quad .$$

Coproducts. Given a family $(A_k \vdash B_k)_{k\in K}$ of objects in $\mathscr{G}_I$ calculate (making essential use of disjointness of fibres):

$$\begin{aligned}
\textstyle\sum_{k\in K} [\![A_k \rhd B_k]\!] X &= \textstyle\sum_{k\in K} \sum a : A_k.\ \prod_{i\in I}(B_{k,i}(a) \Rightarrow X_i) \\
&\cong \textstyle\sum_{k\in K} \sum a : A_k.\ \prod_{i\in I}\left(\left(\biguplus_{k'\in K} B_{k',i}\right)(\kappa_k a) \Rightarrow X_i\right) \\
&\cong \textstyle\sum a : \sum_{k\in K} A_k.\ \prod_{i\in I}\left(\left(\biguplus_{k\in K} B_{k,i}\right)(a) \Rightarrow X_i\right) \\
&= \left[\!\!\left[\textstyle\sum_{k\in K} A_k \rhd (\biguplus_{k\in K} B_{j,i})_{i\in I}\right]\!\!\right] X
\end{aligned}$$

showing by reflection along T that

$$\textstyle\sum_{k\in K}(A_k \rhd B_k) \cong \left(\sum_{k\in K} A_k \rhd \biguplus_{k\in K} B_k\right) \quad . \qquad \square$$

Container functors are also closed under exponentiation by a constant type.

Proposition 3.7 *If $F : \mathbb{C}^I \to \mathbb{C}$ is a container functor then so is the exponential functor F^K defined by $F^K X \equiv K \to FX$.*

PROOF. Let $F = [\![A \rhd B]\!]$ and calculate

$$\begin{aligned}
K \to [\![A \rhd B]\!] X &= K \to \left(\textstyle\sum a : A.\ \prod_{i\in I}(B_i a \to X_i)\right) \\
&\cong \textstyle\sum f : K \to A.\ \prod k : K.\ \prod_{i\in I}(B_i(fk) \to X_i) \\
&\cong \textstyle\sum f : K \to A.\ \prod_{i\in I}\left(\left(\sum k : K.\ B_i(fk)\right) \to X_i\right) \\
&= \left[\!\!\left[f : K \to A \rhd \left(\textstyle\sum k : K.\ B_i(fk)\right)_{i\in I}\right]\!\!\right] X
\end{aligned}$$

showing that we can define $K \to (A \rhd B) \equiv \left(f : K \to A \rhd (\sum k : K. B_i(fk))_{i\in I}\right)$. $\square$

Given containers $F \in \mathscr{G}_{I+1}$ and $G \in \mathscr{G}_I$ we can compose their extensions to construct the functor

$$[\![F]\!]\,[[\![G]\!]] \equiv (\mathbb{C}^I \xrightarrow{(\mathrm{id}_{\mathbb{C}^I}, [\![G]\!])} \mathbb{C}^I \times \mathbb{C} \cong \mathbb{C}^{I+1} \xrightarrow{[\![F]\!]} \mathbb{C}) \quad .$$

Writing this equation as $[\![F]\!]\,[[\![G]\!]]\vec{X} = [\![F]\!]\,(\vec{X}, [\![G]\!]\,\vec{X})$ we can see that this defines a form of substitution.

This substitution lifts to a functor $-[-] : \mathscr{G}_{I+1} \times \mathscr{G}_I \to \mathscr{G}_I$ as follows. For a container in $\mathscr{G}_{I+1}$ write $(A \rhd B, E) \in \mathscr{G}_{I+1}$, where $B \in (\mathbb{C}/A)^I$ and $E \in \mathbb{C}/A$ and define:

$$(A \rhd B, E)[(C \rhd D)] \equiv \left(a : A, f : Ea \to C \rhd \left(B_i a + \textstyle\sum e : Ea.\ D_i(fe)\right)_{i\in I}\right) \quad .$$

In other words, given type constructors $F(\vec{X},Y)$ and $G(\vec{X})$ this construction defines the composite type constructor $F[G](\vec{X}) \equiv F(\vec{X},G(\vec{X}))$.

Proposition 3.8 *Substitution of containers commutes with substitution of functors thus:* $[\![F]\!]\,[[\![G]\!]] \cong [\![F[G]]\!]$.

PROOF. Calculate (for conciseness we write exponentials using superscripts where convenient and elide the variable $a\!:\!A$ throughout):

$$\begin{aligned}
&\left[\!\!\left[A \rhd \vec{B},E\right]\!\!\right]\,[[\![C \rhd D]\!]]X \\
&\qquad = \textstyle\sum_A \left(\left(\prod_{i\in I} X_i^{B_i}\right) \times \left(E \to \sum c\!:\!C.\ \prod_{i\in I} X_i^{D_i c}\right)\right) \\
&\qquad \cong \textstyle\sum_A \left(\left(\prod_{i\in I} X_i^{B_i}\right) \times \left(\sum f\!:\!C^E.\ \prod e\!:\!E.\ \prod_{i\in I} X_i^{D_i(fe)}\right)\right) \\
&\qquad \cong \textstyle\sum_A \sum f\!:\!C^E.\ \prod_{i\in I}\left(X_i^{B_i} \times \left(\prod e\!:\!E.\, X_i^{D_i(fe)}\right)\right) \\
&\qquad \cong \textstyle\sum_A \sum f\!:\!C^E.\ \prod_{i\in I}\left(\left(B_i + \sum e\!:\!E.\, D_i(fe)\right) \to X_i\right) \\
&\qquad \cong \left[\!\!\left[(A \rhd \vec{B},E)[C \rhd D]\right]\!\!\right] X \ \ .
\end{aligned}$$

As all the above isomorphisms are natural in X we get the desired isomorphism of functors. $\square$

This shows how composition of containers captures the composition of container functors. More generally, it is worth observing that a composition of containers of the form $-\circ- : \mathscr{G}_I \times \mathscr{G}_J^I \to \mathscr{G}_J$ reflecting composition of functors $\mathbb{C}^J \to \mathbb{C}^I \to \mathbb{C}$ can also be defined making containers into a bicategory with 0-cells the index sets I and the category of homs from I to J given by the container category $\mathscr{G}_I^J$ (Abbott, 2003, proposition 4.4.4).

Finally we should look at the treatment of variables and the weakening of containers as type expressions. First note that every type variable X_i can be regarded as a container.

Proposition 3.9 *Every projection function* $\pi_i : \mathbb{C}^I \to \mathbb{C}$ *defined by* $\pi_i\vec{X} \equiv X_i$ *for each* $i \in I$ *is a container functor.*

PROOF. $[\![1 \rhd (i=j)_{j\in I}]\!]\,\vec{X} \cong \prod_{j\in I}((i=j) \to X_j) \cong X_i$. $\square$

Given a type expression $F(X_1,\ldots,X_n)$ in n variables and a variable renaming function $f\!:\!n \to m$ we can construct a type expression $F(X_{f1},\ldots,X_{fn})$ in m variables. This construction extends to containers in an obvious way.

Proposition 3.10 *Each function $f : I \to J$ lifts to a functor $\uparrow^f : \mathscr{G}_I \to \mathscr{G}_J$ with $\left[\!\left[\uparrow^f F \right]\!\right] X \cong [\![F]\!] (X \circ f)$, where we regard X as a functor $J \to \mathbb{C}$.*

PROOF. Define $\uparrow^f (A \triangleright \vec{B}) \equiv (A \triangleright (\sum_{i \in I}(fi = j) \times B_i)_{j \in J})$ and calculate

$$\begin{aligned}\left[\!\left[\uparrow^f (A \triangleright \vec{B}) \right]\!\right] X &= \sum a : A.\ \prod\nolimits_{j \in J} \left(\left(\sum\nolimits_{i \in I} (fi = j) \times B_i a \right) \to X_j \right) \\ &\cong \sum a : A.\ \prod\nolimits_{j \in J} \prod\nolimits_{i \in I} (((fi = j) \times B_i a) \to X_j) \\ &\cong \sum a : A.\ \prod\nolimits_{i \in I} (B_i a \to X_{fi}) = \left[\!\left[A \triangleright \vec{B} \right]\!\right] (X \circ f)\ . \qquad \square\end{aligned}$$

For example, in the special case of weakening a container $(A \triangleright B)$ in n variables by adding one variable in the final position we obtain $\uparrow (A \triangleright B) = (A \triangleright B')$ where $B'_i \equiv B_i$ for $i \leq n$ and $B'_{n+1} \equiv 0$. More generally we can weaken along any inclusion $f : I \rightarrowtail J$ of variables transforming $(A \triangleright B)$ into $(A \triangleright B') \equiv \uparrow^f (A \triangleright B)$ where $B'_{fi} = B_i$ and $B'_j = 0$ otherwise. We will normally leave such weakenings implicit.

4 Inductive and Coinductive Containers

We have already observed in section 2 that the existence of W-types is equivalent to assuming that every container functor in one parameter has an initial algebra. Indeed, this assignment extends to a functor $\mu : \mathscr{G}_1 \to \mathscr{G}_0$, and similarly M-types gives us a functor ν.

Proposition 4.1 *The construction of W-types and M-types extends to functors $\mu : \mathscr{G}_1 \to \mathscr{G}_0$ and $\nu : \mathscr{G}_1 \to \mathscr{G}_0$ respectively.*

PROOF. Since $\mathscr{G}_0 \cong \mathbb{C}$ the action of μ on objects is given by taking W-types. For each morphism of containers $\alpha : F \to G$ define $\mu\alpha : \mu F \to \mu G$ by induction over the algebra $\sup^G \cdot [\![\alpha]\!]_{\mu G} : [\![F]\!] (\mu G) \to \mu G$, so $\mu\alpha$ uniquely satisfies the equation $\mu\alpha \cdot \sup^F = \sup^G \cdot [\![\alpha]\!]_{\mu G} \cdot [\![F]\!] (\mu\alpha)$. It is a routine verification that this construction is functorial; the construction for ν is dual. $\square$

In this section we will see how to extend this construction to the following more general result.

Theorem 4.2 *There exist functors $\mu : \mathscr{G}_{I+1} \to \mathscr{G}_I$ and $\nu : \mathscr{G}_{I+1} \to \mathscr{G}_I$ satisfying isomorphisms $[\![\mu F]\!] \cong \mu [\![F]\!]$ and $[\![\nu F]\!] \cong \nu [\![F]\!]$.*

PROOF. For each F a container functor in $n+1$ parameters and n fixed parameters $\vec{X}$ we know that $F(\vec{X},-)$ is a container functor in one parameter, so the initial algebra $\mu Y.\,F(\vec{X},Y)$ exists as a W-type for each choice of $\vec{X}$, and similarly $\nu Y.\,F(\vec{X},Y)$ exists via M-types.

Thus we have functors $\mu\,[\![-]\!] : \mathscr{G}_{I+1} \to [\mathbb{C}^I,\mathbb{C}]$ and $\nu\,[\![-]\!] : \mathscr{G}_{I+1} \to [\mathbb{C}^I,\mathbb{C}]$. It remains to show that if $F(\vec{X},Y)$ is a container functor over parameters $\vec{X}$ and Y then $\mu Y.\,F(\vec{X},Y)$ and $\nu Y.\,F(\vec{X},Y)$ are container functors over $\vec{X}$. We show this for μ in proposition 4.3 below and for ν in proposition 4.5. The existence of functors $\mu,\nu : \mathscr{G}_{I+1} \rightrightarrows \mathscr{G}_I$ then follows by full and faithfulness of $[\![-]\!]$. $\square$

Now let $F = (S \rhd P, Q) \in \mathscr{G}_{I+1}$ be a container in multiple parameters with extension

$$\begin{aligned}
[\![F]\!]\,(\vec{X},Y) &\equiv [\![S \rhd P,Q]\!]\,(\vec{X},Y)\\
&= \textstyle\sum s:S.\,\left(\prod_{i\in I}(P_i s \to X_i)\right) \times (Qs \to Y) \ .
\end{aligned}$$

To show that $\mu Y.\,[\![F]\!]\,(\vec{X},Y)$ and $\nu Y.\,[\![F]\!]\,(\vec{X},Y)$ are container functors with respect to $\vec{X}$ we need to compute I-indexed containers $(A_\mu \rhd B_\mu)$ and $(A_\nu \rhd B_\nu)$ such that $[\![A_\mu \rhd B_\mu]\!]\,\vec{X} \cong \mu Y.\,[\![F]\!]\,(\vec{X},Y)$ and $[\![A_\nu \rhd B_\nu]\!]\,\vec{X} \cong \nu Y.\,[\![F]\!]\,(\vec{X},Y)$. Clearly we can calculate

$$\begin{aligned}
A_\mu &\cong [\![A_\mu \rhd B_\mu]\!]\,1 \cong \mu Y.\,[\![F]\!]\,(1,Y) \cong \mu Y.\,[\![S \rhd Q]\!]\,Y \cong \mathsf{W}_S Q\\
A_\nu &\cong [\![A_\nu \rhd B_\nu]\!]\,1 \cong \nu Y.\,[\![F]\!]\,(1,Y) \cong \nu Y.\,[\![S \rhd Q]\!]\,Y \cong \mathsf{M}_S Q \ ,
\end{aligned}$$

but the construction of $\mathsf{W}_S Q \vdash B_\mu$ and $\mathsf{M}_S Q \vdash B_\nu$ will involve the inductive construction of families; we will show how to construct $A \vdash B$ using W-types in proposition 4.6 below.

In the rest of this section we will simplify the presentation by ignoring the index set I and writing $P \to X$ for $\prod_{i\in I}(P \to X_i)$. In particular, this means that the family $B \in (\mathbb{C}/A)^I$ will be treated uniformly (as if $I = 1$). It is a straightforward exercise to generalise the development to arbitrary index sets. We will therefore take

$$[\![F]\!]\,(X,Y) \equiv \textstyle\sum s:S.\,(Ps \to X) \times (Qs \to Y) \ .$$

For any container $G \equiv (A \rhd B)$ we can calculate the substitution

$$F[G] = (S \rhd P,Q)[(A \rhd B)] = \left(s:S, f:Qs \to A \ \rhd\ Ps + \textstyle\sum q:Q.\,B(fq)\right) \ .$$

This can be written more concisely as $\left(S, A^Q \rhd P + \sum_Q \varepsilon^* B\right)$, where $\varepsilon : A^Q \times Q \to A$ is the evaluation map. Observe now that any fixed point $\psi : [\![S \rhd Q]\!]A \cong A$ induces an isomorphism of positions between $F[G]$ and G, or equivalently an isomorphism $\psi : [\![F[G]]\!]\,1 \cong [\![G]\!]\,1$ and it's clear that any fixed point $F[G] \cong G$ which agrees with ψ must be of the form $(\psi, \varphi^{-1}) : F[G] \to G$ for some family of isomorphisms

$$s : S, f : Qs \to A \vdash \varphi_{s,f} : Ps + \sum q : Q.\, B(fq) \cong B(\psi(s,f)) \ .$$

It turns out that for the initial algebra $\mu Y.\, F(X,Y)$ the existence of a family $\mathsf{W}_S Q \vdash B_\mu$ satisfying the fixed point above is sufficient.

Proposition 4.3 *Given the notation above, if there exists a family* $\mathsf{W}_S Q \vdash B$ *equipped with a family of isomorphisms*

$$s : S, f : Qs \to \mathsf{W}_S Q \vdash \varphi : Ps + \sum q : Q.\, B(fq) \cong B(\sup(s,f))$$

then $[\![\mathsf{W}_S Q \rhd B]\!]\, X \cong \mu Y.\, [\![F]\!]\,(X,Y)$.

PROOF. See appendix A.1. □

It may seem surprising that *any* isomorphism $\varphi : P + \sum_Q \varepsilon^* B \cong \sup^* B$ is sufficient, as in general we would expect this isomorphism problem to have multiple solutions. Indeed, we can discover from the proposition above that in this case B is defined uniquely up to isomorphism, since $\mu Y.\, [\![F]\!]\,(X,Y)$ is itself unique.

This can be explained intuitively by observing that B corresponds to the type of paths into a finite tree, and consequently there cannot be any infinite paths. This occurs because the structure of the functor $X \mapsto P + \sum_Q \varepsilon^* X$ respects the structure of the initial algebra sup, thereby forcing B to be unique. An example of this occurs in Wraith's theorem (Johnstone, 1977, theorem 6.19) which treats the special case $A = \mathbb{N}$.

The corresponding result for final coalgebras requires a little more structure on B_ν, as the isomorphism φ does *not* fully determine B_ν over $\mathsf{M}_S Q$: in effect there can be infinite paths into an infinite tree. Instead, we will require $\mathsf{M}_S Q \vdash B_\nu$ to be an *initial family* over *the fixed point* $\sup : [\![S \rhd Q]\!]\, \mathsf{M}_S Q \cong \mathsf{M}_S Q$, as we will want to select only finite paths into the infinite tree generated by $\mathsf{M}_S Q$.

The following definition and proposition 4.6 are specialised to the application of this paper. For more general notions of initial family see Gambino and Hyland (2004).

Definition 4.4 *An* initial family *over* a fixed point $\psi : [\![S \rhd Q]\!]\, A \cong A$ *is defined to be an initial algebra for the functor* $\mathbb{C}/A \to \mathbb{C}/A$ *taking* X *to* $\psi^{-1*}(P + \sum_Q \varepsilon^* X)$.

In other words, a family $A \vdash B$ is initial over ψ if it is equipped with a morphism $\varphi : P + \sum_Q \varepsilon^* B \to \psi^* B$ and given any other family $A \vdash X$ also equipped with a morphism $\alpha : P + \sum_Q \varepsilon^* X \to \psi^* X$ there exists a morphism $\overline{\alpha} : B \to X$ uniquely satisfying the equation $\psi^*\overline{\alpha}\cdot\phi = \alpha\cdot\left(P+\sum_Q \varepsilon^*\overline{\alpha}\right)$. Using the internal language we can write the two components of this equation thus:

$$
\begin{array}{l}
s:S,\, f:A^{Qs},\, p:Ps \vdash \overline{\alpha}_{\psi(s,f)}(\varphi_{s,f}(\kappa p)) = \alpha(\kappa p) \\
s:S,\, f:A^{Qs},\, q:Qs,\, b:B(fq) \vdash \overline{\alpha}_{\psi(s,f)}(\varphi_{s,f}(\kappa'(q,b))) = \alpha(\kappa'(q,\overline{\alpha}_{fq}b))\ \ .
\end{array}
$$

If we can find an initial family over the final coalgebra $\sup^{-1} : \mathsf{M}_S Q \cong [\![S \rhd Q]\!]\,\mathsf{M}_S Q$ then this is sufficient to show that $\nu Y.\,[\![F]\!]\,(X,Y)$ is a container functor.

Proposition 4.5 *Given a family* $\mathsf{M}_S Q \vdash B$ *equipped with an initial family*

$$s:S,\, f:Qs \to \mathsf{W}_S Q \vdash \varphi : Ps + \sum q:Q.\, B(fq) \longrightarrow B(\sup(s,f))$$

over sup *then* $[\![A \rhd B]\!]\,X \cong \nu Y.\,[\![F]\!]\,(X,Y)$.

PROOF. See appendix A.2. □

So the construction of both μ and ν types as containers depends on the construction of initial families over fixed points. Concurrently with this work Gambino and Hyland (2004) has shown that the existence of such initial families follows from the existence of W-types. Here we provide a detailed construction as a regular subtype of a W-type.

Proposition 4.6 *For every fixed point* $\psi : [\![S \rhd Q]\!]\,A \cong A$ *there exists an initial family* $A \vdash \mathrm{Pos}_{P,\psi}$ *over* ψ *for the functor* $X \mapsto P + \sum_Q \varepsilon^* X$.

PROOF. See appendix A.3. □

Combining propositions 4.3 and 4.5 with 4.6 and 4.8 below we obtain theorem 4.2, summarised by the following corollary.

Corollary 4.7 *If* $\mathbb{C}$ *has W-types then containers are closed under the construction of* μ*-types and* ν*-types, and provide a complete semantics of strictly positive types.* □

4.1 Constructing M-types

Up to now we have separately assumed the existence of W-types and M-types in the ambient category $\mathbb{C}$. In this section we will show that in fact coinductive types can be constructed using inductive types.

If we assume $\mathbb{C}$ to have enough infinite limits, in particular to be closed under the formation of ω-limits, then it is easy to see that M-types exist: writing $T \equiv [\![S \triangleright P]\!]$ construct the ω-limit

$$1 \longleftarrow T1 \longleftarrow \cdots \longleftarrow T^n 1 \longleftarrow \cdots \longleftarrow \varprojlim_{n\in\mathbb{N}} T^n 1 \quad ,$$

then as T preserves ω-limits (indeed T preserves all connected limits as so does the functor $\sum_S$) it is a well known result (eg, Poigné, 1992) that $\nu T \equiv \varprojlim_{n\in\mathbb{N}} T^n 1$ is a final coalgebra. This approach was taken in Abbott et al. (2003a); Abbott (2003).

In the present treatment we cannot assume the existence of ω-limits: for example, in the effective topos the natural numbers object is *not* the external limit of $\mathbb{N}$ copies of 1. One possible approach is to construct the family $n : \mathbb{N} \vdash T^n 1$ as a family in $\mathbb{C}$ together with an internal representation of the restriction morphisms $T^{n+k}1 \to T^n 1$ and take its *internal* limit, which certainly does exist. We do not do this in this paper, as the necessary machinery is not developed here.

However, we can use this (internal) limit construction to understand the construction in the present paper. Each projection $\pi_n : \nu T \to T^n 1$ takes a potentially infinite tree and truncates it to depth n; such truncated trees can be expressed as subobjects of the W-type $\widehat{M} \equiv \mu X.\, 1 + TX$. Writing $\perp$ and sup for the two components of the constructor $1 + T\widehat{M} \to \widehat{M}$, we can define an inclusion $i_n : T^n 1 \rightarrowtail \widehat{M}$ inductively with $i_0 \equiv \perp$ and $i_{n+1}(s, f) \equiv \sup(s, i_n \cdot f)$.

This means that the family of composites $i_n \cdot \pi_n$ can be understood as a morphism $\mathbb{N} \times \nu T \to \widehat{M}$, or equivalently, a morphism $\nu T \to \widehat{M}^{\mathbb{N}}$: this last morphism turns out to be a regular monomorphism. Each infinite tree in νT is represented as an evolving family of finite truncated trees, and it's clear that $f : \mathbb{N} \to \widehat{M}$ is in νT only if fn is a truncation of $f(n+1)$. Correctly captured, this turns out to be the defining equation for νT as a regular subobject of $\widehat{M}^{\mathbb{N}}$.

Thus we get the following theorem.

Proposition 4.8 *Every Martin-Löf category is closed under the formation of M-types, that is, every unary container functor has a final coalgebra.*

PROOF. See appendix B.1. □

5 Conclusions and Further Work

We can summarise the main results of the paper in the following corollary:

Corollary 5.1 *Each strictly positive type F in n variables can be interpreted as an n-ary container $(\!|F|\!) : \mathscr{G}_n$. Assuming as given the interpretation of n-ary SPTs $(\!|F|\!) = (S \rhd \vec{P})$, $(\!|G|\!) = (T \rhd \vec{Q})$ and $n+1$-ary SPT $(\!|H|\!) = (U \rhd \vec{R}, R')$, we have the following translation:*

$$
\begin{aligned}
(\!|K|\!) &= (K \rhd j \mapsto 0) \\
(\!|X_i|\!) &= (1 \rhd j \mapsto (i = j)) \\
(\!|F + G|\!) &= (S + T \rhd j \mapsto P_j \,\hat{+}\, Q_j) \\
(\!|F \times G|\!) &= ((s,t) S \times T : \rhd j \mapsto P_j\, s + Q_j\, t) \\
(\!|K \to F|\!) &\mapsto \big(f : K \to S \rhd j \mapsto \textstyle\sum k : K.\, P_j\,(f\,k)\big) \\
(\!|\mu X.H|\!) &= (\mathsf{W}_U R' \rhd j \mapsto \mathrm{Pos}_{R_j,\mathrm{sup}}) \\
(\!|\nu X.H|\!) &= (\mathsf{M}_U R' \rhd j \mapsto \mathrm{Pos}_{R_j,\mathrm{sup}})
\end{aligned}
$$

In the special case $n = 1$ this implies that all closed strictly positive types can be interpreted as objects in any Martin-Löf category.

PROOF. By combining propositions 3.6, 3.7, 3.9, 3.10 and corollary 4.7.

The reader will notice that our definition of strictly positive types is restricted to a simple type discipline even though we work in a dependently typed setting. A natural extension of the work presented here would allow the definitions of strictly positive families which can be interpreted as initial algebras of endofunctors on a given slice category. We are currently working on this, it seems that W-types, i.e. Martin-Löf categories are still sufficient to interpret strictly positive families. This has important consequences for the implementation of systems like Epigram (McBride, 2004) which use schematic inductive definitions. The correctness of the schemes is not checked and is a likely cause of unsoundness. Using our construction we can translate the schematic definitions into a fixed core theory whose terms can be easily checked.

Nested datatypes (Altenkirch and Reus, 1999; Bird and Paterson, 1999) provide another challenge: to treat them we would need to represent higher order functors. However, it is likely that Martin-Löf categories are still sufficient as a framework.

Another interesting line is to allow quotients of positions to be able to treat types like Bags. Indeed this is already present in Joyal's definition of analytic functors and can be easily adapted to the category of containers, we have presented first results in Abott et al. (2004). There is an interesting interaction with our work on derivatives, e.g. using quotients we should be able to prove a version of Taylor's theorem in a type-theoretic setting. This construction will take place within a predicative topos with W-types which extends Martin-Löf categories by effective quotients.

Acknowledgements

The content of this paper has greatly benefitted from numerous discussions with our colleagues, in particular: Conor McBride, Benno van den Berg, Peter Dybjer, Nicola Gambino, Peter Hancock, Martin Hofmann, Martin Hyland and Federico De Marchi.

References

M. Abbott. *Categories of Containers*. PhD thesis, University of Leicester, 2003.

M. Abbott, T. Altenkirch, and N. Ghani. Categories of containers. In *Proceedings of Foundations of Software Science and Computation Structures*, volume 2620 of *Lecture Notes in Computer Science*, 2003a.

M. Abbott, T. Altenkirch, and N. Ghani. Representing nested inductive types using w-types. In *International Colloquium on Automata, Languages and Programming, ICALP*, 2004a. to appear.

M. Abbott, T. Altenkirch, N. Ghani, and C. McBride. Derivatives of containers. In *6th International Conference on Typed Lambda Calculi and Applications*, volume 2701 of *Lecture Notes in Computer Science*, 2003b.

M. Abbott, T. Altenkirch, N. Ghani, and C. McBride. ∂ for data. submitted for publication, February 2004b.

A. Abel and T. Altenkirch. A predicative strong normalisation proof for a λ-calculus with interleaving inductive types. In *Types for Proof and Programs, TYPES '99*, volume 1956 of *Lecture Notes in Computer Science*, 2000.

M. Abott, T. Altenkirch, N. Ghani, and C. McBride. Constructing polymorphic programs with quotient types. 7th International Conference on Mathematics of Program Construction (MPC 2004) *to appear*, February 2004.

P. Aczel. On relating type theories and set theories. *Lecture Notes in Computer Science*, 1657:1–18, 1999. URL `citeseer.ist.psu.edu/aczel98relating.html`.

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

J. Bénabou. Fibred categories and the foundation of naive category theory. *Journal of Symbolic Logic*, 50(1):10–37, 1985.

R. Bird and R. Paterson. Generalised folds for nested datatypes. *Formal Aspects of Computing*, 11(3), 1999.

P. Dybjer. Representing inductively defined sets by wellorderings in Martin-Löf's type theory. *Theoretical Computer Science*, 176:329–335, 1997.

R. Dyckhoff and W. Tholen. Exponentiable morphisms, partial products and pullback complements. *J. Pure Appl. Algebra*, 49(1-2):103–116, 1987.

N. Gambino and M. Hyland. Wellfounded trees and dependent polynomial functors. In S. Berardi, M. Coppo, and F. Damiani, editors, *Types for Proofs and Programs (TYPES 2003)*, Lecture Notes in Computer Science, 2004.

R. Hasegawa. Two applications of analytic functors. *Theoretical Computer Science*, 272(1-2):112–175, 2002.

M. Hofmann. On the interpretation of type theory in locally cartesian closed categories. In *CSL*, pages 427–441, 1994.

M. Hofmann. *Extensional Constructs in Intensional Type Theory*. Springer, 1997.

P. Hoogendijk and O. de Moor. Container types categorically. *Journal of Functional Programming*, 10(2):191–225, 2000.

G. Huet, G. Kahn, and C. Paulin-Mohring. *The* Coq *Proof Assistant - A tutorial - Version 8.0*, Apr. 2004. URL `http://coq.inria.fr`.

B. Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. Elsevier, 1999.

P. T. Johnstone. *Topos Theory*. Academic Press, 1977.

P. T. Johnstone. Partial products, bagdomains and hyperlocal toposes. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors, *Applications of Categories in Computer Science*, London Mathematical Society Lecture Note Series, pages 315–339. Cambridge University Press, 1991.

A. Joyal. Foncteurs analytiques et espèces de structures. In *Combinatoire Énumérative*, number 1234 in Lecture Notes in Mathematics, pages 126–159. Springer, 1986.

P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, Napoli, 1984.

C. McBride. Epigram, 2004. `http://www.dur.ac.uk/CARG/epigram`.

I. Moerdijk and E. Palmgren. Wellfounded trees in categories. *Annals of Pure and Applied Logic*, 104:189–218, 2000.

B. Nordström, K. Petersson, and J. M. Smith. *Programming in Martin-Löf's Type Theory*. Number 7 in International Series of Monographs on Computer Science. Oxford University Press, 1990.

B. A. Pasynkov. Partial topological products. *Transactions of the Moscow Mathematical Society*, 13:153–272, 1965.

A. Poigné. Basic category theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1 of *Handbook of Logic in Computer Science*. Oxford University Press, 1992.

T. Streicher. *Semantics of Type Theory*. Progress in Theoretical Computer Science. Birkhäuser Verlag, 1991.

D. Turner. Elementary strong functional programming. In R. Plasmeijer and P. Hartel, editors, *First International Symposium on Functional Programming Languages in Education*, number 1022 in Lecture Notes in Computer Science, pages 1–13. Springer, 1996.

B. van den Berg and F. De Marchi. Non-well-founded trees in categories. submitted for publication, 2004.

A Proofs for Induction and Coinduction

Throughout this appendix F denotes the container $F \equiv (S \triangleright P, Q)$.

Proposition A.1 *If there exists a family* $\mathsf{W}_S Q \vdash B$ *equipped with an isomorphism*

$$s : S,\, f : Qs \to \mathsf{W}_S Q \;\vdash\; \varphi_{s,f} : Ps + \sum q : Qs.\, B(fq) \cong B(\sup(s, f))$$

then $[\![\mathsf{W}_S Q \triangleright B]\!]\, X \cong \mu Y.\, [\![F]\!]\,(X, Y)$.

PROOF. For conciseness write $A \equiv \mathsf{W}_S Q$ and $G \equiv (A \triangleright B)$ through this proof. First observe that $[\![F]\!]\,(X, [\![G]\!]\, X) = [\![F]\!]\,[[\![G]\!]] X \cong [\![F[G]]\!]\, X$ and calculate

$$F[G] = (S \triangleright P, Q)[(A \triangleright B)] = \big(s : S,\, f : Qs \to A \triangleright Ps + \sum q : Q.\, B(fq)\big) \quad ;$$

it is clear that $\alpha \equiv (\sup, \varphi^{-1}) : F[G] \to G$ is an $F[-]$ algebra. To show that each $[\![\alpha]\!]_X$ generates an initial $[\![F]\!]\,(X, -)$-algebra let an algebra $\beta : [\![F]\!]\,(X, Y) \to Y$ be given: we need to construct $\overline{\beta} : [\![G]\!]\, X \to Y$ uniquely making

$$\begin{tikzcd}
{[\![F]\!]\,(X, [\![G]\!]\, X) \cong [\![F[G]]\!]\, X} \arrow[r, "{[\![\alpha]\!]_X}"] \arrow[d, "{[\![F]\!]\,(X, \overline{\beta})}"'] & {[\![G]\!]\, X} \arrow[d, "\overline{\beta}"] \\
{[\![F]\!]\,(X, Y)} \arrow[r, "\beta"'] & Y
\end{tikzcd} \tag{A.1}$$

commute. The corresponding equation can be written as

$$s : S,\, f : Ps \to X,\, g : Qs \to [\![G]\!]\, X \;\vdash\; \overline{\beta}(\sup(s, g_1), k) = \beta(s, f, \overline{\beta} \cdot g) \tag{A.2}$$

for

$$g_1 \equiv \pi \cdot g \qquad g_2(q, b) \equiv (\pi'(gq))b \qquad k \equiv [f ; g_2] \cdot \varphi^{-1} \quad . \tag{A.3}$$

The decomposition of g into $g_1 : Qs \to A$ and $g_2 : (\sum q : Qs. B(g_1 q)) \to X$ is at the heart of the isomorphism $[\![F]\!]\,[[\![G]\!]] \cong [\![F[G]]\!]$, and of course the bijection between $k : B(\sup(s, g_1)) \to X$ and f, g_2 is mediated by the isomorphism φ. Note also that g can be reconstructed as $gq = (g_1 q,\, g_2(q, -))$ where $g_2(q, -) \equiv \lambda b.\, g_2(q, b)$.

We can now construct $\overline{\beta} : (\sum_A X^B) \to Y$ by W-induction by constructing

$$a : \mathsf{W}_S Q \;\vdash\; \overline{\beta}(a, -) : (Ba \to X) \to Y$$

and using the W-induction rule wrec. To apply this rule we need to define the induction step h taking induction data and returning a value of the above type.

The following type expression turns out to be appropriate:

$$s:S,\,g_1:Qs\to A,\,r:\prod q:Qs.\,((B(g_1q)\to X)\to Y),\,k:B(\sup(s,g_1))\to X\ \vdash$$
$$\qquad (h(s,g_1,r))k \equiv \beta(s,\,f,\,\lambda q.\ (rq)(g_2(q,-)))\ \ ,$$

where $[f;g_2]=k\cdot\varphi$ as in (A.3) above. If we now define $\overline{\beta}(a,-)\equiv \mathrm{wrec}_h a$ then the induction equation becomes, using the variables of (A.2) and the identities of (A.3):

$$\begin{aligned}\overline{\beta}(\sup(s,g_1),k) &= (h(s,g_1,\lambda q.\ \overline{\beta}(g_1q,-)))k\\ &= \beta(s,\,f,\,\lambda q.\ \overline{\beta}(g_1q,\,g_2(q,-)))\\ &= \beta(s,\,f,\,\lambda q.\ \overline{\beta}(gq)) = \beta(s,\,f,\,\overline{\beta}\cdot g)\end{aligned}$$

which is precisely the equation (A.2), showing that $\overline{\beta}$ is the required initial morphism and that indeed $[\![G]\!]X$ is an initial algebra. □

Now let $\sup^{-1}:\mathsf{M}_SQ\to[\![S\rhd Q]\!]\,\mathsf{M}_SQ$ be a final coalgebra for $[\![S\rhd Q]\!]$ with inverse sup.

Proposition A.2 *Given a family* $\mathsf{M}_SQ\vdash B$ *equipped with an initial family*

$$s:S,\,f:Qs\to\mathsf{W}_SQ\ \vdash\ \varphi_{s,f}:Ps+\sum q:Qs.\,B(fq)\longrightarrow B(\sup(s,f))$$

over sup *then* $[\![\mathsf{M}_SQ\rhd B]\!]X\cong\nu Y.\,[\![F]\!]\,(X,Y)$.

PROOF. First note that the construction of $in^{-1}:T_{A\rhd B}X\to F(X,T_{A\rhd B}X)$ in the proof above works unchanged and so in^{-1} is a coalgebra. We now want to show that this is a final coalgebra, so let $\beta:Y\to F(X,Y)$ be a coalgebra; it will be sufficient to construct $\overline{\beta}:Y\to T_{A\rhd B}X$ uniquely satisfying $in\cdot F(X,\overline{\beta})\cdot\beta=\overline{\beta}$.

Start by writing $\beta=(s,g,h):Y\to\sum_S(X^P\times Y^Q)$, which we can write as

$$Y\xrightarrow{\ s\ }S\qquad Y\ \vdash\ s^*P\xrightarrow{\ g\ }X\qquad Y\ \vdash\ s^*Q\xrightarrow{\ h\ }Y\quad .$$

The goal then is to construct $\overline{\beta}=(a,k):Y\to\sum_A X^B$ thus:

$$Y\xrightarrow{\ a\ }A\qquad Y\ \vdash\ a^*B\xrightarrow{\ k\ }X\quad ,$$

and the equation $in \cdot F(X, \overline{\beta}) \cdot \beta = \overline{\beta}$ translates into the commutative square

$$\begin{tikzcd}
Y \arrow[r, "{(s,g,h)}"] \arrow[d, dashed, "{(a,k)}"'] & \sum_S (X^P \times Y^Q) \arrow[d, dashed, "{\sum_S (X^P \times (a,k)^Q)}"] \\
\sum_A X^B \arrow[r, "in^{-1}"'] & \sum_S \left(X^P \times (\sum_A X^B)^Q\right)
\end{tikzcd} \quad .$$

Observe that $\pi \cdot in^{-1} \cdot (a,k) = s$, in other words we can write

$$a = \psi \cdot (s,f) \quad \text{for some} \quad Y \vdash s^* Q \xrightarrow{f} A \quad .$$

Evaluating both edges of this square leads to the equation

$$(s,\ g,\ (a \cdot h,\ h^* k)) = (s,\ k \cdot \phi \cdot \kappa,\ (f,\ k \cdot \phi \cdot \kappa'))$$

which can be interpreted as the following three equations in the type theory (where the detailed dependency of each function symbol is made explicit):

$$\begin{array}{llr}
y : Y,\ p : P(sy) \vdash\ g_y p = k_y \varphi_{sy, f_y} \kappa p & & g = k \cdot \phi \cdot \kappa \\
y : Y,\ q : Q(sy) \vdash\ a h_y q = f_y q & & a \cdot h = f \\
y : Y,\ q : Q(sy),\ b : B(f_y q) \vdash\ k_{h_y q} b = k_y \varphi \kappa'(q,b) & & h^* k = k \cdot \varphi \cdot \kappa' \quad .
\end{array}$$

Now the equation $a = \psi \cdot (s,f) = \psi \cdot (s, a \cdot h)$ fully determines $a : Y \to A$ by finality of A, so the problem remains to determine k. The equations for k can be captured by the following commutative triangle:

$$Y \vdash \quad \begin{tikzcd}
s^* P + \sum_{s^* Q} f^* B \arrow[rr, "{(s,f)^* \varphi}"] \arrow[dr, "{[g; h^* k]}"'] & & a^* B \arrow[dl, "k"] \\
& X &
\end{tikzcd} \quad .$$

Note that $s^* P + \sum_{s^* Q} f^* B = (s,f)^* (P + \sum_Q \varepsilon^* B)$ and similarly $a^* B = (s,f)^* \psi^* B$, so we can transpose the right hand edge of this triangle to produce the top and right edges of the square below

$$S, A^Q \vdash \quad \begin{tikzcd}
P + \sum_Q \varepsilon^* B \arrow[r, "\varphi"] \arrow[d, "\mathrm{id}_P + \sum_Q \varepsilon^* \overline{k}"'] & \psi^* B \arrow[d, "\psi^* \overline{k}"] \\
P + \sum_Q \varepsilon^* \prod_a X \arrow[r, "\alpha"'] & \psi^* \prod_a X
\end{tikzcd} \quad .$$

Here $A \vdash \overline{k}: B \to \prod_a X$ is the transpose of k. As ψ is an isomorphism we can write $\prod_{(s,f)} X \cong \psi^* \prod_a X$, and so in particular $\psi^* \overline{k} \cdot \varphi$ is the transpose of $k \cdot (s,f)^* \varphi$. If we can construct $\alpha = [\alpha_0; \alpha_1]$ such that $\alpha \cdot (\mathrm{id}_P + \sum_Q \varepsilon^* \overline{k})$ is the transpose of $[g; h^* k]$ then we can appeal to initiality of φ to conclude that $\overline{k}$ (and hence k) is uniquely determined and so $T_{A \triangleright B}$ is a terminal coalgebra.

Taking α_0 to be the transpose of $g : s^* P = (s,f)^* P \to X$ it remains to construct α_1.

Start by observing that $A \vdash \prod_a X$ can be written as

$$a' : A \vdash \prod_a X \equiv \left(\sum y : Y.\, \mathrm{Eq}(ay, a')\right) \Rightarrow X = X^{\sum y:Y.\mathrm{Eq}(ay,a')} \quad ,$$

or more suggestively, as $\prod_a X \equiv \prod y : Y \upharpoonright \mathrm{Eq}(ay, a').X$. Now for $s' : S$ and $f' : A^{Q(s')}$, we can write $\psi^* \prod_a X = \prod y : Y \upharpoonright \mathrm{Eq}(ay, \psi(s', f')).X$, and similarly for $q : Q(s')$ we have $\varepsilon^* \prod_a X = \prod y : Y \upharpoonright \mathrm{Eq}(ay, f'q).X$. The map $A \vdash \overline{k} : B \to \prod_a X$ can be described by the equation

$$a' : A,\ b : B(a),\ y : Y \upharpoonright \mathrm{Eq}(ay, a') \vdash (\overline{k}_{a'} b) y \equiv k_y b \quad .$$

Now define $S, A^Q, Q \vdash \alpha_1 : \varepsilon^* \prod_a X \to \psi^* \prod_a X$ by the equation in context

$$\begin{aligned} & s' : S,\ f' : A^{Q(s')},\ q : Q(s'),\ \theta : \textstyle\prod y : Y \upharpoonright \mathrm{Eq}(ay, f'q).X,\ y : Y \upharpoonright \mathrm{Eq}(ay, \psi(s', f')) \vdash \\ & \qquad (\alpha_1 \theta) y \equiv \theta(h_y q) \quad . \end{aligned}$$

This is well defined so long as $ay = \psi(s', f') \implies ah_y q = f'q$, but this follows from the equation $ay = \psi(sy, a \cdot h_y)$, which in particular implies that $f' = a \cdot h_y$.

It remains to verify that $\alpha_1 \cdot \sum_Q \varepsilon^* \overline{k}$ is the transpose of $h^* k$; this follows from the calculation

$$\begin{aligned} & s' : S,\ f' : A^{Q(s')},\ q : Q(s'),\ b : B(f'q),\ y : Y \upharpoonright \mathrm{Eq}(ay, \psi(s', f')) \vdash \\ & \qquad (\alpha_1 \overline{k}_{f'q} b) y = (\overline{k}_{f'q} b)(h_y q) = k_{h_y q} b \quad . \end{aligned}$$

□

Proposition A.3 *For every fixed point $\psi : [\![S \triangleright Q]\!] A \cong A$ the functor $\mathbb{C}/A \to \mathbb{C}/A$ taking X to $\psi^{-1*}(P + \sum_Q \varepsilon^* X)$ has an initial algebra.*

PROOF. Write $S, A^Q \vdash \varphi : P + \sum_Q \varepsilon^* B \to \psi^* B$ for the isomorphism that we wish to construct. As already noted, we cannot directly appeal to W-types to construct this fixed point, so the first step is to create a fixed point equation that we *can* solve. Begin by "erasing" the type dependency of B and construct (writing $\sum_Q Y \cong Q \times Y$, etc)

$$\begin{aligned} \widehat{B} &\equiv \mu Y.\, \textstyle\sum_S \sum_{A^Q} (P + Q \times Y) \cong \mu Y.\, \left(\sum_S (A^Q \times P) + \left(\sum_S (A^Q \times Q)\right) \times Y\right) \\ &\cong \mathrm{List}\left(\textstyle\sum_S (A^Q \times Q)\right) \times \sum_S (A^Q \times P) \quad ; \end{aligned}$$

there is no problem in constructing arbitrary lists in $\mathbb{C}$ so $\widehat{B}$ clearly exists.

The task now is to select the "well-formed" elements of $\widehat{B}$. A list in $\widehat{B}$ can be thought of as a putative path through a tree in $\mu Y.\ [\![S \triangleright P,Q]\!]\,(X,Y)$; we want $B(a)$ to be the set of all valid paths to X-substitutable locations in the tree.

An element of $\widehat{B}$ can be conveniently written as a list followed by a tuple thus

$$([(s_0,f_0,q_0),\ldots,(s_{n-1},f_{n-1},q_{n-1})],(s_n,f_n,p))$$

for $s_i : S$, $f_i : A^{Q(s_i)}$, $q_i : Q(s_i)$ and $p : P(s_n)$. The condition that this is a well formed element of $B(\psi(s_0,f_0))$ can be expressed as the n equations

$$f_i(q_i) = \psi(s_{i+1},f_{i+1}) \quad \text{for } i < n$$

which can be captured as an equaliser diagram

$$\begin{tikzcd} \sum_A B \arrow[r, tail, "e"] \arrow[dr, "\pi_B"'] & \widehat{B} \arrow[r, shift left, "\alpha"] \arrow[r, shift right, "\beta"'] \arrow[d, "\varpi"] & \mathrm{List}A \\ & A & \end{tikzcd}$$

where α, β and ϖ are defined inductively on $\widehat{B}$ as follows (and $\pi_B \equiv \varpi \cdot e$):

$$\begin{aligned} \alpha(\mathsf{nil},p') &= \mathsf{nil} & \alpha(\mathsf{cons}((s,f,q),l),p') &= \mathsf{cons}(fq,\alpha(l,p')) \\ \varpi(\mathsf{nil},(s,f,p)) &= \psi(s,f) & \varpi(\mathsf{cons}((s,f,q),l),p') &= \psi(s,f) \\ \beta(\mathsf{nil},p') &= \mathsf{nil} & \beta(\mathsf{cons}(b,l),p') &= \mathsf{cons}(\varpi(l,p'),\beta(l,p')) \ . \end{aligned}$$

The property that $b : \widehat{B}$ is an element of B can be written $b : B(\varpi b)$ and using the equations above we can establish:

$$(\mathsf{nil},(s,f,p)) : B(\psi(s,f)) \tag{A.4}$$

$$fq = \varpi(l,p') \wedge (l,p') : B(fq) \implies (\mathsf{cons}((s,f,q),l),p') : B(\psi(s,f)) \ . \tag{A.5}$$

The converse to (A.5) also holds, since $(\mathsf{cons}((s,f,q),l),p') : B(\psi(s,f)) \iff \mathsf{cons}(fq,\alpha(l,p')) = \mathsf{cons}(\varpi(l,p'),\beta(l,p')) \iff fq = \varpi(l,p') \wedge (l,p') : B(fq)$.

The isomorphism $\widehat{\varphi} : \sum_S \sum_{A^Q}(P + Q \times \widehat{B}) \cong \widehat{B}$ can now be used to construct the isomorphism φ for B. Writing an element of $\sum_S \sum_{A^Q}(P + Q \times \widehat{B})$ as $(s,f,\kappa p)$ or

$(s,f,\kappa'(q,b))$, the function $\widehat{\varphi}$ can be computed thus:

$$
\begin{array}{ccc}
\sum_S \sum_{A^Q}(P+Q\times\widehat{B}) & \overset{\widehat{\varphi}}{\cong} & \begin{array}{c}\mathsf{List}\left(\sum_S(A^Q\times Q)\right)\\ \times\sum_S(A^Q\times P)\end{array} = \widehat{B} \\
(s,f,\kappa p) & \longleftrightarrow & (\mathsf{nil},(s,f,p)) \\
(s,f,\kappa'(q,(l,p'))) & \longleftrightarrow & (\mathsf{cons}((s,f,q),l),p')\ .
\end{array}
$$

To show that $\widehat{\varphi}$ restricts to a morphism $\varphi : P + \sum_Q \varepsilon^* B \to \psi^* B$ we need to show for each $s:S$ and $f:A^Q$ that $x:(P(s)+\sum q:Q(s).\,B(fq))$ implies $\widehat{\varphi}(s,f,x):B(\psi(s,f))$.

When $x=\kappa p$ we immediately have $\widehat{\varphi}(s,f,\kappa p) = (\mathsf{nil},(s,f,p)) : B(\psi(s,f))$ by (A.4) above. Now let $(s,f,\kappa'(q,(l,p')))$ be given with $(l,p'):B(fq)$ (which means, in particular, that $\varpi(l,p')=fq$) and consider the equation $\widehat{\varphi}(s,f,\kappa'(q,(l,p'))) = (\mathsf{cons}((s,f,q),l),p')$, then by (A.5) this is also in $B(\psi(s,f))$. Thus $\widehat{\varphi}$ restricts to

$$
s:S, f:A^{Q(s)} \vdash \varphi_{s,f} : P(s) + \sum q:Q(s).\,B(fq) \longrightarrow B(\psi(s,f))\ .
$$

We have, in effect, constructed φ making the diagram below commute:

$$
\begin{tikzcd}
\sum_S\sum_{A^Q}\left(P+\sum_Q\varepsilon^*B\right) \arrow[rrr, "\varphi"] \arrow[dr, "\pi"] \arrow[dd, hook] & & & \sum_A B \arrow[dl, "\pi_B"'] \arrow[dd, hook, "e"] \\
& \sum_S A^Q \arrow[r, "\psi"] & A & \\
\sum_S\sum_{A^Q}(P+Q\times\widehat{B}) \arrow[rrr, "\widehat{\varphi}"'] \arrow[ur, "\pi"] & & & \widehat{B} \arrow[ul, "\varpi"]
\end{tikzcd}\ .
$$

Finally to show that φ is an *initial* morphism let $A \vdash X$ be given together with $S,A^Q \vdash h:P+\sum_Q\varepsilon^*X \to \psi^*X$. The condition that a map $A \vdash \overline{h}:B\to X$ is an algebra morphism can be written as the pair of equations

$$
s:S,\ f:A^{Q(s)},\ p:P(s) \vdash \overline{h}_{\psi(s,f)}\varphi_{s,f}\kappa p = h_{s,f}\kappa p \tag{A.6}
$$
$$
s:S,\ f:A^{Q(s)},\ q:Q(s),\ b:B(fq) \vdash \overline{h}_{\psi(s,f)}\varphi_{s,f}\kappa'(q,b) = h_{s,f}\kappa'(q,\overline{h}_{fq}b)\ . \tag{A.7}
$$

We will construct $\overline{h}:B\to X$ by induction over $\widehat{B}$, but some preparation is required. Write $A\vdash\widehat{B}_\varpi$ for $\widehat{B}$ regarded as a type over A with display map ϖ, then $\widehat{B}\cong\sum_A\widehat{B}_\varpi$. Similarly observe that B as the equaliser $B\cong\mathrm{Eq}(\alpha,\beta)$ can now be written using the isomorphism $A\vdash B\cong\sum_{\widehat{B}_\varpi}\mathrm{Eq}(\alpha,\beta)$. We can now transpose $\overline{h}$ into a form suitable for induction over $\widehat{B}$ thus:

$$\frac{\dfrac{A \vdash B \cong \sum_{\widehat{B}_{\varpi}} \mathrm{Eq}(\alpha,\beta) \xrightarrow{\overline{h}} X}{\widehat{B} \cong \sum_{A} \widehat{B}_{\varpi} \vdash \mathrm{Eq}(\alpha,\beta) \longrightarrow \varpi^{*}X}}{\widehat{B} \vdash \tilde{h} : (\varpi^{*}X)^{\mathrm{Eq}(\alpha,\beta)}} \ .$$

We can relate $\overline{h}$ and $\tilde{h}$ by the equation (the parameter w can be silently ignored: only its presence is important)

$$a : A,\ b : \widehat{B}_{\varpi}(a),\ w : \mathrm{Eq}(\alpha b, \beta b) \vdash \tilde{h}(b) = \overline{h}_a(b) : X(a) \ . \tag{A.8}$$

We can now construct $\tilde{h}$ and verify the equations it satisfies by constructing two terms h_0 and h_1 as follows. We can use the membership rules (A.4, A.5) to reason about elements of $X^{\mathrm{Eq}(\alpha,\beta)}$, ie $b : B(\varpi b)$ iff $\alpha b = \beta b$ iff $b^{*}\mathrm{Eq}(\alpha,\beta)$ is inhabited. Now write $\mathrm{Wf}(b) \equiv \mathrm{Eq}(\alpha b, \beta b)$ (abbreviating the type that says that "b is a well formed element of B"), and then $\mathrm{Wf}(\mathsf{nil},(s,f,p)) \cong 1$ and so we can define h_0:

$$s : S,\ f : A^{Q(s)},\ p : P(s) \vdash h_0 p \equiv h_{s,f}\kappa p : X(\psi(s,f)) \ .$$

Now consider the construction of h_1 in context:

$$\begin{aligned} & s : S,\ f : A^{Q(s)},\ q : Q(s),\ l : \mathrm{List}\big(\textstyle\sum_S (A^Q \times Q)\big),\ p' : \textstyle\sum_S (A^Q \times P), \\ & x : (l,p')^{*}\big((\varpi^{*}X)^{\mathrm{Eq}(\alpha,\beta)}\big) \vdash h_1 : (\mathsf{cons}((s,f,q),l),p')^{*}\big((\varpi^{*}X)^{\mathrm{Eq}(\alpha,\beta)}\big) \ . \end{aligned}$$

In context s,f,q,l,p' define $b \equiv (\mathsf{cons}((s,f,q),l),p')$; this can now be written as

$$w_1 : \mathrm{Wf}(l,p'),\ x : X(\varpi(l,p')),\ w_2 : \mathrm{Wf}(b) \vdash h_1 : X(\varpi b) \ .$$

Now $\varpi b = \psi(s,f)$ and the existence of $w_2 : \mathrm{Wf}(b)$ implies $\varpi(l,p') = fq$ and hence $x : X(fq)$ and so we can define $h_1 \equiv h_{s,f}\kappa'(q,x)$. Now $\tilde{h} \equiv \mathrm{lrec}_{h_0,h_1}$ can be constructed by induction and finally define $\overline{h}$ to be the transpose of $\tilde{h}$.

It remains to verify that the equations for $\tilde{h}$ transpose using (A.8) to the equations (A.6, A.7) for $\overline{h}$:

$$\begin{aligned} & \tilde{h}(\mathsf{nil},(s,f,p)) = \overline{h}_{\psi(s,f)}(\mathsf{nil},(s,f,p)) = \overline{h}_{\psi(s,f)}\varphi_{s,f}\kappa p \\ & h_0(s,f,p) = h_{s,f}\kappa p \\ & \tilde{h}(\mathsf{cons}((s,f,q),l),p') = \overline{h}_{\psi(s,f)}\varphi_{s,f}\kappa'(q,(l,p')) \\ & h_1((s,f,q),l,p',\tilde{h}(l,p')) = h_{s,f}\kappa'(q,\overline{h}((l,p'))) \ . \end{aligned}$$

$\square$

B Proof of Existence of M-types

Proposition B.1 *In a Martin-Löf category, every container functor in one parameter has a final coalgebra.*

PROOF. Let $S \vdash P$ be the family for which $\mathsf{M}_S P \equiv \nu X.\, [\![S \triangleright P]\!] X$ is to be constructed; for conciseness, write $TX \equiv [\![S \triangleright P]\!] X = \sum_S X^P$ throughout this proof. Define $\widehat{M} \equiv \mu X.\, 1 + TX$, writing $\bot : \widehat{M}$ and $\sup : T\widehat{M} \to \widehat{M}$ for the two components of the initial algebra $1 + T\widehat{M} \to \widehat{M}$. The idea of this proof is to represent an element $m : \mathsf{M}_S P$ by a family of functions $m : \mathbb{N} \to \widehat{M}$ where each $m_n : \widehat{M}$ represents the infinite tree m truncated at depth n: the value $\bot$ represents points where the tree has been cut off.

We can define coalgebra to algebra coinduction over $\widehat{M}^{\mathbb{N}}$. First construct the T-algebra $\alpha : T(\widehat{M}^{\mathbb{N}}) \to \widehat{M}^{\mathbb{N}}$ by cases over $\mathbb{N}$:

$$\alpha_0(s, f) \equiv \bot \qquad\qquad \alpha_{n+1}(s, f) \equiv \sup(s, f_n) \ ,$$

where $f_n \equiv \lambda p : Ps.\, (fp)n$ for $f : Ps \to \widehat{M}^{\mathbb{N}}$. It will be convenient to use this convention for the parameter n throughout this proof. The morphism α will later restrict to the inverse to the final coalgebra for $\mathsf{M}_S P$.

Now let $\beta : X \to TX$ be any given coalgebra; writing the components of βx as $\beta_0 x : S$ and $\beta_1 x : P(\beta_0 x) \to X$ construct $\overline{\beta} : X \to \widehat{M}^{\mathbb{N}}$ by induction over $\mathbb{N}$:

$$\overline{\beta}_0 x \equiv \bot \qquad\qquad \overline{\beta}_{n+1} x \equiv \sup(\beta_0 x, \overline{\beta}_n \cdot \beta_1 x) \ .$$

Observe that $\overline{\beta}$ makes the diagram

$$\begin{tikzcd} TX \arrow[d, "T\overline{\beta}"'] & X \arrow[l, "\beta"'] \arrow[d, "\overline{\beta}"] \\ T(\widehat{M}^{\mathbb{N}}) \arrow[r, "\alpha"'] & \widehat{M}^{\mathbb{N}} \end{tikzcd} \tag{B.1}$$

commute:

$$\begin{aligned} \alpha_0(T\overline{\beta}(\beta x)) &= \bot = \overline{\beta}_0 x \\ \alpha_{n+1}(T\overline{\beta}(\beta x)) &= \alpha_{n+1}(T\overline{\beta}(\beta_0 x, \beta_1 x)) = \alpha_{n+1}(\beta_0 x, \overline{\beta} \cdot \beta_1 x) \\ &= \sup(\beta_0 x, (\overline{\beta} \cdot \beta_1 x)_n) = \sup(\beta_0 x, \overline{\beta}_n \cdot \beta_1 x) = \overline{\beta}_{n+1} x \ . \end{aligned}$$

Furthermore, $\overline{\beta}$ is the *unique* morphism making (B.1) commute: let g also satisfy $g = \alpha \cdot Tg \cdot \beta$, then

$$
\begin{aligned}
g_0 x &= \alpha_0(Tg(\beta x)) = \perp = \overline{\beta}_0 x \\
g_{n+1} x &= \alpha_{n+1}(Tg(\beta x)) = \alpha_{n+1}(Tg(\beta_0 x, \beta_1 x)) = \alpha_{n+1}(\beta_0 x,\, g \cdot \beta_1 x) \\
&= \sup(\beta_0 x,\, g_n \cdot \beta_1 x) = \sup(\beta_0 x,\, \overline{\beta}_n \cdot \beta_1 x) = \overline{\beta}_{n+1} x \ .
\end{aligned}
$$

This shows that for every coalgebra $\beta : X \to TX$ there exists a unique morphism $\overline{\beta} : X \to M$ satisfying the equation $\alpha \cdot T\overline{\beta} \cdot \beta = \overline{\beta}$.

Note however that α is not an isomorphism, and in particular there is no suitable coalgebra on $\widehat{M}^{\mathbb{N}}$: to construct the final coalgebra we need to define $M \hookrightarrow \widehat{M}^{\mathbb{N}}$ to be the subobject of "well-formed" sequences of trees. To do this we would like to construct a *truncation* morphism $\mathbb{N} \vdash \widehat{M} \to \widehat{M} + 1$ with component at $n : \mathbb{N}$ cutting off elements of $\widehat{M}$ to depth n — the extra value in the codomain represents the result of truncating a tree where $\perp$ occurs anywhere in the body of the cut off tree, so trees being "ill-formed".

In practice it is necessary to define $\overline{M} \equiv \mu X.\, 1 + TX + 1$ with algebra components written $\overline{\perp}$, $\overline{\sup}$ and $\star$ respectively and to construct $\mathrm{trunc} : \widehat{M} \to \overline{M}^{\mathbb{N}}$. This is because the question of whether $\perp$ occurs at an in appropriate depth is in general undecidable, so the simpler form of trunc as a morphism into $\widehat{M} + 1$ discussed above is not implementable.

Define $\overline{\mathrm{trunc}} : \overline{M} \to \overline{M}^{\mathbb{N}}$ by induction over $\overline{M}$ and $\mathbb{N}$ by the following clauses:

$$
\begin{aligned}
\overline{\mathrm{trunc}}_0 x &\equiv \overline{\perp} & \qquad \overline{\mathrm{trunc}}_{n+1} \overline{\perp} &\equiv \star \\
\overline{\mathrm{trunc}}_{n+1}(\overline{\sup}(s, f)) &\equiv \overline{\sup}(s,\, \overline{\mathrm{trunc}}_n \cdot f) & \qquad \overline{\mathrm{trunc}}_{n+1} \star &\equiv \star \ .
\end{aligned}
$$

Note that the construction of $\overline{\mathrm{trunc}}$ is an instance of W-type induction with algebra $[u; v; w] : 1 + T(\overline{M}^{\mathbb{N}}) + 1 \to \overline{M}^{\mathbb{N}}$ defined by induction over $\mathbb{N}$ with $u_0 \equiv v_0(s, f) \equiv w_0 \equiv \perp$, $u_{n+1} \equiv w_{n+1} \equiv \star$ and $v_{n+1}(s, f) \equiv \overline{\sup}(s, f_n)$.

There is an obvious inclusion $\iota : \widehat{M} \hookrightarrow \overline{M}$ defined inductively by:

$$
\iota \perp \equiv \overline{\perp} \qquad\qquad \iota(\sup(s, f)) \equiv \overline{\sup}(s,\, \iota \cdot f) \ .
$$

Finally define $\mathrm{trunc} \equiv \overline{\mathrm{trunc}} \cdot \iota$ which therefore satisfies equations:

$$
\mathrm{trunc}_0 x = \iota \perp \qquad\qquad \mathrm{trunc}_{n+1}(\sup(s, f)) = \overline{\sup}(s,\, \mathrm{trunc}_n \cdot f) \ .
$$

We can now say that $m : \widehat{M}^{\mathbb{N}}$ is "well-formed" iff each m_n is a truncation to depth n of all the larger trees m_{n+k}, which can be captured as $\forall n : \mathbb{N}.\, (\iota m_n = \mathrm{trunc}_n m_{n+1})$. So define

$$
M \equiv \sum m : \widehat{M}^{\mathbb{N}}.\, \prod n : \mathbb{N}.\, (\iota m_n = \mathrm{trunc}_n m_{n+1}) \ , \tag{B.2}
$$

describing a regular subobject of $\widehat{M}^{\mathbb{N}}$. Note that for $(s,f):TM$ the equation above translates into the equation $\iota \cdot f_n = \mathrm{trunc}_n \cdot f_{n+1}$; this can be used to show that α restricts to $\alpha : TM \to M$, ie $\iota(\alpha_n x) = \mathrm{trunc}_n(\alpha_{n+1}x)$ for $x : TM$, thus:

$$\begin{aligned}
\iota(\alpha_0(s,f)) &= \iota\bot = \mathrm{trunc}_0(\alpha_1(s,f)) \\
\iota(\alpha_{n+1}(s,f)) &= \iota(\mathrm{sup}(s,f_n)) = \overline{\mathrm{sup}}(s,\, \iota \cdot f_n) = \overline{\mathrm{sup}}(s,\, \mathrm{trunc}_n \cdot f_{n+1}) \\
&= \mathrm{trunc}_{n+1}(\mathrm{sup}(s,f_{n+1})) = \mathrm{trunc}_{n+1}(\alpha_{n+2}(s,f)) \quad .
\end{aligned}$$

For the rest of this proof we'll write α for the restricted morphism $\alpha : TM \to M$. The morphism $\overline{\beta}$ constructed from a coalgebra β also factors through $M \hookrightarrow \widehat{M}^{\mathbb{N}}$:

$$\begin{aligned}
\iota(\overline{\beta}_0 x) &= \iota\bot = \mathrm{trunc}_0(\overline{\beta}_{n+1} x) \\
\iota(\overline{\beta}_{n+1} x) &= \iota(\mathrm{sup}(\beta_0 x,\, \overline{\beta}_n \cdot \beta_1 x)) = \overline{\mathrm{sup}}(\beta_0 x,\, \iota \cdot \overline{\beta}_n \cdot \beta_1 x) \\
&= \overline{\mathrm{sup}}(\beta_0 x,\, \mathrm{trunc}_n \cdot \overline{\beta}_{n+1} \cdot \beta_1 x) = \mathrm{trunc}_{n+1}(\mathrm{sup}(\beta_0 x,\, \overline{\beta}_{n+1} \cdot \beta_1 x)) \\
&= \mathrm{trunc}_{n+1}(\overline{\beta}_{n+2} x)
\end{aligned}$$

showing that $\iota \cdot \overline{\beta}_n = \mathrm{trunc}_n \cdot \beta_{n+1}$. Now writing $\overline{\beta} : X \to M$ we can see that $\overline{\beta}$ is still the unique solution to the equation $\overline{\beta} = \alpha \cdot T\overline{\beta} \cdot \beta$; to complete the proof it remains to show that α is an isomorphism.

By definition (B.2) a term $m : M$ satisfies the equation $\iota m_{n+1} = \mathrm{trunc}_{n+1} m_{n+2}$; by disjointness of coproducts and the definition of trunc_{n+1} we can see that this equation must be of the form $\iota m_{n+1} = \overline{\mathrm{sup}}(s, \mathrm{trunc}_n \cdot f_{n+1})) = \mathrm{trunc}_{n+1}(\mathrm{sup}(s,f_{n+1})) = \mathrm{trunc}_{n+1} m_{n+2}$ for some s and f_{n+1}. We can therefore write $m_{n+1} = \mathrm{sup}(s,f_n)$ where f_n satisfies the equation $\iota \cdot f_n = \mathrm{trunc}_n \cdot f_{n+1}$. By defining $\alpha' m \equiv (s,f)$ we obtain a morphism $\alpha' : M \to TM$.

Now $\alpha'(\alpha(s,f)) = (s',f')$ where $\mathrm{sup}(s',f'_n) = \alpha_{n+1}(s,f) = \mathrm{sup}(s,f_n)$, showing that $\alpha' \cdot \alpha = \mathrm{id}_{TM}$. Conversely, writing $\alpha' m = (s,f)$ where $m_{n+1} = \mathrm{sup}(s,f_n)$ and $\iota \cdot f_n = \mathrm{trunc}_n \cdot f_{n+1}$ we can show that $\alpha(\alpha' m) = m$:

$$\begin{aligned}
\iota(\alpha_0(\alpha' m)) &= \iota\bot = \mathrm{trunc}_0 m_1 = \iota m_0 \\
\iota(\alpha_{n+1}(\alpha' m)) &= \iota(\alpha_{n+1}(s,f)) = \iota(\mathrm{sup}(s,f_n)) = \overline{\mathrm{sup}}(s,\, \iota \cdot f_n)) \\
&= \overline{\mathrm{sup}}(s,\, \mathrm{trunc}_n \cdot f_{n+1}) = \mathrm{trunc}_{n+1}(\mathrm{sup}(s,f_{n+1})) \\
&= \mathrm{trunc}_{n+1} m_{n+2} = \iota m_{n+1} \quad .
\end{aligned}$$

Thus $\alpha' = \alpha^{-1}$ and we see that M is a final coalgebra for $[\![S \triangleright P]\!]$. □