Representing Nested Inductive Types using W-types

Michael Abbott[1], Thorsten Altenkirch[2], and Neil Ghani[1]

[1] Department of Mathematics and Computer Science, University of Leicester
michael@araneidae.co.uk, ng13@mcs.le.ac.uk
[2] School of Computer Science and Information Technology, Nottingham University
txa@cs.nott.ac.uk

Abstract. We show that strictly positive inductive types, constructed from polynomial functors, constant exponentiation and arbitrarily nested inductive types exist in any Martin-Löf category (extensive locally cartesian closed category with W-types) by exploiting our work on container types. This generalises a result by Dybjer (1997) who showed that non-nested strictly positive inductive types can be represented using W-types. We also provide a detailed analysis of the categorical infrastructure needed to establish the result.

1 Introduction

Inductive types play a central role in programming and constructive reasoning. From an intuitionistic point of view we can understand strictly positive inductive types (SPITs) as well-founded trees, which may be infinitely branching. The language of SPITs is built from polynomial types and exponentials, enriched by a constructor μ for inductive types. In this language we can conveniently construct familiar types such as the natural numbers, $\mathbb{N} \equiv \mu X.\, 1 + X$; binary trees, $\mathrm{BTree} \equiv \mu X.\, 1 + X \times X$; lists parameterised over a type $\mathrm{List}\, A \equiv \mu X.\, 1 + A \times X$; ordinals, $\mathrm{Ord} \equiv \mu X.\, 1 + X + X^{\mathbb{N}}$; and finitely branching trees as the fixpoint of Lists, $\mathrm{FTree} \equiv \mu Y.\, \mathrm{List}\, Y = \mu Y.\, \mu X.\, 1 + X \times Y$. Categorically, μ corresponds to taking the initial algebra of a given functor.

The grammar of SPITs can be easily defined inductively, see definition 6.1. However, we would like to have a simple semantic criterion which guarantees the existence of SPITs. Dybjer (1997) shows that inductive types over strictly positive operators constructed using only polynomials in a single type variable and fixed exponentiation can be constructed in extensional Type Theory using W-types, the type of well-founded trees introduced in Martin-Löf (1984). However, Dybjer (1997) does not consider any nesting of inductive types, e.g. the example FTree is not covered by his definition. Here we present a more general result which shows that nested inductive types can be constructed using only W-types and we analyse the categorical framework in more detail.

An important ingredient in our construction is the insight that SPITs give rise to containers, which we have investigated in Abbott et al. (2003) and which are the topic of Abbott (2003). The basic notion of a *container* is a dependent pair of types $A \vdash B$ creating a functor $T_{A \triangleright B} X \equiv \sum a : A.\, X^{B(a)}$. A morphism of containers $(A \vdash B) \to (C \vdash D)$

is a pair of morphisms $(u : A \to C, f : u^*D \to B)$. With this definition of a category $\mathcal{G}$ of containers we can construct a full and faithful functor $T : \mathcal{G} \to [\mathbb{C}, \mathbb{C}]$.

However, when constructing fixed points it is also necessary to take account of containers with parameters, so we define $T : \mathcal{G}_I \to [\mathbb{C}^I, \mathbb{C}]$ for each parameter index set I. For the purposes of this paper the index set I can be regarded as a finite set, but this makes little practical difference to the development.

It is easy to show that containers are closed under sums and products and constant exponentiation, see Abbott et al. (2003); this is also done in Dybjer (1997) for containers in one variable. W-types are precisely the initial algebras of containers in one variable (theorem 3.6), hence constructing inductive types over a single variable SPITs is straightforward and already covered (in part) by Dybjer's work. However, the general case for nested types corresponds to showing that containers are closed under initial algebras. The problem boils down (proposition 4.1) to solving an equation on families of types up to isomorphism, which is achieved in proposition 5.1.

The work presented here also overcomes a shortcoming of Abbott et al. (2003): there we constructed initial algebras of containers using the assumption that the ambient category is locally finitely presentable. Alas, this assumption rules out many interesting examples of categories, in particular realisability models such as ω-sets. This is fixed here, since we only have to require that the category has all W-types, i.e. initial algebras of container functors, which can be easily established for realisability models. Since dependent types and inductive types are the core of Martin-Löf's Type Theory, we call categories with this structure *Martin-Löf categories*, see definition 3.7.

Dybjer and Setzer (1999, 2001) present general schemes for inductive (and inductive-recursive) definitions but they do not present a reduction to a combinator like W-types. Moreover, they also use universes extensively.

Recently, Gambino and Hyland (2004) have presented another version of our result, corollary 5.2. They use the notion of a *dependent polynomial functor* and present an apparently simpler construction; however, we have not been able to reconstruct their proof.

2 Definitions and Notation

This paper uses the dependent internal language of a locally cartesian closed category $\mathbb{C}$: see Streicher (1991), Hofmann (1994), Jacobs (1999) and Abbott (2003) for details. The key idea is 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* is defined.

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 $a : A \vdash B(a)$ 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 $B(a)$ as an alias for B and $B(fu)$ as an alias for f^*B, and we'll write $a : A \vdash B(a)$ or even just $A \vdash B$ for $B \in \mathbb{C}/A$—variables will 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 Hofmann (1994) and Abbott (2003).

Terms of type $A \vdash B$ correspond to *global elements* of B, which is to say morphisms $t : 1 \to B$ in $\mathbb{C}/A$. In the internal language we write $a{:}A \vdash t(a) : B(a)$ for such a morphism in $\mathbb{C}$. We will write t for $t(a)$ where practical, again omitting a variable when it can be inferred. Given object $A \vdash B$ and $A \vdash C$ we will write $A \vdash f : B \to C$ for a morphism in $\mathbb{C}/A$, and similarly $A \vdash f : B \cong C$ 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 $b : B(a)$ in context $a{:}A$. In the type theory we'll write $\sum_A B \in \mathbb{C}$ as $1 \vdash \sum a{:}A.B(a)$, or more concisely $\vdash \sum_A B$, with elements $\vdash (t,u) : \sum a : A.B(a)$ corresponding to elements $\vdash t : A$ and $\vdash u : B(t)$.

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 $\Gamma, a : A \vdash B(a)$ 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 $\Gamma, a : A, b : B(a) \vdash C(a,b)$ or just $\Gamma, A, B \vdash C$ as a shorthand for $\Gamma, (a,b) : \sum_A B \vdash C(a,b)$.

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 $\Gamma \vdash \prod a : A.B(b)$ for $\Gamma \vdash \prod_A B$ derived from $\Gamma, A \vdash B$. Finally the *equality type* $A, A \vdash \mathrm{Eq}_A$ 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 $\Gamma, A, A \vdash \mathrm{Eq}_A$. Given parallel morphisms u, v into A the equality type has the key property that an element of $\mathrm{Eq}(u,v) = (u,v)^* \mathrm{Eq}_A$ exists precisely when $u = v$ as morphisms of $\mathbb{C}$.

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 $\kappa_i : A_i \to \sum_{i \in I} A_i$ 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 $- \overset{\circ}{+} -$ 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; this satisfies two identities: $\sum_{A+C}(B \overset{\circ}{+} D) \cong \sum_A B + \sum_C D$ and $\pi_{B \overset{\circ}{+} D} = \pi_B + \pi_D$ (modulo the preceding isomorphism).

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 $[\mathbb{C}^I, \mathbb{C}]$.

Basic Properties of Containers

We summarise here the development of containers in Abbott et al. (2003).

Definition 2.1. *Given an index set I define the* category of containers $\mathcal{G}_I$ *as follows:*

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

Note that the alternative of defining an $n+1$-ary container as an indexed family of n-ary containers is equivalent to this definition (Abbott, 2003, proposition 4.1.1).

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

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

A morphism $(u,f):(A \triangleright B) \to (C \triangleright D)$ 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 $(A \triangleright B) \in \mathcal{G}_I$, thought of as a syntactic presentation of a datatype, generates a fibred functor $T_{A\triangleright B}:\mathbb{C}^I \to \mathbb{C}$ which is its semantics.

Definition 2.2. *Define the* container construction functor $T:\mathcal{G}_I \to [\mathbb{C}^I,\mathbb{C}]$ *as follows. Given* $(A \triangleright B) \in \mathcal{G}_I$ *and* $X \in \mathbb{C}^I$ *define*

$$T_{A\triangleright B}X \equiv \sum a\!:\!A.\ \prod_{i\in I} X_i^{B_i(a)} \ ,$$

and for $(u,f):(A \triangleright B) \to (C \triangleright D)$ *define* $T_{u,f}:T_{A\triangleright B} \to T_{C\triangleright D}$ *to be the natural transformation* $T_{u,f}X:T_{A\triangleright B}X \to T_{C\triangleright D}X$ *thus:*

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

The following proposition follows more or less immediately by the construction of T.

Proposition 2.3 (Abbott et al., 2003, proposition 3.3). *For each container* $F \in \mathcal{G}_I$ *and each container morphism* $\alpha:F \to G$ *the functor* T_F *and natural transformation* T_α *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 2.4 (ibid., theorem 3.4). *The functor* $T:\mathcal{G}_I \to [\mathbb{C}^I,\mathbb{C}]$ *is full and faithful.* □

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 $\mathrm{List}X \to \mathrm{List}X$ 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 $\mathcal{G}_I$ inherits products and coproducts from $\mathbb{C}$, and that T preserves them:

Proposition 2.5 (ibid., propositions 4.1, 4.2). *If* $\mathbb{C}$ *has products and coproducts then* $\mathcal{G}_I$ *has products and coproducts preserved by* T. □

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

$$T_F[T_G] \equiv (\mathbb{C}^I \xrightarrow{(\mathrm{id}_{\mathbb{C}^I}, T_G)} \mathbb{C}^I \times \mathbb{C} \cong \mathbb{C}^{I+1} \xrightarrow{T_F} \mathbb{C}) \ .$$

This composition can be lifted 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 $(A \triangleright B, E)\in\mathscr{G}_{I+1}$, where $B\in(\mathbb{C}/A)^I$ and $E\in\mathbb{C}/A$ and define:

$$(A \triangleright B,E)[(C \triangleright D)] \equiv \Big(a\!:\!A, f\!:\!C^{E(a)} \triangleright \big(B_i(a)+\textstyle\sum e\!:\!E(a).\ D_i(fe)\big)_{i\in I}\Big)\ .$$

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 2.6 (ibid., proposition 6.1). *Composition of containers commutes with composition of functors thus: $T_F[T_G]\cong T_{F[G]}$.* □

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).

3 Initial Algebras and W-Types

In this section we discuss the construction of initial algebras for container functors and the principles in the ambient category $\mathbb{C}$ used to construct them.

Initial algebras can be regarded as the fundamental building blocks used to introduce recursive datatypes into type theory. Initial algebras define "well founded" structures, which can be regarded as the expression of terminating processes.

Definition 3.1. *An* algebra *for a functor $F:\mathbb{C}\to\mathbb{C}$ is an object $X\in\mathbb{C}$ together with a morphism $h\!:\!FX\to X$; refer to X as the* carrier *of the algebra. An* algebra morphism *$(X,h)\to(Y,k)$ is a morphism $f\!:\!X\to Y$ satisfying the identity $f\cdot h = k\cdot Ff$. An* initial algebra *for F is then an initial object in the category of algebras and algebra morphisms.*

The following result tells us that initial algebras for a functor F are *fixed points* of F, and indeed the initial algebra is often called the least fixed point.

Proposition 3.2 (Lambek's Lemma). *Initial algebras are isomorphisms.* □

The following useful result about initial algebras tells us that initial algebras with parameters extend to functors, and so can be constructed "pointwise".

Proposition 3.3. *Given a functor $F:\mathbb{D}\times\mathbb{C}\to\mathbb{C}$ if each endofunctor $F(X,-)$ on $\mathbb{C}$ has an initial algebra $(GX,\alpha X)$ then G extends to a functor $G\!:\!\mathbb{D}\to\mathbb{C}$ and α to a natural transformation $\alpha\!:\!F[G]\to G$.* □

We can now define an operation μ constructing the least fixed point of a functors. If we regard a functor $F\!:\!\mathbb{D}\times\mathbb{C}\to\mathbb{C}$ as a type constructor $F(X,Y)$ then we can can regard the fixed points defined below as types.

Definition 3.4. *Given a functor $F : \mathbb{D} \times \mathbb{C} \to \mathbb{C}$ regarded as a type constructor $F(X,Y)$ define $\mu Y.F(X,Y)$ to be the initial algebra of the functor $F(X,-)$.*

To extend this definition of μ types to containers observe that for containers $F \in \mathscr{G}_{I+1}$ and $G \in \mathscr{G}_I$ the operation $G \mapsto F[G]$, with $T_{F[G]}X \cong T_F(X, T_G X)$ is an endofunctor on $\mathscr{G}_I$. Thus given $F \in \mathscr{G}_{I+1}$ we will write μF for the initial algebra of $F[-] : \mathscr{G}_I \to \mathscr{G}_I$.

We will show in this paper that the functor $\mu : \mathscr{G}_{I+1} \to \mathscr{G}_I$ exists, and that the initial algebra of a container functor is a container functor.

W-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 $\mathrm{W}a\!:\!A.B(a)$ (or $\mathrm{W}_A B$) should be regarded as the type of "well founded trees" constructed by regarding each $a\!:\!A$ as a constructor of arity $B(a)$.

The standard presentation of a W-type is through one type forming rule, an introduction rule and an elimination rule, together with an equation. As the type theoretic development in this paper focuses entirely on categorical models, we take W types to be *extensionally* defined. Indeed, extensional Type Theory as presented in Martin-Löf (1984) represents the canonical example of a Martin-Löf category.

Definition 3.5. *A type system* has W-types *iff it has a type constructor*

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

together with a constructor term

$$\Gamma,\; a\!:\!A,\; f : (\mathrm{W}_A B)^{B(a)} \;\vdash\; \mathrm{sup}(a,b) : \mathrm{W}_A B \qquad \text{(sup)}$$

and an elimination rule

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

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

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

Note that the elimination rule together with equality types ensures that wrec_h is unique.

Theorem 3.6. W*-types are precisely the initial algebras of container functors in one parameter:*

$$\mathrm{W}_A B \cong \mu X.\, \textstyle\sum_A X^B = \mu X.\, T_{A \triangleright B} X \quad . \qquad \square$$

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

Definition 3.7. *A* Martin-Löf category *is an extensive locally cartesian closed category with an initial algebra for every container functor (i.e. W-types).*

We can either assume that $\mathbb{C}$ has W-types given axiomatically or, if $\mathbb{C}$ satisfies the necessary preconditions, derive them from theorem 3.9 below. Alternatively if $\mathbb{C}$ is a topos we can appeal to proposition 3.6 of Moerdijk and Palmgren (2000).

Proposition 3.8 (Moerdijk and Palmgren, 2000, proposition 3.6). *W-types exist in any elementary topos with a natural numbers object.* □

Yet another alternative is to use:

Theorem 3.9 (Abbott et al., 2003, theorem 6.8). *If* $\mathbb{C}$ *is locally cartesian closed and locally presentable then* $\mathbb{C}$ *has all* W*-types.* □

4 Initial Algebras of Containers

One consequence of theorem 3.6 is that in the presence of W-types we can immediately construct μ types for containers in one parameter. However, the construction of a μ type for a container in multiple parameters is a more delicate matter and will require the introduction of some additional definitions.

Let $F : \mathbb{C}^{I+1} \to \mathbb{C}$ be a container in multiple parameters, which we can write as

$$F(X,Y) \equiv T_{S \rhd P,Q}(X,Y) = \sum s : S.\ \left(\prod_{i \in I} X_i^{P_i(s)}\right) \times Y^{Q(s)} = \sum_S \left(\prod_I X^P \times Y^Q\right) \ .$$

The task is to compute $(A \rhd B)$ such that $T_{A \rhd B} X \cong \mu Y.F(X,Y)$. Clearly

$$A \cong T_{A \rhd B} 1 \cong \mu Y.\ F(1,Y) \cong \mu Y.\ \sum s : S.\ Y^{Q(s)} \cong \mathrm{W}_S Q \ ,$$

but the construction of $\mathrm{W}_S Q \vdash B$ is more tricky.

In the rest of this paper we will ignore the index set I and write X^P for $\prod_I X^P$. In particular, this means that the family $B \in (\mathbb{C}/\mathrm{W}_S Q)^I$ will be treated uniformly (as if $I = 1$). The required extra working to take account of I can be routinely added, but will further complicate a presentation which is quite complex enough already. We will therefore take

$$F(X,Y) \equiv \sum_S (X^P \times Y^Q) \ .$$

To simplify the algebra of types we will write $S, A^Q \vdash P + \sum_Q \varepsilon^* B$ as an abbreviation for the type expression (where ε is the evaluation map $A^Q \times Q \to A$):

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

For conciseness write the initial algebra on $A = \mathrm{W}_S Q$ as $\psi : \sum_S A^Q \to A$.

Proposition 4.1. *Given the notation above, if* $\mathrm{W}_S Q \vdash B$ *is equipped with an fibred family of isomorphisms:*

$$S, A^Q\ \vdash\ \varphi : P + \sum_Q \varepsilon^* B \cong \psi^* B$$

then $T_{A \rhd B} X \cong \mu Y.F(X,Y)$.

Proof. First we show that each $T_{A \rhd B}X$ is an $F(X,-)$ algebra thus:

$$\begin{aligned} F(X,T_{A \rhd B}X) &= \textstyle\sum_S \left(X^P \times (\sum_A X^B)^Q\right) \cong \sum_S \left(X^P \times \sum_{AQ} \prod_Q X^{\varepsilon^* B}\right) \\ &\cong \textstyle\sum_S \sum_{AQ} \left(X^P \times \prod_Q X^{\varepsilon^* B}\right) \cong \sum_S \sum_{AQ} X^{P + \sum_Q \varepsilon^* B} \\ &\overset{\varphi^{-1}}{\cong} \textstyle\sum_S \sum_{AQ} X^{\psi^* B} \overset{(\psi,\mathrm{id})}{\cong} \sum_A X^B = T_{A \rhd B}X \ . \end{aligned}$$

With variables $s:S$, $g:X^{P(s)}$ and $h:\left(\sum_A X^B\right)^{Q(s)}$ note that we can decompose h into components $\pi \cdot h : A^{Q(s)}$ and $\pi' \cdot h : \prod q : Q(s) . X^{B(\pi h q)}$ and so the algebra morphism $in : F(X, T_{A \rhd B}X) \to T_{A \rhd B}X$ can be conveniently written as

$$in(s,g,h) = (\psi(s, \pi \cdot h), [g; \pi' \cdot h] \cdot \varphi^{-1}) \ ;$$

conversely, given variables $s:S$, $f:A^{Q(s)}$ and $k:X^{B(\psi(s,f))}$ similarly note that $k \cdot \varphi \cdot \kappa'$ can be regarded as a term of type $\prod q : Q(s) . X^{B(fq)}$ and so we can write

$$in^{-1}(\psi(s,f),k) = (s,\ k \cdot \varphi \cdot \kappa,\ (f, k \cdot \varphi \cdot \kappa')) \ .$$

To show that in is an *initial* $F(X,-)$-algebra we need to construct from any algebra $\alpha : F(X,Y) \to Y$ a unique map $\overline{\alpha} : T_{A \rhd B}X \to Y$ satisfying the algebra morphism equation $\overline{\alpha} \cdot in = \alpha \cdot F(X, \overline{\alpha})$:

$$\begin{tikzcd} F(X, T_{A \rhd B}X) \arrow[r, "in"] \arrow[d, dashed, "F(X,\overline{\alpha})"'] & T_{A \rhd B}X \arrow[d, dashed, "\overline{\alpha}"] \\ F(X,Y) \arrow[r, "\alpha"'] & Y \end{tikzcd} \ .$$

The map $\overline{\alpha}$ can be transposed to a term $A \vdash \widetilde{\alpha} : X^B \Rightarrow Y$ which we will construct by induction on $A = \mathrm{W}_S Q$. Given $s:S$, $f:A^{Q(s)}$ and $k:X^{B(\psi(s,f))}$ construct $g \equiv k \cdot \varphi \cdot \kappa : X^{P(s)}$ and $h \equiv k \cdot \varphi \cdot \kappa' : \prod q : Q(s) . X^{B(fq)}$. In this context define $H(s,f,\beta)(k) \equiv \alpha(s,g,\beta(h))$ and compute

$$\begin{aligned} \widetilde{\alpha}(\psi(s,f))(k) &= \overline{\alpha}(\psi(s,f),k) = \overline{\alpha} \cdot in \cdot (s,g,(f,h)) \\ &= \alpha \cdot F(X,\overline{\alpha}) \cdot (s,g,(f,h)) = \alpha(s,g,\overline{\alpha} \cdot (f,h)) \\ &= \alpha(s,g,(\widetilde{\alpha} \cdot f)(h)) = H(s,f,\widetilde{\alpha} \cdot f)(k) \ . \end{aligned}$$

This shows that $\widetilde{\alpha} = \mathrm{wrec}_H$ and thus that $T_{A \rhd B}X$ is an $F(X,-)$-initial algebra. □

Note that as a corollary of this proposition the isomorphism $P + \sum_Q \varepsilon^* B \cong \psi^* B$ over $\mathrm{W}_S Q$ defines B up to isomorphism, since the container $T_{A \rhd B}$ is determined up to isomorphism as an initial algebra.

Of course, it remains to prove the hypothesis of the theorem above, that a family $A \vdash B$ with the given isomorphism φ exists; we do this in proposition 5.1.

5 Constructing a Fixed Point over an Initial Algebra

Proposition 4.1 relies on the hypothesis that the functor $X \mapsto P + \sum_Q \varepsilon^* X$ has a fixed point "over" the initial algebra $\psi : T_{S \rhd Q} A \to A$, or in other words there exists a B such that $P + \sum_Q \varepsilon^* B \cong \psi^* B$. This fixed point does indeed exist, as a *subtype* of a W-type.

Proposition 5.1. *For each fixed point $\psi : T_{S \rhd Q} A \cong A$ there exists an object $A \vdash B$ such that there is an isomorphism:*

$$S, A^Q \;\vdash\; P + \sum\nolimits_Q \varepsilon^* B \cong \psi^* B \ .$$

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)

$$\begin{aligned} \widehat{B} &\equiv \mu Y.\, \sum\nolimits_S \sum\nolimits_{A^Q} (P + Q \times Y) \cong \mu Y.\, \Big(\sum\nolimits_S (A^Q \times P) + \big(\sum\nolimits_S (A^Q \times Q)\big) \times Y\Big) \\ &\cong \mathrm{List}\Big(\sum\nolimits_S (A^Q \times Q)\Big) \times \sum\nolimits_S (A^Q \times P) \ ; \end{aligned}$$

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 $\mu Y.\, T_{S \rhd P,Q}(X,Y)$; 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

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

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 can be expressed inductively as follows:

$$\top \implies (\mathsf{nil},(s,f,p)):B(\psi(s,f)) \tag{1}$$
$$fq=\varpi(l,p')\wedge(l,p'):B(fq) \implies (\mathsf{cons}((s,f,q),l),p'):B(\psi(s,f))\ . \tag{2}$$

The converse to (2) 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:

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

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 (1) 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 (2) 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:

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

To show that φ is an isomorphism we need to show that $\widehat{\varphi}^{-1}$ restricts to an inverse to φ. As before we can analyse $b:B(\psi(s,f))$ into two cases, and show that in both cases $\widehat{\varphi}^{-1}b:P(s)+\sum q:Q(s).B(fq)$.

When $b=(\mathsf{nil},(s,f,p))$ then $\widehat{\varphi}^{-1}b=(s,f,\kappa p)$ which can be regarded as an element of $P(s)$. When $b=(\mathsf{cons}((s,f,q),l),p')$ and so $\widehat{\varphi}^{-1}b=(s,f,\kappa'(q,(l,p')))$ it is enough to observe that $b:B(\psi(s,f))$ implies $(l,p'):B(fq)$ and hence $\widehat{\varphi}^{-1}b$ arises from an element of $\sum q:Q(s).B(fq)$. □

We conclude our development with the following summary result as a corollary.

Corollary 5.2. *If $\mathbb{C}$ has W-types then containers are closed under the construction of μ-types.*

Note that that since μF is a fixed point, it satisfies the isomorphism $\mu F \cong F[\mu F]$.

6 Strictly Positive Inductive Types

We now have enough machinery in place to observe that all strictly positive types can be described as containers.

Definition 6.1. *A* strictly positive inductive type (SPIT) in n variables *(Abel and Altenkirch, 2000) is a type expression (with type variables $X_1, \dots, X_n$) built up inductively according to the following rules:*

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

Note that the type expression for a SPIT F can be interpreted as a functor $F : \mathbb{C}^n \to \mathbb{C}$, and indeed we can see that each strictly positive type corresponds to a container in $\mathcal{G}_n$.

Let strictly positive types F, G be represented by containers $(A \triangleright B)$ and $(C \triangleright D)$ respectively, then the table below shows the correspondence between strictly positive types and containers.

$$\begin{aligned} K &\mapsto (K \triangleright 0) & X_i &\mapsto (1 \triangleright (\delta_{i,j})_{j \in I}) \\ F + G &\mapsto (A + C \triangleright B \overset{\circ}{+} D) & F \times G &\mapsto (a : A,\, c : C \triangleright B(a) \times D(c)) \\ K \Rightarrow F &\mapsto \big(f : A^K \triangleright \textstyle\sum k : K.\, B(fk)\big) \end{aligned}$$

As we have seen in this paper the construction of fixed points can be described in a uniform way. Let F be represented by $(S \triangleright P, Q) \in \mathcal{G}_{I+1}$, then for each fixed point $\psi : T_{S \triangleright Q} A \cong A$ of $T_{S \triangleright Q}$ we have constructed in proposition 5.1 an isomorphism over ψ, written here as $A \vdash B_A$, of the form

$$s : S,\, f : A^{Q(s)} \;\vdash\; \varphi : P(s) + \textstyle\sum q : Q(s).\, B_A(fs) \longrightarrow B_A(\psi(s, f)) \;\; ;$$

we can now define

$$\mu Y.\, F \mapsto (\mathrm{W}_S Q \triangleright B_{\mathrm{W}_S Q}) \;\; .$$

Our development can be summarised by the following:

Theorem 6.2. *All strictly positive inductive types can be represented within a Martin-Löf category.*

Proof. This is a consequence of corollary 5.2 and the discussion above.

7 Discussion and further work

An important extension of the work presented here is to include coinductive types, $\nu X.F$, corresponding to terminal coalgebras, to cover non-well founded data structures such as streams (Stream $A = \nu X.A \times X$), which are used extensively in lazy functional programming. We have also established (see Abbott, 2003, p. 78 and Abbott et al., 2004), that Martin-Löf categories are closed under ν-types—this can be reduced to constructing the dual of W-types which we dub M-types.

Another interesting extension would be to consider inductive and coinductively defined families (such as vectors or simply typed λ-terms). Again, we conjecture that it should be possible to represent those within Martin-Löf categories. This result would provide further evidence establishing that these categories provide a convenient and concise base for intuitionistic Type Theory.

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*, 2003.

M. Abbott, T. Altenkirch, and N. Ghani. Representing strictly positive types. Draft, 2004.

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.

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

P. Dybjer and A. Setzer. A finite axiomatization of inductive-recursive definitions. In *Typed Lambda Calculus and Applications*, pages 129–146, 1999.

P. Dybjer and A. Setzer. Indexed induction-recursion. *Lecture Notes in Computer Science*, 2183, 2001.

N. Gambino and M. Hyland. Wellfounded trees and dependent polynomial functors. available from `http://www.dpmms.cam.ac.uk/~ng266/papers.html`, February 2004.

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

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

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

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.

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