Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 342 (2005) 3–27
Theoretical Computer Science
www.elsevier.com/locate/tcs

Containers: Constructing strictly positive types

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

[a] *Diamond Light Source, Rutherford Appleton Laboratory, UK*
[b] *School of Computer Science and Information Technology, Nottingham University, UK*
[c] *Department of Mathematics and Computer Science, University of Leicester, UK*

Abstract

We introduce the notion of a *Martin-Löf category*—a locally cartesian closed category with disjoint coproducts and initial algebras of container functors (the categorical analogue of W-types)—and then establish that nested strictly positive inductive and coinductive types, which we call *strictly positive types*, exist in any Martin-Löf category.

Central to our development are the notions of *containers* and *container functors*. These provide a new conceptual analysis of data structures and polymorphic functions by exploiting dependent type theory as a convenient way to define constructions in Martin-Löf categories. We also show that morphisms between containers 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.

Keywords: 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 in many situations it is sufficient and indeed often easier to restrict ourselves to total functions, thus allowing us to view types as sets. David

* Corresponding author. Tel.: +44 115 84 66516; fax: +44 115 951 4254.
E-mail addresses: michael@araneidae.co.uk (M. Abbott), txa@cs.nott.ac.uk (T. Altenkirch), ng13@mcs.le.ac.uk (N. Ghani).

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.06.002

Turner [37] calls this approach *strong functional programming*, though *total* might have been a better word. Not all recursive types make sense in this view, for example we can hardly understand $D \cong 1 + (D \to D)$ as a set. Moreover, even if we restrict ourselves to well behaved types like lists over A which are a solution to $\mathrm{List}\ A \cong 1 + A \times \mathrm{List}\ A$ (since every element of a list is either nil or a cons of an element of A and a list), 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, i.e. the functor corresponding to a datatype declaration, which in the case of lists over A is $X \mapsto 1 + A \times X$. We write this initial algebra as $\mathrm{List}\ A = \mu X.\ 1 + A \times X$.

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

In this paper we investigate **strictly positive types** which we define to be those types which can be formed using 0, 1, $+$, $\times$, $\to$, 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: the 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 $\mathrm{Ord} \equiv \mu X.\ 1 + X + (\mathbb{N} \to X) = \mu Y.\ 1 + Y + ((\mu X.\ 1 + X) \to Y)$, 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 unary container is given by a type of *shapes* S and a family of *position* types indexed by S thus: $s : S \vdash Ps$. As a container we write this as $(s : S \rhd Ps)$ or just $(S \rhd P)$. The *extension* of this container is a functor $[\![S \rhd P]\!]$, which on objects is given by $[\![S \rhd P]\!]X = \sum s : S.\, (P\, s \to X)$. We say that any functor naturally isomorphic to the extension of some container is a *container functor*.

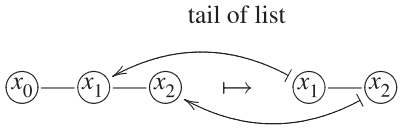
Thus for any type X an element of $\sum s : S.\, (Ps \to X)$ is a pair (s, f) where $s : S$ is a shape and $f : Ps \to X$ is a function assigning an element of X to each position for the corresponding shape Ps. For example List is represented by the container $(n : \mathbb{N} \rhd \mathrm{Fin}\, n)$ where $\mathrm{Fin}\, n = \{0, 1, \ldots, n-1\}$: a list is given by the length (its shape) and a function which assigns an element to each position in the list. List^∞ is represented by $(n : \mathbb{N}^\infty \rhd \mathrm{Fin}'\, n)$ where $\mathbb{N}^\infty \equiv \nu X.1+X$ is the set of co-natural numbers extending the usual natural numbers by an infinite element $\infty = 1+\infty$ and Fin' extends Fin by $\mathrm{Fin}'\, \infty \equiv \mathbb{N}$, that is the positions of an infinite list are the natural numbers. We show (Corollary 6.1) that all strictly positive types can be represented as containers.

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

- $u : S \to T$ is a function on shapes,
- $g : \prod s : S.\, Q(us) \to Ps$ 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, and conversely (Theorem 3.4) every natural transformation between containers arises from a unique container morphism. As an example the reverse list function $\mathrm{rev}_A : \mathrm{List}\ A \to \mathrm{List}\ A$ is represented

by the container morphism $(\mathrm{id}_{\mathbb{N}}, r)$ where $r : \prod n : \mathbb{N}.\ (\mathrm{Fin}\, n \to \mathrm{Fin}\, n)$ is defined as $rni \equiv n-1-i$. 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. Consider for example 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



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 [3,6].

We use here the language of extensional Martin-Löf Type Theory [28] with W-types and a constant inhabiting $\mathsf{true} \neq \mathsf{false}$ (**MLW**$^{\mathbf{ext}}$, see [8]) as the internal language of locally cartesian closed categories with disjoint coproducts and initial algebras of unary container functors—we call these *Martin-Löf categories*.

The present paper is the journal version of our conference papers [2,4]; this paper extends our previous results. We show here that W-types are sufficient to represent *all* strictly positive types allowing arbitrary nestings of μ and ν (Corollary 5.5). Thus, we improve on our previous results in two ways:

- In [2] we required that the ambient category have infinite limits and colimits (or at least be accessible), which rules out many interesting examples including syntactic categories of Martin-Löf type theory, categories of ω-sets, categories of PERs and realisability toposes.
- In [4] we show that nested μ-types can be represented using W-types, but did not consider ν-types or M-types.

The extension to ν-types is non-trivial: it follows from Proposition 5.2 which is stronger than the corresponding Proposition 6.1 in [4]—here 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 W-types, which we do in Proposition 4.1.

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 [23] 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 us whether there is a more formal relation between the two approaches.

Our work is clearly related to the work of Joyal [26] on species and analytical functors whose relevance for Computer Science has been recently noticed by Hasegawa [19]. 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.

Dybjer [17] has shown that non-nested inductive types can be encoded with W-Types. His construction is a sub-case of Corollary 6.1 covering only 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 [18] have put our results in a more general context and indeed their Theorem 12 generalises our Proposition 5.2 to dependently typed containers, which they call dependent polynomial functors. Similarly, their Theorem 14 is closely related to our Proposition 5.3. We also learnt from their work that this construction is related to the proof in Moerdijk and Palmgren [31] that W-types localise to slice categories.

After learning about our Proposition 4.1 that M-types are derivable from W-types, van den Berg and de Marchi [38] 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 containers and prove some basic properties such as the representation theorem and closure under polynomial operations. In Section 4 we show that M-types are derivable from W-types, and finally the core of the paper is Section 5 where we show that container types are closed under μ and ν. We close with conclusions and discuss further work.

2. Background

2.1. The categorical semantics of dependent types

This paper can be read in two ways (see Proposition 2.5):

(1) as a construction within the extensional type theory **MLW**$^{\mathbf{ext}}$ (see [8]) with finite types, W-types, a proof of $\mathsf{true} \neq \mathsf{false}$ and no universes;
(2) as a construction in the internal language of locally cartesian closed categories with disjoint coproducts and initial algebras of container functors in one variable—we call these **Martin-Löf categories**.

The key idea of 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.[1] The details of this construction can be found in [35,36,20,22,24,1]; see also [16] on internal languages. In particular, Seely [35] allows us to treat Martin-Löf type theory (without W-types) as the internal language of a locally cartesian closed category.

[1] Note that an important technicality [20] means that a type $A \vdash B$ cannot strictly be identified with its display map $\pi_B \in \mathbb{C}/A$, instead a single display map may arise from many isomorphic types. However, to avoid excessive pedantry, in the presentation of this paper we will identify $\mathbb{C}/A$ with the equivalent category of types over A.

Elements of A (in a context U) 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 along f is written as $u : U \vdash B(fu)$. When the variable $a : A$ is clear (and can be elided) we may write B instead of Ba, and similarly $B(fu)$ can be written as f^*B when u is elided—thus linking the type theoretic notation directly back to the underlying categorical interpretation. Thus we can write $a : A \vdash Ba$ or even just $A \vdash B$ for $B \in \mathbb{C}/A$, occasionally omitting variables from the internal language for conciseness where practical.

Note that substitution by pullback extends to a functor $f^* : \mathbb{C}/A \to \mathbb{C}/U$: to simplify the presentation we will assume that substitution corresponds precisely to a choice of pullback, but for a more detailed treatment of the issues involved see [13,20,1].

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 when a is elided. 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 we write $a : A \vdash fa : Ba \cong Ca$ for an isomorphism.

The morphism in $\mathbb{C}$ associated with $B \in \mathbb{C}/A$ will be written as $\pi_B : \sum_A B \to A$ (this is also known as 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 will 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 $\Gamma \vdash u : Bt$.

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 $\Gamma, a, b : A \vdash a = b$. Write $\mathsf{refl}_a : a = a$. Note that we work with an extensional type theory where equality in the type theory coincides with equality of morphisms in $\mathbb{C}$. We believe that our development could also be implemented in an intensional system [27,32] by using setoids [21].

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

Thus we can interpret the language of the dependently typed lambda calculus as the **internal language** of a locally cartesian closed category: this is captured in [1, Proposition 3.3.5] and is the basis of the claim in [35, Theorem 6.3] that locally cartesian closed categories are equivalent to Martin-Löf theories (without W-types). As pointed out by Hofmann [20] this claim is not strictly accurate [2]: a more careful treatment requires the machinery of fibrations.

[2] The problem is that substitution cannot be identified with taking pullbacks unless this operation can be made strictly associative, which is not in general possible. The details of this problem and its solution are covered in detail in [20] and in Chapters 2 and 3 of [1], and is related to the observations in footnotes 1 and 3.

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 $A \xrightarrow{\mathsf{inl}} A + B \xleftarrow{\mathsf{inr}} B$ 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 $(A \vdash \mathsf{inl}^* C,\ B \vdash \mathsf{inr}^* C)$ is an equivalence: write $- \overset{\circ}{+} -$ for the inverse functor. Thus given $A \vdash B$ and $C \vdash D$ (with display maps π_B and π_D) we write $A + C \vdash B \overset{\circ}{+} D$ for their disjoint sum, and we can see that $\pi_{B \overset{\circ}{+} D} \cong \pi_B + \pi_D$ as objects of $\mathbb{C}/A + C$, where $\pi_B + \pi_D : \Sigma_A B + \Sigma_C D \to A + C$ is a sum of morphisms.

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

We write $\prod_{i \in I} A_i$ and $\sum_{i \in I} A_i$ for finite products and coproducts (with projections $\pi_j : \prod_{i \in I} A_i \to A_j$ and coprojections $\mathsf{inl}_j : A_j \to \sum_{i \in I} A_i$ for $j \in I$) indexed by a finite set I, and we write a disjoint finite sum of families as $\sum_{i \in I} A_i \vdash \biguplus_{i \in I} B_i$.

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

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

$$\prod a : A.\, \sum b : Ba.\, C(a, b) \cong \sum f : \prod a : A.\, Ba.\, \prod a : A.\, C(a, fa), \tag{IC1}$$

$$\prod_{i \in I} \sum b : B_i.\, C_i b \cong \sum a : \prod_{i \in I} B_i.\ \prod_{i \in I} C_i(\pi_i a), \tag{IC2}$$

$$\prod a : A.\, (Ba \to C) \cong \left(\sum a : A.\, Ba\right) \to C, \tag{Cu1}$$

$$\prod_{i \in I} (B_i \to C) \cong \left(\sum_{i \in I} B_i\right) \to C, \tag{Cu2}$$

$$\left(\biguplus_{i \in I} B_i\right)(\mathsf{inl}_i\, a) \cong B_i a, \tag{DC1}$$

$$\sum_{i \in I} \sum a : A_i.\, C(\mathsf{inl}_i\, a) \cong \sum a : \sum_{i \in I} A_i.\, Ca. \tag{DC2}$$

We will need to make some explicit use of the machinery of fibrations, so recall [12,13,33,15,24,1] that a *(split) fibration*[3] $\mathbb{E}$ over a category $\mathbb{C}$ is given by assigning to each object $\Gamma \in \mathbb{C}$ a category $\mathbb{E}_\Gamma$, the *fibre over* Γ, together with for each morphism

[3] More generally a *(cloven) fibration* is defined by relaxing the equations between reindexing functors to natural isomorphisms together with coherence equations; however, we can regard all the fibrations in this paper (the category $\mathbb{C}$ fibred over itself and its I-fold product $\mathbb{C}^I$) as split via the technique described in [20]. Similarly we will without further comment take our fibred functors and natural transformations to be split.

$\gamma : \Delta \to \Gamma$ in $\mathbb{C}$ a functor $\gamma^* : \mathbb{E}_\Gamma \to \mathbb{E}_\Delta$, the *reindexing functor* over γ, satisfying equations $\mathrm{id}^*_\Gamma = \mathrm{id}_{\mathbb{C}/\Gamma}$ and $(\gamma \cdot \delta)^* = \delta^*\gamma^*$. A *fibred functor* $F : \mathbb{D} \to \mathbb{E}$ between fibrations $\mathbb{D}$ and $\mathbb{E}$ over $\mathbb{C}$ assigns to each $\Gamma \in \mathbb{C}$ a functor $F_\Gamma : \mathbb{D}_\Gamma \to \mathbb{E}_\Gamma$ such that for each $\gamma : \Delta \to \Gamma$ the equation $\gamma^* F_\Gamma = F_\Delta \gamma^*$ holds. Similarly, a *fibred natural transformation* $\alpha : F \to G$ between fibred functors is a family of natural transformations $\alpha_\Gamma : F_\Gamma \to G_\Gamma$ such that $\gamma^* \alpha_\Gamma = \alpha_\Delta \gamma^*$.

Given a (finite) index set I define $[\mathbb{C}^I, \mathbb{C}^J]$ to be the category of *fibred* functors and natural transformations $\mathbb{C}^I \to \mathbb{C}^J$ 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 $[\mathbb{C}^I, \mathbb{C}]$; observe also that $[\mathbb{C}^I, \mathbb{C}^J] \cong [\mathbb{C}^I, \mathbb{C}]^J$, and so most of our development can be done with $J = 1$.

2.2. W-types and M-types

In Martin-Löf's Type Theory [28,32] 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 W-types in type theory is through one type forming rule, an introduction rule and an elimination rule, together with an equation. We refer to [1, Chapter 2] for the precise rules of the Type Theory we are using, which are basically standard (see also [8]). However, it is worthwhile to remind the reader of the rules covering W-types, quoting from Abbott [1, Definition 5.2.1]:

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

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

together with a constructor term

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

and an elimination rule

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

satisfying the following equation for variables $a : A$ and $f : Ba \to \mathsf{W}_A B$:

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

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

Note that the elimination rule together with equality types ensures that wrec_h is unique, and it is easy to see that the rule wrec implies that sup is an initial algebra on $\mathsf{W}_A B$ for the functor $X \mapsto \sum a : A.\,(Ba \to X)$; it is not much harder to see that W-types can be constructed from initial algebras [1, Theorem 5.2.2]. Moerdijk and Palmgren [31] show that the global version of W-types implies the existence of W-types in each slice. Functors of this form play a special role in this paper: the following definition is justified in Definition 3.2 and its sequel.

Definition 2.3. A functor $F : \mathbb{C} \to \mathbb{C}$ is a *container functor* iff it is naturally isomorphic to a functor of the form $X \mapsto \sum a : A.\,(Ba \to X)$, or more concisely $\sum_A (B \to X)$, for some family $A \vdash B$ in $\mathbb{C}$, i.e., objects $A \in \mathbb{C}$ and $B \in \mathbb{C}/A$.

We consider that the existence of initial algebras for container functors summarises the essence of Martin-Löf's Type Theory (without universes) from a categorical perspective, hence the following definition.

Definition 2.4. A *Martin-Löf category* is a locally cartesian closed category with disjoint coproducts and initial algebras for container functors, in other words closed under the formation of W-types.

Thus the relation between Martin-Löf categories and the syntax of type theory can be summarized by the following proposition, the proof of which is implicit in [1].

Proposition 2.5. *Extensional dependent type theory with Sigma-types, Pi-types, W-types, a proof of* $\mathsf{true} \neq \mathsf{false}$ *and no universes is the internal language of Martin-Löf categories.*

Dually, we introduce M-types as the terminal coalgebras of container 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; thus the following definition is purely categorical. We will see in Proposition 4.1 that every Martin-Löf category has M-types.

Definition 2.6. A locally cartesian closed category has *M-types* iff it has final coalgebras for container functors. The M-type for a container functor $X \mapsto \sum a : A.\,(Ba \to X)$ will be written as $\mathsf{M}_A B$ with coalgebra $\mathsf{sup}^{-1} : \mathsf{M}_A B \to \sum_A (B \to \mathsf{M}_A B)$.

Note that the M-type coalgebra sup^{-1} is, as its name suggests, the inverse of a constructor sup. This means that for both W- and M-types the constructor is written as sup; where it is necessary to distinguish them we will write sup^{μ} and sup^{ν} respectively.

We know that W-types exist in toposes with a natural numbers object [31, Proposition 3.6] and in categories which are both locally cartesian closed and locally accessible [2, Theorem 6.8]. Moreover, W-types exist in models of Type Theory based on realisability such as the categories ω-**Set** of ω-sets and **PER** of partial equivalence relations on $\mathbb{N}$ (equivalent to the full subcategory of ω-**Set** of *modest* sets). See Jacobs [24] for the definitions of ω-**Set** and **PER** and the verifications that they are locally cartesian closed [24, ex. 1.2.7]; the fact

that W-types exist can be seen by modifying the construction in [9, pp. 79–80]. It is easy to see that coproducts in these categories are disjoint and hence we have:

Proposition 2.7. *ω-**Set** and **PER** are Martin-Löf categories.*

On the other hand, note that both these categories lack coequalisers and most infinite limits. In particular ω-limits do not in general exist in ω-**Set** or **PER**: it's easy to see that $\prod_{n\in\mathbb{N}} 2$ must have $2^{\mathbb{N}}$ elements, but all objects in **PER** are countable so it cannot be an object of **PER**. The limit in ω-**Set**, if it exists, is modest and hence would correspond to an object in **PER**.

2.3. Strictly positive types

Strictly positive types can be inductively defined as follows.

Definition 2.8. A *strictly positive type in n variables* [7] 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 (i.e. one with no type variables) then K is a strictly positive type;
- each type variable X_i is a strictly positive type;
- if F, G are strictly positive types then so are $F + G$ and $F \times G$;
- if K is a constant type and F a strictly positive type then $K \to F$ is a strictly positive type;
- if F is a strictly positive type in $n+1$ variables then $\mu X.\ F$ and $\nu X.\ F$ are strictly positive types in n variables (for X any type variable).

Define a *non-inductive* strictly positive type to be built up inductively as without any application of μ or ν: from constant types K, variables X, products $\times$, coproducts $+$ and function types from a constant type $K \to -$.

As we will show, non-inductive strictly positive types can be interpreted in any locally cartesian closed category with disjoint coproducts (this already follows from Dybjer [17]) and (general) strictly positive types can be interpreted in any Martin-Löf category.

3. Basic properties of containers

Throughout this section we will take as given a locally cartesian closed category $\mathbb{C}$ with finite disjoint coproducts. We will now introduce the category of containers $\mathcal{G}$ equipped with its interpretation or *extension* functor $[\![-]\!] : \mathcal{G} \to [\mathbb{C}, \mathbb{C}]$. When constructing fixed points it is also necessary to take account of containers with parameters, so we define $[\![-]\!] : \mathcal{G}_I \to [\mathbb{C}^I, \mathbb{C}]$ for each parameter index set I. For the purposes of this paper the index set I can be assumed to be finite, 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.

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

- Objects are pairs $\left(A \in \mathbb{C},\ B \in (\mathbb{C}/A)^I\right)$; write this as $(A \triangleright B) \in \mathcal{G}_I$.
- 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 I parameters consists of a single shape A together with a family of positions $A \vdash B_i$ for each $i \in I$.

A container $(A \triangleright B) \in \mathcal{G}_I$ can be written using type theoretic notation as a pair $\vdash A$ and $i : I,\ a : A \vdash B_i a$, and similarly a morphism $(u, f) : (A \triangleright B) \to (C \triangleright D)$ can be written as a pair $\vdash u : A \to C$ and $i : I,\ a : A \vdash f_i a : D_i(ua) \to B_i a$.

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

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

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

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

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

Say that a functor $F : \mathbb{C}^I \to \mathbb{C}$ is a *container functor* if it is naturally isomorphic to a functor of the form $[\![A \triangleright B]\!]$ for some container $(A \triangleright B)$ (see also Definition 2.3).

The following proposition follows from the construction of $[\![-]\!]$ as a type expression: that $[\![F]\!]$ is fibred means that for any $\Gamma \vdash X$ we can construct $\Gamma \vdash [\![F]\!]X$, and that given any substitution $\gamma : \Delta \to \Gamma$ we can write $\gamma^*([\![F]\!]X) = [\![F]\!](\gamma^* X)$. This is simply a categorical statement of the fairly obvious observation that substitution through $[\![F]\!]$ works.

Proposition 3.3. *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 $\mathbb{C}$.*

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.

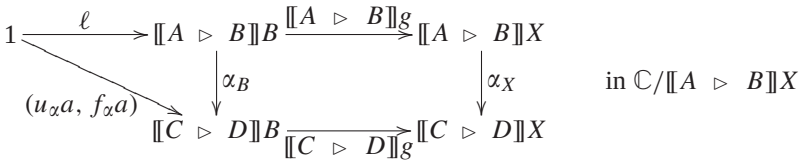
Theorem 3.4 (*Representation*). *The functor $[\![-]\!] : \mathcal{G}_I \to [\mathbb{C}^I, \mathbb{C}]$ is full and faithful.*

Proof. To show that $[\![-]\!]$ 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 $[\![-]\!]$.

Given $\alpha : [\![A \triangleright B]\!] \to [\![C \triangleright D]\!]$ define $\ell \equiv (a, \mathrm{id}_{Ba}) : [\![A \triangleright B]\!]B$ 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 $A \vdash \alpha_B \ell : [\![C \triangleright D]\!]B = \sum_C \prod_I (D \to B)$; write this as $\alpha_B \ell = (u_\alpha, f_\alpha)$, where $u_\alpha a : C$ and $f_\alpha a : \prod_{i \in I}(D_i(u_\alpha a) \to B_i a)$ 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 $[\![-]\!]$.

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 $g : \prod_{i \in I}(B_i a \to X_i)$ be given, consider the diagram



and evaluate

$$\begin{aligned}\alpha_X(a, g) &= \alpha_X(([\![A \triangleright B]\!]g)\ell) = ([\![C \triangleright D]\!]g)(\alpha_B \ell) \\ &= ([\![C \triangleright D]\!]g)(u_\alpha a, f_\alpha a) \\ &= (u_\alpha a, g \cdot f_\alpha a) = [\![u_\alpha, f_\alpha]\!]_X(a, g).\end{aligned}$$

This shows that $\alpha = [\![u_\alpha, f_\alpha]\!]$ as required. □

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 $(1 \triangleright n)$ and hence there is a bijection between polymorphic functions $X^n \to X^m$ and functions $m \to n$. Similarly, any polymorphic function List $X \to$ 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 $\mathcal{G}_I$ inherits products and coproducts from $\mathbb{C}$, and that $[\![-]\!]$ preserves them:

Proposition 3.5. *If $\mathbb{C}$ has finite products and coproducts then $\mathcal{G}_I$ has finite products and coproducts and they are preserved by $[\![-]\!]$.*

Proof. Since $[\![-]\!]$ is full and faithful we can reflect the construction of products and coproducts along $[\![-]\!]$, by showing that products and coproducts of objects in $[\mathbb{C}^I, \mathbb{C}]$ in the image of $[\![-]\!]$ are themselves in the image of $[\![-]\!]$.

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

$$\begin{aligned}\prod_{k\in K}[\![A_k \rhd B_k]\!]X &= \prod_{k\in K}\sum a : A_k.\ \prod_{i\in I}(B_{k,i}a \to X_i)\\ &\cong \sum a : \prod_{k\in K} A_k.\ \prod_{k\in K}\prod_{i\in I}(B_{k,i}(\pi_k a) \to X_i)\\ &\cong \sum a : \prod_{k\in K} A_k.\ \prod_{i\in I}\left(\left(\sum_{k\in K} B_{k,i}(\pi_k a)\right) \to X_i\right)\\ &= \left[\!\!\left[\prod_{k\in K} A_k \rhd \sum_{k\in K}(\pi_k^*)^I B_k\right]\!\!\right]X\end{aligned}$$

showing by reflection along $[\![-]\!]$ that

$$\begin{aligned}\prod_{k\in K}(A_k \rhd B_k) &\cong \left(\prod_{k\in K} A_k \rhd \sum_{k\in K}(\pi_k^*)^I B_k\right)\\ &= \left(a : \prod_{k\in K} A_k \rhd \sum_{k\in K} B_k(\pi_k a)\right).\end{aligned}$$

Coproducts: Given a family $(A_k \rhd B_k)_{k\in K}$ of objects in $\mathcal{G}_I$ calculate (making essential use of disjoint coproducts):

$$\begin{aligned}\sum_{k\in K}[\![A_k \rhd B_k]\!]X &= \sum_{k\in K}\sum a : A_k.\ \prod_{i\in I}(B_{k,i}a \to X_i)\\ &\cong \sum_{k\in K}\sum a : A_k.\ \prod_{i\in I}\left(\left(\coprod_{k'\in K} B_{k',i}\right)(\mathsf{inl}_k\, a) \to X_i\right)\\ &\cong \sum a : \sum_{k\in K} A_k.\ \prod_{i\in I}\left(\left(\coprod_{k\in K} B_{k,i}\right)a \to X_i\right)\\ &= \left[\!\!\left[\sum_{k\in K} A_k \rhd \left(\coprod_{k\in K} B_{k,i}\right)_{i\in I}\right]\!\!\right]X\end{aligned}$$

showing by reflection along $[\![-]\!]$ that

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

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

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

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

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

$$\begin{aligned}&(S \rhd P, Q)[(A \rhd B)]\\ &\quad\equiv \left(s : S,\ f : Qs \to A \rhd \left(P_i s + \sum q : Qs.\ B_i(fq)\right)_{i\in I}\right).\end{aligned}$$

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

Proposition 3.6. *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 $s : S$ throughout):

$$\begin{aligned}
&[\![S \triangleright \boldsymbol{P}, Q]\!]\big[[\![A \triangleright B]\!]\big]X \\
&\quad = \sum_S \left(\left(\prod_{i\in I} X_i^{P_i}\right) \times \left(Q \to \sum a : A.\ \prod_{i\in I} X_i^{B_i a}\right)\right) \\
&\quad \cong \sum_S \left(\left(\prod_{i\in I} X_i^{P_i}\right) \times \left(\sum f : A^Q.\ \prod q : Q.\ \prod_{i\in I} X_i^{B_i(fq)}\right)\right) \\
&\quad \cong \sum_S \sum f : A^Q.\ \prod_{i\in I} \left(X_i^{P_i} \times \prod q : Q.\ X_i^{B_i(fq)}\right) \\
&\quad \cong \sum_S \sum f : A^Q.\ \prod_{i\in I} \left(\left(P_i + \sum q : Q.\ B_i(fq)\right) \to X_i\right) \\
&\quad \cong [\![(S \triangleright \boldsymbol{P}, Q)[(A \triangleright B)]]\!]X.
\end{aligned}$$

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

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 $- \circ - : \mathcal{G}_I \times \mathcal{G}_J^I \to \mathcal{G}_J$ 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 $\mathcal{G}_I^J$ [1, Proposition 4.4.4].

A canonical form for terms of type $[\![F]\!][[\![G]\!]]X \cong [\![F[G]]\!]X$ will be helpful later on. Observe that either side of this isomorphism can be written as $\theta(s, f, g, h)$ for some suitable and easy to compute isomorphism θ, with components of the following types:

$$s : S \qquad f : Qs \to A \qquad g : Ps \to X \qquad h : \prod q : Qs.\ (B(fq) \to X). \tag{1}$$

Now we look at the treatment of type variables—this gives us a notion of weakening of containers as type expressions. First note that every type variable X_i can be regarded as a container.

Proposition 3.7. *Every projection functor $\pi_i : \mathbb{C}^I \to \mathbb{C}$ defined by $\pi_i \boldsymbol{X} \equiv X_i$ for each $i \in I$ is a container functor.*

Proof.

$$[\![1 \triangleright (i = j)_{j\in I}]\!]\boldsymbol{X} \cong \prod_{j\in I}((i = j) \to X_j) \cong X_i. \qquad \square$$

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

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

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

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 $i \leqslant n$ 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.

Similarly, we can write $\uparrow K \equiv \uparrow^{¡_I} K \cong (K \triangleright 0) \in \mathcal{G}_I$ (where $¡_I : 0 \to I$) for what can sensibly be called a *constant container*—its extension is a constant functor equal to K. We can now show that containers are closed under exponentiation by constant containers.

Proposition 3.9. *Containers are closed under exponentiation by constant containers, and this is preserved by $[\![-]\!]$: given $F \in \mathcal{G}_I$ then $[\![\uparrow K \to F]\!]X \cong K \to [\![F]\!]X$.*

Proof. Let $F = (A \triangleright B)$ and calculate

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

If we now define $\uparrow K \to F \equiv (f : K \to A \triangleright (\sum k : K.\ B_i(fk))_{i\in I})$ (or write this as just $K \to F$) then by reflection along $[\![-]\!]$ and the isomorphism $[\![G]\!] \times K \cong [\![G \times \uparrow K]\!]$ (for any $G \in \mathcal{G}_I$) we can see that $K \to F$ is the required exponential. $\square$

The following proposition is now an obvious consequence of the constructions and results in this section; this is basically a reformulation of the main result of [17] using the language of containers.

Corollary 3.10. *Every non-inductive strictly positive type F in n variables can be interpreted as an n-ary container $(\!|F|\!) \in \mathcal{G}_n$ (and an n-ary functor $[\![(\!|F|\!)]\!] : \mathbb{C}^n \to \mathbb{C}$) such that $(\!|K|\!) = K$, $(\!|F+G|\!) = (\!|F|\!)+(\!|G|\!)$, $(\!|F\times G|\!) = (\!|F|\!)\times(\!|G|\!)$, $(\!|K \to F|\!) = K \to (\!|F|\!)$ and $[\![(\!|X_i|\!)]\!](X_1,\ldots,X_n) = X_i$.*

4. Constructing M-types from W-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

$$1 \longleftarrow T1 \longleftarrow \cdots \longleftarrow T^n1 \longleftarrow \cdots \longleftarrow \varprojlim_{n\in\mathbb{N}} T^n1, \tag{2}$$

then as T preserves ω-limits (indeed T preserves all connected limits since the functor $\sum_S$ also does) it is a well known result (e.g. [34]) that $\nu T \equiv \varprojlim_{n\in\mathbb{N}} T^n1$ is a final coalgebra. This approach was taken in [2,1].

In the present treatment we do not wish to assume the existence of ω-limits: recall that the Martin-Löf categories ω-**Set** and **PER** do not have *external* ω-limits of the form (2), and the same problem applies to the effective topos. One possible approach is to construct the family $n : \mathbb{N} \vdash T^n1$ as a family in $\mathbb{C}$ together with an internal representation of the restriction morphisms $T^{n+k}1 \to T^n1$ 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 $\pi_n : \nu T \to T^n1$ takes a potentially infinite tree and truncates it to depth n; such truncated trees can be expressed as elements of the W-type $\widehat{M} \equiv \mu X.\ 1 + TX$. Writing $\bot$ and sup for the two components of the constructor $1 + T\widehat{M} \to \widehat{M}$, we can define an inclusion $i_n : T^n1 \rightarrowtail \widehat{M}$ inductively with $i_0 \equiv \bot$ and $i_{n+1}(s, f) \equiv \mathsf{sup}(s, i_n \cdot f)$.

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 is 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 proposition.

Proposition 4.1. *Every Martin-Löf category is closed under the formation of M-types, that is, every unary container functor has a final coalgebra.*

Proof. Let $A \vdash B$ be the family for which $\mathsf{M}_A B \equiv \nu X.\ [\![A \triangleright B]\!]X$ is to be constructed; for conciseness, write $TX \equiv [\![A \triangleright B]\!]X = \sum_A X^B$ throughout this proof. Define $\widehat{M} \equiv \mu X.\ 1 + TX$, writing $\bot : \widehat{M}$ and $\mathsf{sup} : T\widehat{M} \to \widehat{M}$ 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 $m : \mathsf{M}_A B$ 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 construct a T-algebra $\alpha : T(\widehat{M}^{\mathbb{N}}) \to \widehat{M}^{\mathbb{N}}$ by cases over $\mathbb{N}$:

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

with variables $a : A$ and $f : Ba \to \widehat{M}^{\mathbb{N}}$. We define $f_n \equiv \lambda b : Ba.\,(fb)n$—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 $\mathsf{M}_A B$.

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

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

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

$$\begin{array}{ccc} TX & \xleftarrow{\ \beta\ } & X \\ {\scriptstyle T\overline{\beta}}\downarrow & & \downarrow{\scriptstyle \overline{\beta}} \\ T(\widehat{M}^{\mathbb{N}}) & \xrightarrow[\ \alpha\ ]{} & \widehat{M}^{\mathbb{N}} \end{array} \tag{3}$$

commute:

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

Furthermore, $\overline{\beta}$ is the *unique* morphism making (3) commute: let g also satisfy $g = \alpha \cdot Tg \cdot \beta$, then

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

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 $\bot$ occurs anywhere in the body of the cut off tree.

In practice it is necessary to define $\overline{M} \equiv \mu X.\ 1 + TX + 1$ with algebra components written $\overline{\bot}$, $\overline{\mathsf{sup}}$ and $\star$ respectively and to construct $\mathrm{trunc} : \widehat{M} \to \overline{M}^{\mathbb{N}}$. This is because the question of whether $\bot$ occurs at an 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:

$$\begin{array}{rlrl} \overline{\mathrm{trunc}}_0\, x &\equiv \overline{\bot} & \overline{\mathrm{trunc}}_{n+1}\, \overline{\bot} &\equiv \star \\ \overline{\mathrm{trunc}}_{n+1}(\overline{\mathsf{sup}}(a,\, f)) &\equiv \overline{\mathsf{sup}}(a,\, \overline{\mathrm{trunc}}_n \cdot f) & \overline{\mathrm{trunc}}_{n+1}\, \star &\equiv \star. \end{array}$$

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 $u_0 \equiv v_0(a, f) \equiv w_0 \equiv \bot$, $u_{n+1} \equiv w_{n+1} \equiv \star$ and $v_{n+1}(a, f) \equiv \overline{\mathsf{sup}}(a, f_n)$.

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

$$\imath\bot \equiv \overline{\bot} \qquad \imath(\mathsf{sup}(a, f)) \equiv \overline{\mathsf{sup}}(a,\, \imath \cdot f).$$

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

$$\mathrm{trunc}_0\, x = \imath\bot \qquad \mathrm{trunc}_{n+1}(\mathsf{sup}(a, f)) = \overline{\mathsf{sup}}(a,\, \mathrm{trunc}_n \cdot f).$$

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 $\forall n : \mathbb{N}.\, (\imath m_n = \mathrm{trunc}_n\, m_{n+1})$. So define

$$M \equiv \sum m : \widehat{M}^{\mathbb{N}}. \prod n : \mathbb{N}.\, (\imath m_n = \mathrm{trunc}_n\, m_{n+1}), \tag{4}$$

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

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

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 $M \hookrightarrow \widehat{M}^{\mathbb{N}}$:

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

showing that $\imath \cdot \overline{\beta}_n = \mathrm{trunc}_n \cdot \beta_{n+1}$. 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 (4) a term $m : M$ satisfies the equation $\imath m_{n+1} = \mathrm{trunc}_{n+1}\, m_{n+2}$; by disjointness of coproducts and the definition of trunc_{n+1} we can see that this equation must be of the form $\imath m_{n+1} = \overline{\mathsf{sup}}(a, \mathrm{trunc}_n \cdot f_{n+1})) = \mathrm{trunc}_{n+1}(\mathsf{sup}(a, f_{n+1})) = \mathrm{trunc}_{n+1}\, m_{n+2}$ for some a and f_{n+1}. We can therefore write $m_{n+1} = \mathsf{sup}(a, f_n)$ where f_n satisfies the equation $\imath \cdot f_n = \mathrm{trunc}_n \cdot f_{n+1}$. By defining $\alpha' m \equiv (a, f)$ we obtain a morphism $\alpha' : M \to TM$.

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

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

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

5. Inductive and coinductive containers

Throughout this section take $\mathbb{C}$ to be a Martin-Löf category. Here we will show that the interpretation of non-inductive strictly positive types in containers (Corollary 3.10) extends to the full range of strictly positive types (Corollary 5.5). More generally, we will show that if $F(\boldsymbol{X}, Y)$ is a container functor $F : \mathbb{C}^{I+1} \to \mathbb{C}$ then the fixed points $\mu Y.\ F(\boldsymbol{X}, Y)$ and $\nu Y.\ F(\boldsymbol{X}, Y)$ are also container functors $\mathbb{C}^I \to \mathbb{C}$.

Note that throughout this section we treat μ and ν as partial operators on functors, taking an endofunctor F to (the object part of) its initial algebra μF and its final coalgebra νF, where these objects exist—note that these constructions are necessarily functorial. We also indulge in some obvious abuse of notation, constructing for example a functor $\mu F : \mathbb{D} \to \mathbb{C}$ from a functor $F : \mathbb{D} \times \mathbb{C} \to \mathbb{C}$ and using a notation with variables to describe these. It is not until Corollary 5.5 that we link this notation explicitly to the syntax of strictly positive types.

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

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

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

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

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 these families using W-types in Proposition 5.2 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

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

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

$$\begin{aligned} F[G] &= (S \triangleright P, Q)[(A \triangleright B)] \\ &= \left(s : S,\ f : Qs \to A \triangleright Ps + \sum q : Q.\, B(fq)\right). \end{aligned}$$

This can be written more concisely as $(S, A^Q \triangleright P + \sum_Q \varepsilon^* B)$, where $\varepsilon : A^Q \times Q \to A$ is the evaluation map. Observe now that any fixed point $\psi : [\![S \triangleright Q]\!]A \cong A$ induces an isomorphism of positions between $F[G]$ and G, or equivalently an isomorphism $\psi : [\![F[G]]\!]1 \cong [\![G]\!]1$ and it is 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

$$s : S,\ f : Qs \to A \ \vdash\ \varphi_{s,f} : Ps + \sum q : Q.\, B(fq) \cong B(\psi(s, f)). \tag{5}$$

More generally it will be useful to require that (B, φ) form an *initial family over* ψ.

Definition 5.1. An *initial family* over *a fixed point* $\psi : [\![S \triangleright Q]\!]A \cong A$ 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 $\varphi : P + \sum_Q \varepsilon^* B \to \psi^* B$, as in (5) above, which is initial in the category of such families and morphisms. It turns out that such initial families always exist.

Proposition 5.2. *Given a container* $F \equiv (S \triangleright P, Q) \in \mathcal{G}_{I+1}$ *and an object* $A \in \mathbb{C}$ *equipped with a fixed point* $\psi : [\![S \triangleright Q]\!]A \cong A$ *there exists an initial family* $A \vdash \mathrm{Pos}_{P,\psi}$ *over* ψ *for the functor* $X \mapsto P + \sum_Q \varepsilon^* X$.

Proof. Write $S, A^Q \vdash \varphi : P + \sum_Q \varepsilon^* B \to \psi^* B$ for the initial family to be constructed. Note that the functor $B \mapsto P + \sum_Q \varepsilon^* B$ is *not* a container functor, so we cannot directly appeal to W-types to construct this fixed point; thus the first step is to create a fixed point equation that we *can* solve. Begin by "erasing" the type dependency of B and construct (observing that $\sum_Q Y \cong Q \times Y$, etc.)

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

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

The task now is to select the "well-formed" elements of $\widehat{B}$. An element of $\widehat{B}$ can be thought of as a putative path through a tree in $\mu Y.\, [\![F]\!](X, Y)$; we want Ba 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 $s_i : S$, $f_i : Qs_i \to A$, $q_i : Qs_i$ and $p : Ps_n$. The condition that this is a well formed element of $B(\psi(s_0, f_0))$ can be expressed as the n equations

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

showing that B can be captured as a regular subobject of $\widehat{B}$. That this is indeed the required initial family is shown in [1, Proposition 5.5.1]. □

The details of this sketch proof are given in [1], or the result can be derived as a corollary of [18, Theorem 12] by observing that the functor $X \mapsto \psi^{-1*}(P + \sum_Q \varepsilon^* X)$ is a "dependent container functor" (which they call a "dependent polynomial functor") and therefore has an initial algebra.

Being initial, φ is an isomorphism. Writing $G \equiv (A \triangleright \mathrm{Pos}_{P,\psi})$ for the container associated with an initial family, note that $\alpha \equiv (\psi, \varphi^{-1})$ is an isomorphism of containers $\alpha : F[G] \cong G$, and using the decomposition of $[\![F[G]]\!]X$ of (1), see the discussion following Proposition 3.6, we can write the action of $[\![\alpha]\!]_X$ as

$$[\![\alpha]\!]_X(s, f, g, h) = (\psi(s, f), K(g, h)),$$

where $K(g, h) : \mathrm{Pos}(\psi(s, f)) \to X$ can be defined by cases thus:

$$K(g, h)(\varphi(\mathsf{inl}\, p)) \equiv gp \qquad K(g, h)(\varphi(\mathsf{inr}(q, b))) \equiv hqb. \tag{6}$$

Above and in the proofs that follow we use the functional programming convention for brackets that $hqb = (hq)b$. We can now use initial families to construct initial and final containers. First initial algebras of containers.

Proposition 5.3. *Given a container $F \equiv (S \triangleright P, Q) \in \mathcal{G}_{I+1}$ then*

$$[\![\mathsf{W}_S Q \triangleright \mathrm{Pos}_{P,\mathsf{sup}^\mu}]\!]X \cong \mu Y.\ [\![F]\!](X, Y);$$

writing $\mu F \equiv (\mathsf{W}_S Q \triangleright \mathrm{Pos}_{P,\mathsf{sup}^\mu})$ we can conclude that $[\![\mu F]\!] \cong \mu[\![F[-]]\!]$.

Proof. For conciseness write $A \equiv \mathsf{W}_S Q$, $B \equiv \mathrm{Pos}_{P,\mathsf{sup}\,\mu}$ and $G \equiv (A \triangleright B)$ throughout this proof. First recall that $[\![F]\!](X, [\![G]\!]X) = [\![F]\!][[\![G]\!]]X \cong [\![F[G]]\!]X$ and observe that $\alpha \equiv (\mathsf{sup}, \varphi^{-1}) : F[G] \to G$ 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

$$\begin{array}{ccc} [\![F]\!](X, [\![G]\!]X) \cong [\![F[G]]\!]X & \xrightarrow{[\![\alpha]\!]_X} & [\![G]\!]X \\ \Big\downarrow{\scriptstyle [\![F]\!](X, \overline{\beta})} & & \Big\downarrow{\scriptstyle \overline{\beta}} \\ [\![F]\!](X, Y) & \xrightarrow[\beta]{} & Y \end{array} \tag{7}$$

commute. Using Eq. (1) to write the context as $\theta(s, f, g, h) : [\![F[G]]\!]X$ the corresponding equation can be computed as

$$\begin{aligned} &s : S,\ f : Qs \to A,\ g : Ps \to X,\ h : \textstyle\prod q : Qs.\, (B(fq) \to X) \vdash \\ &\quad \overline{\beta}(\mathsf{sup}(s, f), K) = \beta(s,\ g,\ \lambda q.\ \overline{\beta}(fq, hq)), \end{aligned} \tag{8}$$

where $K \equiv K(g,h)$ is defined as in (6)—we will elide the arguments (g,h) which are constant through this proof. 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 the appropriate induction step:

$$\begin{aligned} s : S,\ f : Qs \to A,\ r : \textstyle\prod q : Qs.((B(fq) \to X) \to Y),\ k : B(\mathsf{sup}(s,f)) \to X\ \vdash \\ H(s,f,r)k \equiv \beta(s, T_1(k,-), T_2(k,r,-)), \end{aligned}$$

where $T_1(k,p) \equiv k(\varphi(\mathsf{inl}\, p))$ and $T_2(k,r,q) \equiv rq(\lambda b.k(\varphi(\mathsf{inr}(q,b))))$. In the context of (8) we can compute $T_1(K,p) = gp$ and $T_2(K,r,q) = rq(hq)$. If we now define $\overline{\beta}(a,-) \equiv \mathsf{wrec}_H\, a$ then in this context we can compute

$$\begin{aligned} \overline{\beta}(\mathsf{sup}(s,f),\ K) &= \mathsf{wrec}_H(\mathsf{sup}(s,f))K = H(s,\ f,\ \mathsf{wrec}_H \cdot f)K \\ &= \beta(s,\ T_1(K,-),\ T_2(K,\ \mathsf{wrec}_H \cdot f,-)) \\ &= \beta(s,\ g,\ \lambda q.\,\mathsf{wrec}_H(fq)(hq)) = \beta(s,\ g,\ \lambda q.\overline{\beta}(fq,hq)), \end{aligned}$$

which is precisely Eq. (8), showing that $\overline{\beta}$ is the required initial morphism and that indeed $[\![G]\!]X$ is an initial algebra. □

Where convenient we will write $\mathrm{Pos}_\mu \equiv \mathrm{Pos}_{P,\mathsf{sup}^\mu}$ and $\mathrm{Pos}_\nu \equiv \mathrm{Pos}_{P,\mathsf{sup}^\nu}$. Note that the proof above that $[\![\mu F]\!] \cong \mu[\![F]\!]$ only uses the isomorphism $P + \sum_Q \varepsilon^* \mathrm{Pos}_\mu \cong \mathrm{Pos}_\mu$ and makes no use of initiallity; this may seem surprising, as we might expect the isomorphism problem for Pos_μ to have multiple solutions.

This can be explained intuitively by observing that Pos_μ 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 Pos_μ to be unique. An example of this occurs in Wraith's theorem [25, Theorem 6.19] which treats the special case $\mathsf{W}_S Q = \mathbb{N}$.

The corresponding proof for ν is more intricate because we now have to exploit the initiallity of the family $\mathsf{M}_S Q \ \vdash\ \mathrm{Pos}_{P,\mathsf{sup}^\nu}$.

Proposition 5.4. *Given a container $F \equiv (S \triangleright P, Q) \in \mathcal{G}_{I+1}$ then*

$$\left[\mathsf{M}_S Q \ \triangleright\ \mathrm{Pos}_{P,\mathsf{sup}^\nu}\right] X \cong \nu Y.\ [\![F]\!](X,Y);$$

writing $\nu F \equiv (\mathsf{M}_S Q \ \triangleright\ \mathrm{Pos}_{P,\mathsf{sup}^\nu})$ we have $[\![\nu F]\!] \cong \nu[\![F[-]]\!]$.

Proof. Let $A \equiv \mathsf{M}_S Q$, $B \equiv \mathrm{Pos}_{P,\mathsf{sup}}$ and $G \equiv (A \triangleright B)$ as before and observe that $\alpha : F[G] \to G$ exists as above and has an inverse $\alpha^{-1} = (\mathsf{sup}^{-1}, \varphi)$. We will show that each $[\![\alpha^{-1}]\!]_X$ is a final $[\![F]\!](X,-)$-coalgebra. Let $\beta : Y \to [\![F]\!](X,Y)$ be a coalgebra: we will construct $\overline{\beta} : Y \to [\![G]\!]X$ uniquely satisfying

$$\overline{\beta} = [\![\alpha]\!]_X \cdot [\![F]\!](X,\overline{\beta}) \cdot \beta. \tag{9}$$

Write the coalgebra $\beta : Y \to \sum_S (X^P \times Y^Q)$ as $\beta y = (sy, gy, hy)$ with components

$$s : Y \longrightarrow S \qquad g : \prod y : Y.\,(P(sy) \to X) \qquad f : \prod y : Y.\,(Q(sy) \to Y),$$

and similarly write $\overline{\beta} : Y \to \sum_A X^B$ as $\overline{\beta} y = (ay, ky)$ with components $a : Y \to A$ and $k : \prod y : Y.\,(B(ay) \to X)$. In context $y : Y$ equation (9) can be computed as

$$(ay, ky) = (\mathsf{sup}(sy,\ a \cdot fy),\ K(gy,\ k \cdot fy)), \tag{10}$$

where K is defined in (6). It is immediately evident that a is fully determined by the final coalgebra property of $A = \mathsf{M}_S Q$. To construct k we will need to appeal to the initial family property of B: we will work backwards to discover the correct construction.

First observe that k can be regarded as a morphism $k : a^* B \to X$ in $\mathbb{C}/Y$, and hence can be transposed to a morphism $\overline{k} : B \to \prod_a X$ in $\mathbb{C}/A$—this is in the right form to construct using the initial family property. We can write $\prod_a X$ using equality in context $a' : A$ as $(\sum y : Y.\,ay = a') \to X$ and so we now want to construct

$$a' : A \ \vdash\ \overline{k}_{a'} : Ba' \longrightarrow \big(\big(\textstyle\sum y : Y.\,ay = a'\big) \to X\big),$$

this will arise by initiality of families from a suitable morphism $\overline{K}$

$$\begin{aligned} s' : S,\ f' : Qs' \to A \ \vdash\ & Ps' + \textstyle\sum q : Qs'.\ \big(\big(\sum y : Y.\,ay = f'q\big) \to X\big) \\ \xrightarrow{\overline{K}}\ & \big(\big(\textstyle\sum y : Y.\,ay = \mathsf{sup}(s',\ f')\big) \to X\big). \end{aligned}$$

We can define $\overline{K}x(y, e)$ in context $s' : S,\ f' : Qs' \to A,\ y : Y$ and $e : ay = \mathsf{sup}(s',\ f')$ by the following clauses:

$$\begin{aligned} &p : Ps' \ \vdash\ \overline{K}(\mathsf{inl}\ p)(y, e) \equiv gyp, \\ &q : Qs',\ g' : \big(\textstyle\sum y : Y.\,ay = f'q\big) \to X \ \vdash\ \overline{K}(\mathsf{inr}(q, g'))(y, e) \equiv g'(fyq, \mathsf{refl}_{f'q}), \end{aligned}$$

where well typedness follows by equality reasoning: first e tells us that $\mathsf{sup}(s',\ f') = ay = \mathsf{sup}(sy,\ a \cdot fy)$ and so (as sup is an isomorphism) $s' = sy$ and $f' = a \cdot fy$. The definitions above can now be seen to be well typed by direct computation.

The initial families equation defining $\overline{k}$ now becomes $\overline{k} \cdot \varphi = \overline{K} \cdot (P + \sum_Q \varepsilon^* \overline{k})$, or writing it out more fully:

$$\begin{aligned} \overline{k}_{\mathsf{sup}(s',\,f')}(\varphi(\mathsf{inl}\ p))(y, e) &= \overline{K}(\mathsf{inl}\ p)(y, e) = gyp, \\ \overline{k}_{\mathsf{sup}(s',\,f')}(\varphi(\mathsf{inr}(q, b)))(y, e) &= \overline{K}(\mathsf{inr}(q, \overline{k}_{f'q} b))(y, e) = \overline{k}_{f'q} b(fyq, \mathsf{refl}_{f'q}). \end{aligned}$$

Finally reconstruct k from $\overline{k}$ as $kyb = \overline{k}_{ay} b(y, \mathsf{refl}_{ay})$. These equations then become

$$\begin{aligned} ky(\varphi(\mathsf{inl}\ p)) &= gyp = K(gy,\ k \cdot fy)(\varphi(\mathsf{inl}\ p)), \\ ky(\varphi(\mathsf{inr}(q, b)) &= k(fyq)b = K(gy,\ k \cdot fy)(\varphi(\mathsf{inr}(q, b))) \end{aligned}$$

showing that k is indeed uniquely determined to satisfy Eq. (10), thus establishing that $[\![\alpha^{-1}]\!]_X$ is the desired final coalgebra. □

Note that the construction of νF only uses *initial* families, that is to say, initiality and not finality of Pos_ν is the required defining property. This can be understood by observing that although an element $t : \mathsf{M}_S Q$ may represent an infinite tree, any position in $\mathrm{Pos}_\nu\, t$ represents a *finite* path into t.

Finally observe that μF is the object of an initial algebra for the substitution functor $F[-]:\mathcal{G}_I \to \mathcal{G}_I$ and similarly νF is the object of a final coalgebra: this follows by the reflection of Propositions 5.3 and 5.4 along $[\![-]\!]$. The following corollary summarises the results of this section.

Corollary 5.5. *In a Martin-Löf category every strictly positive type F in n variables can be interpreted as an n-ary container $(\!|F|\!) \in \mathcal{G}_n$ such that $(\!|\mu X_{n+1}.\ F|\!) = \mu(\!|F|\!)$, $(\!|\nu X_{n+1}.\ F|\!) = \nu(\!|F|\!)$, and the interpretations of Corollary* 3.10 *hold.*

6. Conclusions and further work

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

Corollary 6.1. *Each strictly positive type F in n variables can be interpreted as an n-ary container $(\!|F|\!) : \mathcal{G}_n$. Given the interpretation of n-ary strictly positive types $(\!|F|\!) = (A \triangleright B)$, $(\!|G|\!) = (C \triangleright D)$ and $n+1$-ary strictly positive type $(\!|H|\!) = (S \triangleright P, Q)$, we have the following translation*:

$$\begin{aligned}
(\!|K|\!) &= (K \ \triangleright\ j \mapsto 0),\\
(\!|X_i|\!) &= (1 \ \triangleright\ j \mapsto (i = j)),\\
(\!|F + G|\!) &= (A + C \ \triangleright\ j \mapsto B_j \dot{+} D_j),\\
(\!|F \times G|\!) &= ((a, c) : A \times C \ \triangleright\ j \mapsto B_j a + D_j c),\\
(\!|K \to F|\!) &= \left(f : K \to A \ \triangleright\ j \mapsto \textstyle\sum k : K.\, B_j(fk)\right),\\
(\!|\mu X_{n+1}.\ H|\!) &= (\mathsf{W}_S Q \ \triangleright\ j \mapsto \mathrm{Pos}_{P_j,\mathsf{sup}^\mu}),\\
(\!|\nu X_{n+1}.\ H|\!) &= (\mathsf{M}_S Q \ \triangleright\ j \mapsto \mathrm{Pos}_{P_j,\mathsf{sup}^\nu}).
\end{aligned}$$

In the special case $n = 0$ this implies that all closed strictly positive types can be interpreted as objects in any Martin-Löf category.

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 and 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 [30,29] which use schematic inductive definitions. The correctness of the schemes is currently not checked and is a likely cause of unsoundness. Using our construction [4] we can translate the schematic definitions into a fixed core theory whose terms can be easily checked.

[4] The constructions in this paper depend essentially on extensional equality, while the current version of Epigram is intensional. However, McBride and Altenkirch are currently working on an extensional but decidable implementation of Epigram based on [10].

Nested datatypes [11,14] 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, i.e. finite multisets. 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 [5]. There is an interesting interaction with our work on derivatives [3,6], 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. We would also like to thank the referees for their comments which have helped to improve this paper.

References

[1] M. Abbott, Categories of containers, Ph.D. Thesis, University of Leicester, 2003.
[2] M. Abbott, T. Altenkirch, N. Ghani, Categories of containers, in: Proc. Foundations of Software Science and Computation Structures, Lecture Notes in Computer Science, Vol. 2620, Springer, Berlin, 2003, pp. 23–38.
[3] M. Abbott, T. Altenkirch, N. Ghani, C. McBride, Derivatives of containers, in: Sixth International Conference on Typed Lambda Calculi and Applications, Lecture Notes in Computer Science, Vol. 2701, Springer, Berlin, 2003, pp. 16–30.
[4] M. Abbott, T. Altenkirch, N. Ghani, Representing nested inductive types using W-types, in: International Colloquium on Automata, Languages and Programming, ICALP, 2004, pp. 59–71.
[5] M. Abbott, T. Altenkirch, N. Ghani, C. McBride, Constructing polymorphic programs with quotient types, in: Seventh International Conference on Mathematics of Program Construction (MPC 2004), Lecture Notes in Computer Science, Vol. 3125, February 2004, pp. 2–15.
[6] M. Abbott, T. Altenkirch, N. Ghani, C. McBride, ∂ for data, February 2004, submitted for publication.
[7] A. Abel, T. Altenkirch, A predicative strong normalisation proof for a λ-calculus with interleaving inductive types, in: Types for Proof and Programs, TYPES '99, Lecture Notes in Computer Science, Vol. 1956, Springer, Berlin, 2000, pp. 1–18.
[8] P. Aczel, On relating type theories and set theories, Lecture Notes in Comput. Sci. 1657 (1999) 1–18.
[9] T. Altenkirch, Constructions, Inductive types and strong normalization, Ph.D. Thesis, University of Edinburgh, November 1993.
[10] T. Altenkirch, Extensional equality in intensional type theory, in: 14th Symposium on Logic in Computer Science, 1999, pp. 412–420.
[11] T. Altenkirch, B. Reus, Monadic presentations of lambda terms using generalized inductive types, in: J. Flum, M. Rodríguez-Artalejo (Eds.), CSL'99, Lecture Notes in Computer Science, Vol. 1683, Springer, Berlin, 1999, pp. 453–468.
[12] J. Bénabou, Fibrations petites et localement petites, C.R. Acad. Sci. Paris 281 (1975) A831–A834.
[13] J. Bénabou, Fibred categories and the foundations of naive category theory, J. Symbol. Logic 50 (1) (1985) 10–37.
[14] R. Bird, R. Paterson, Generalised folds for nested datatypes, Formal Aspects Comput. 11 (3) (1999) 200–222.

[15] F. Borceux, Handbook of Categorical Algebra 2, Encyclopedia of Mathematics, Vol. 51, Cambridge University Press, Cambridge, 1994.
[16] R.L. Crole, Categories for Types, Cambridge University Press, Cambridge, 1993.
[17] P. Dybjer, Representing inductively defined sets by wellorderings in Martin-Löf's type theory, Theoret. Comput. Sci. 176 (1997) 329–335.
[18] N. Gambino, M. Hyland, Wellfounded trees and dependent polynomial functors, in: S. Berardi, M. Coppo, F. Damiani (Eds.), Types for Proofs and Programs (TYPES 2003), Lecture Notes in Computer Science, Springer, Berlin, 2004, pp. 210–225.
[19] R. Hasegawa, Two applications of analytic functors, Theoret. Comput. Sci. 272 (1–2) (2002) 112–175.
[20] M. Hofmann, On the interpretation of type theory in locally cartesian closed categories, in: Computer Science Logic, CSL94, 1994, pp. 427–441
[21] M. Hofmann, Extensional Constructs in Intensional Type Theory, Springer, Berlin, 1997.
[22] M. Hofmann, Syntax and semantics of dependent types, in: A.M. Pitts, P. Dybjer (Eds.), Semantics and Logics of Computation, Vol. 14, Cambridge University Press, Cambridge, 1997, pp. 79–130.
[23] P. Hoogendijk, O. de Moor, Container types categorically, J. Function. Program. 10 (2) (2000) 191–225.
[24] B. Jacobs, Categorical logic and type theory, Studies in Logic and the Foundations of Mathematics, Vol. 141, Elsevier, Amsterdam, 1999.
[25] P.T. Johnstone, Topos Theory, Academic Press, New York, 1977.
[26] A. Joyal, Foncteurs analytiques et espèces de structures, in: Combinatoire Énumérative, Lecture Notes in Mathematics, Vol. 1234, Springer, Berlin, 1986, pp. 126–159.
[27] P. Martin-Löf, An intuitionistic theory of types: predicative part, in: H.E. Rose, J.C. Shepherdson (Eds.), Proceedings of the Logic Colloquium, North-Holland, Amsterdam, 1974, pp. 73–118.
[28] P. Martin-Löf, Intuitionistic Type Theory, Bibliopolis, Napoli, 1984.
[29] C. McBride, Epigram: practical programming with dependent types, Lecture Notes of the Advanced Functional Programming Summerschool in Tartu, Estonia, 2004.
[30] C. McBride, J. McKinna, The view from the left, J. Function. Program. 14 (1) (2004) 16–111.
[31] I. Moerdijk, E. Palmgren, Wellfounded trees in categories, Ann. Pure Appl. Logic 104 (2000) 189–218.
[32] B. Nordström, K. Petersson, J.M. Smith, Programming in Martin-Löf's Type Theory, International Series of Monographs on Computer Science, Vol. 7, Oxford University Press, Oxford, 1990.
[33] R. Paré, D. Schumacher, Abstract families and the adjoint functor theorems, in: P.T. Johnstone, R. Paré (Eds.), Indexed Categories and Their Applications, Lecture Notes in Mathematics, Vol. 661, Springer, Berlin, 1978, pp. 1–125.
[34] A. Poigné, Basic category theory, in: S. Abramsky, D.M. Gabbay, T.S.E. Maibaum (Eds.), Handbook of Logic in Computer Science, Handbook of Logic in Computer Science, Vol. 1, Oxford University Press, Oxford, 1992. pp. 413–640.
[35] R.A.G. Seely, Locally cartesian closed categories and type theory, Math. Proc. Camb. Phil. Soc. 95 (1984) 33–48.
[36] T. Streicher, Semantics of Type Theory, Progress in Theoretical Computer Science, Birkhäuser, 1991.
[37] D. Turner, Elementary strong functional programming, in: R. Plasmeijer, P. Hartel (Eds.), First International Symposium on Functional Programming Languages in Education, Lecture Notes in Computer Science, Vol. 1022, Springer, Berlin, 1996, pp. 1–13.
[38] B. van den Berg, F. de Marchi, Non-well-founded trees in categories, 2004, arXiv math.CT/0409158.