NON-WELLFOUNDED TREES IN HOMOTOPY TYPE THEORY

BENEDIKT AHRENS, PAOLO CAPRIOTTI, AND RÉGIS SPADOTTI

Abstract. We prove a conjecture about the constructibility of *coinductive types*—in the principled form of *indexed* M*-types*—in Homotopy Type Theory. The conjecture says that in the presence of *inductive* types, coinductive types are derivable. Indeed, in this work, we construct coinductive types in a subsystem of Homotopy Type Theory; this subsystem is given by Intensional Martin-Löf type theory with natural numbers and Voevodsky's Univalence Axiom. Our results are mechanized in the computer proof assistant Agda.

1. Introduction

Coinductive data types are used in functional programming to represent infinite data structures. Examples include the ubiquitous data type of streams over a given base type, but also more sophisticated types; as an example we present an alternative definition of equivalence of types (Example 22).

From a categorical perspective, coinductive types are characterized by a *universal property*, which specifies the object with that property *uniquely* in a suitable sense. More precisely, a coinductive type is specified as the *terminal coalgebra* of a suitable endofunctor. In this category-theoretic viewpoint, coinductive types are dual to *inductive* types, which are defined as initial algebras.

Inductive, resp. coinductive, types are usually considered in the principled form of the family of W-types, resp. M-types, parametrized by a type A and a dependent type family B over A, that is, a family of types $(B(a))_{a:A}$. Intuitively, the elements of the coinductive type $\mathsf{M}(A, B)$ are trees with nodes labeled by elements of A such that a node labeled by $a : A$ has $B(a)$-many subtrees, given by a map $B(a) \to \mathsf{M}(A, B)$; see Figure 1 for an example. The *inductive* type $\mathsf{W}(A, B)$ contains only *trees of finite depth*, that is, trees where any path within that tree eventually leads to a node $a : A$ such that $B(a)$ is empty.

In this work, we study coinductive types in Homotopy Type Theory (HoTT). HoTT is an extension of intensional Martin-Löf type theory [10]; we give a brief overview in Section 2.

The universal properties defining inductive and coinductive types, respectively, can be expressed internally to intensional Martin-Löf type theory (and thus internally to HoTT). Awodey, Gambino, and Sojakova [6] use this facility when proving, within a subtheory $\mathcal{H}$ of HoTT, a logical equivalence between

(i) the existence of W-types (a.k.a. the existence of a universal object) and
(ii) the addition of a set of type-theoretic rules to their "base theory" $\mathcal{H}$.

The work of Benedikt Ahrens was partially supported by the CIMI (Centre International de Mathématiques et d'Informatique) Excellence program ANR-11-LABX-0040-CIMI within the program ANR-11-IDEX-0002-02 during a postdoctoral fellowship.

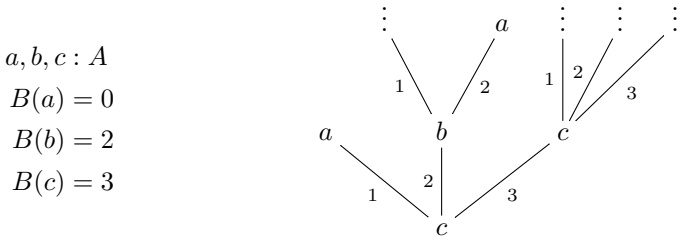

FIGURE 1. Example of a tree (adapted from [7])

We might call the W-types defined internally "internal W-types", and those specified via type-theoretic rules "external" ones. In that sense, Awodey, Gambino, and Sojakova [6] prove a logical equivalence between the existence of internal and external W-types.

The universal property defining (internal) coinductive types in HoTT is dual to the one defining (internal) inductive types. One might hence assume that their existence is equivalent to a set of type-theoretic rules dual (in a suitable sense) to those given for external W-types as in item (ii) above. However, the rules for external W-types cannot be dualized in a naïve way, due to some asymmetry of HoTT related to dependent types as maps into a "type of types" (a *universe*), see the discussion in [1].

In this work, we show instead that coinductive types in the form of M-types can be derived from certain inductive types. (More precisely, only one specific W-type is needed: the type of natural numbers, which is readily specified as a W-type [6].)

The result presented in this work is not surprising; indeed, the constructibility of coinductive types from inductive types has been shown in extensional type theory (see Section 1.1) and was conjectured to work in HoTT during a discussion on the HoTT mailing list [1]. In this work, we give a formal proof of the constructibility of a class of coinductive types from inductive types, with a proof of correctness of the construction.

The theorem we prove here is actually more general than described above: instead of plain M-types as described above, we construct *indexed* M-types, which can be considered as a form of "(simply-)typed" trees, typed over a type of indices I. Plain M-types then correspond to the mono-typed indexed M-types, that is, to those for which $I = 1$. Since all the ideas are already contained in the case of plain M-types, we describe the construction of those extensively, and only briefly state the definitions and the main result for the indexed case. The formalisation in AGDA, however, is done for the more general, indexed, case. An example illustrates the need for these more general *indexed* M-types.

1.1. **Related work.** Inductive types in the form of W-types in HoTT have been studied by Awodey, Gambino, and Sojakova [6]. The content of that work is described above.

Berg and Marchi [7] study the existence of *plain* M-types in models of *extensional* type theory, that is, of type theory with a reflection rule identifying propositional and judgmental equality. They prove the derivability of M-types from W-types in such models, see Corollary 2.5 of the arXiv version of that article. A construction

in extensional type theory of M-types from W-types is given by Abbott, Altenkirch, and Ghani [2].

Martin-Löf type theory without identity reflection, but with the principle of *Uniqueness of Identity Proofs* (Axiom K) can be identified with the 0-truncated fragment of HoTT (modulo the assumption of univalence and HITs). For such a type theory, a construction of (indexed) M-types from W-types is described by Altenkirch et al. [4]. Our work thus generalizes the construction described in [4] by extending it from the 0-truncated fragment to the whole of HoTT.

1.2. **Synopsis.** The paper is organized as follows: In Section 2 we present the type theory we are working in—a "subsystem" of HoTT as presented in [12]. In Section 3 we define signatures for *plain* M-types and, via a universal property, the M-type associated to a given signature. In Section 4 we construct the M-type of a given signature. In Section 5 we state the main result for the case of *indexed* M-types. Finally, in Section 6 we give an overview of the formalisation of our result in the proof assistant Agda.

Acknowledgments. We are grateful to many people for helpful discussions about coinductive types, online as well as offline: Thorsten Altenkirch, Steve Awodey, Martin Escardo, Peter LeFanu Lumsdaine, Ralph Matthes, Paige North, Mike Shulman, and Vladimir Voevodsky. We thank Peter LeFanu Lumsdaine and Paige North for suggesting improvements and clarifications for this paper.

2. The type theory under consideration

The present work takes place within a type theory that is a subsystem of the type theory presented in the HoTT book [12]. The latter is often referred to as Homotopy Type Theory (HoTT); it is an extension of intensional Martin-Löf type theory (IMLTT) [10]. The extension is given by two data: firstly, the *Univalence Axiom*, introduced by Vladimir Voevodsky and proven consistent with IMLTT in the simplicial set model [9]. The second extension is given by *Higher Inductive Types* (HITs), the precise theory of which is still subject to active research. Preliminary results on HITs have been worked out by Sojakova [11] and Lumsdaine and Shulman—see [12, Chap. 6] for an introduction. In the present work, we use the Univalence Axiom, but do not make use of HITs.

The syntax of HoTT is extensively described in a book [12]; we only give a brief summary of the type constructors used in the present work, thus fixing notation. The fundamental objects are *types*, which have *elements* ("inhabitants"), written $a : A$. Types can be dependent on terms, which we write as $x : A \vdash B(x) : \mathcal{U}$. In the preceding judgment, we use a special type $\mathcal{U}$, the "universe" or "type of types". In this work we assume any type being an element of $\mathcal{U}$ in the sense of "typical ambiguity" [12, Chap. 1.3], without worrying about universe levels. The formalisation in Agda ensures that everything works fine in that respect: as we will see later, the universe $\mathcal{U}$ is closed under the construction of M-types.

We use the following type constructors: dependent products $\prod_{(x:A)} B(x)$, with non-dependent variant written $A \to B$, dependent sums $\sum_{(x:A)} B(x)$ with non-dependent variant written $A \times B$, the identity type $x =_A y$ and the coproduct type $A + B$. In particular, we assume the empty type 0 and the singleton type 1. Furthermore, we assume the existence of a type of natural numbers, given as an

inductive type according to the rules given in [12, Chap. 1.9]. Finally, we assume the univalence axiom for the universe $\mathcal{U}$ as presented in [12, Chap. 2.10].

Concerning terms, function application is denoted by parentheses as in $f(x)$ or, occasionally, simply by juxtaposition. We write dependent pairs as (a, b) for $b : B(a)$. Projections are indicated by a subscript, that is, for $x : \sum_{(a:A)} B(a)$ we have $x_0 : A$ and $x_1 : B(x_0)$. Indices are also used occasionally to specify earlier arguments of a function of several arguments; e.g., we write $B_i(a)$ instead of $B(i)(a)$.

3. Definition of M-types via universal property

Coinductive types represent *potentially infinite* data structures, such as streams or infinite lists. As such, they have to be contrasted to *inductive* datatypes, which represent structures that are *necessarily finite*, such as unary natural numbers or finite lists.

3.1. Signatures, a.k.a. containers.

In order to analyze inductive and coinductive types systematically, one usually fixes a notion of "signature": a signature specifies, in an abstract way, the rules according to which the instances of a data structure are built. In the following, we consider signatures to be given by "containers" [4]:

Definition 1. A **container** (or **signature**) is a pair (A, B) of a type A and a dependent type $x : A \vdash B(x) : \mathcal{U}$ over A.

The container (A, B) then determines a type of "trees" built as follows: such a tree consists of a *root node*, labeled by an element $a : A$, and a family of trees—"subtrees" of the original tree—indexed by the type $B(a)$. A tree is *well-founded* if it does not have an infinite chain of subtrees.

To the container (A, B) one associates two types of trees built according to those rules: the type $\mathsf{W}(A, B)$ of well-founded trees, and the type $\mathsf{M}(A, B)$ of all trees, i.e., not necessarily well-founded.

The description of the inhabitants of $\mathsf{W}(A, B)$ and $\mathsf{M}(A, B)$ in terms of trees gives a suitable intuition; formally, those types are defined in terms of a *universal property*. Indeed, $\mathsf{M}(A, B)$ will be defined as (the carrier of) a terminal object in a suitable sense.

3.2. Coalgebras for a signature.

Any container (A, B) specifies an endomorphism on types as follows:

Definition 2. Given a container (A, B), define the **polynomial functor** $P : \mathcal{U} \to \mathcal{U}$ associated to (A, B) as

$$P(X) := P_{A,B}(X) := \sum_{a:A} (B(a) \to X) \ .$$

Given a map $f : X \to Y$, define $Pf : PX \to PY$ as the map

$$Pf(a, g) := (a, f \circ g) \ .$$

Note that Definition 2 does not really define a functor, and, more fundamentally, the universe $\mathcal{U}$ is not a (pre-)category in the sense of [3]. Instead, the appropriate notion for P would be an ∞-(endo)functor on the $(\infty, 1)$-category $\mathcal{U}$ [8]. However, we do not attempt to make any of these notions precise, and do not make use of any "functorial" properties of the defined maps. Our use of the word "functor" merely indicates an analogy to the 1-categorical case.

To any signature $S = (A, B)$ we associate a type of *coalgebras* Coalg_S, and a family of types of morphisms between them:

Definition 3. Given a signature $S = (A, B)$ as in definition 2, an **S-coalgebra** is defined to be a pair (C, γ) consisting of a type $C : \mathcal{U}$ and a map $\gamma : C \to P_S C$. A map of coalgebras from (C, γ) to (D, δ) is defined to be a pair (f, p) of a map $f : C \to D$ and a path $p : \delta \circ f = P_S f \circ \gamma$. Put differently, we set

$$\mathsf{Coalg}_S := \sum_{C:\mathcal{U}} C \to PC$$

and

$$\mathsf{Coalg}_S\Big((C, \gamma), (D, \delta)\Big) := \sum_{f:C\to D} \delta \circ f = Pf \circ \gamma \ .$$

There is an obvious composition of coalgebra morphisms, and the identity map $C \to C$ is the carrier of a coalgebra endomorphism on $(\mathcal{C}, \gamma)$. We also write $(C, \gamma) \Rightarrow (D, \delta)$ for the type of coalgebra morphisms from (C, γ) to (D, δ).

3.3. **What is an M-type?** In this section we define (internal) M-types in HoTT via a universal property.

Definition 4. Given a container (A, B), **the (internal) M-type $\mathsf{M}_{A,B}$ associated to** (A, B) is defined to be the pair $(M, \mathsf{out} : M \to P_{A,B} M)$ with the following universal property: for any coalgebra $(C, \gamma) : \mathsf{Coalg}_{(A,B)}$ of (A, B), the type of coalgebra morphisms $(C, \gamma) \Rightarrow (M, \mathsf{out})$ from (C, γ) to (M, out) is contractible.

The use of the definite article in Definition 4 is justified by the following lemma:

Lemma 5. *The type*

$$\mathsf{Final}_S := \sum_{((X,\rho):\mathsf{Coalg}_S)} \ \prod_{((C,\gamma):\mathsf{Coalg}_S)} \mathsf{isContr}\big((C, \gamma) \Rightarrow (X, \rho)\big)$$

is a proposition.

That is, any inhabitant of Final_S is necessarily unique up to propositional equality. We refer to this inhabitant as the "final coalgebra". In Section 4 we construct the final coalgebra.

In the introduction, we use the adjectives "internal" and "external" to distinguish between types specified via universal properties and type-theoretic rules, respectively. Since we do not consider rules for M-types (that is, external M-types) in this work, we drop the adjective "internal" in what follows.

In the following example, we anticipate the result of the next section, namely the existence of a final coalgebra for any signature (A, B):

Example 6. The coinductive type $\mathsf{Stream}(A_0)$ of streams over a base type A_0 is given by $\mathsf{M}(A, B)$ with $A = A_0$ and $B(a) := 1$ for any $a : A_0$. The corresponding polynomial functor P satisfies $P(X) = A_0 \times X$.

Using finality of $\mathsf{M}(A, B)$ we can define maps into streams and prove that they have the expected computational behaviour. For example, the zip function

$$\mathsf{zip} : \mathsf{Stream}(A) \times \mathsf{Stream}(B) \to \mathsf{Stream}(A \times B)$$

can be obtained from the universal property applied to the coalgebra

$$\theta : \mathsf{Stream}(A) \times \mathsf{Stream}(B) \to (A \times B) \times (\mathsf{Stream}(A) \times \mathsf{Stream}(B))$$
$$\theta(xs, ys) := ((\mathsf{head}(xs), \mathsf{head}(ys)), (\mathsf{tail}(xs), \mathsf{tail}(ys))$$

where $\mathsf{head} : \mathsf{Stream}(X) \to X$ and $\mathsf{tail} : \mathsf{Stream}(X) \to \mathsf{Stream}(X)$ are the two components of the final coalgebra out. The computational behaviour of zip is expressed by the fact that zip is a coalgebra morphism

$$\mathsf{zip}(xs, ys) = \mathsf{cons}((\mathsf{head}(xs), \mathsf{head}(ys)), (\mathsf{zip}(\mathsf{tail}(xs), \mathsf{tail}(ys)))),$$

where $\mathsf{cons} = \mathsf{out}^{-1}$.

4. Derivability of M-types

In Section 3 we defined the type Final_S of final coalgebras of a signature S, and showed that this type is a proposition. In this section, we construct an element of Final_S, thus proving the following theorem:

Theorem 7. *The type*

$$\mathsf{Final}_S = \sum_{((X,\rho))} \prod_{((C,\gamma))} \mathsf{isContr}\big((C,\gamma) \Rightarrow (X,\rho)\big)$$

is contractible.

The construction of the final coalgebra is done in several steps, inspired by a construction of M-types from W-types in a type theory satisfying Axiom K by Altenkirch et al. [4]. Its carrier is defined as the limit of a *chain*:

Definition 8. A **chain** is a pair (X, π) of a family of types $X : \mathbb{N} \to \mathcal{U}$ and a family of functions $\pi_n : X_{n+1} \to X_n$. Here and below we write $X_n := X(n)$ for the nth component of the family X.

The (homotopy) limit of such a chain is given by the type of "compatible tuples":

Definition 9. The **limit** of the chain (X, π) is given by the type

$$L := \sum_{(x:\prod_{(n:\mathbb{N})} X_n)} \prod_{(n:\mathbb{N})} \pi_n x_{n+1} = x_n \quad .$$

The limit is equipped with projections $p_n : L \to X_n$, and $\beta_n : \pi_n \circ p_{n+1} = p_n$. Sometimes, we simply write

$$L = \lim X,$$

when the maps π are clear.

Note that this limit (we drop the adjective "homotopy") is an instance of the general construction of homotopy limits by Avigad, Kapulkin, and Lumsdaine [5].

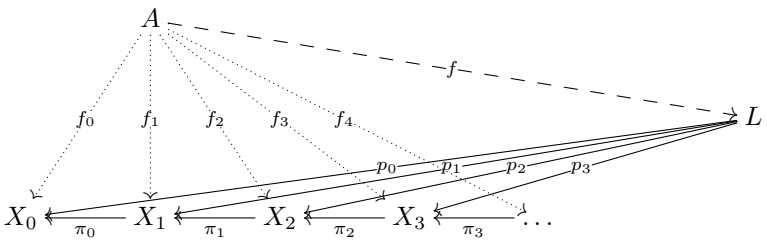

Figure 2. Universal property of L

Lemma 10. *The type L satisfies the following universal property: for all types A, we have an equivalence of types between maps into L and "cones" over X:*

$$A \to L \;\simeq \sum_{(f:\prod_{(n:\mathbb{N})} A \to X_n)} \prod_{(n:\mathbb{N})} \pi_n \circ f_{n+1} = f_n \;=: \mathsf{Cone}(A) \;.$$

The equivalence, from left to right, maps a function $f : A \to L$ to its projections $p_n \circ f$, as shown in Figure 2.

The next lemma is about tuples in **co**chains, that is, tuples in chains with inverted arrows. Those tuples are determined by their first element:

Lemma 11. *Let $X : \mathbb{N} \to \mathcal{U}$ be a family of types, and $l : \prod_{(n:\mathbb{N})} X_n \to X_{n+1}$ a family of functions. Let*

$$Z := \sum_{(x:\prod_{(n:\mathbb{N})} X_n)} \prod_{(n:\mathbb{N})} x_{n+1} = l_n(x_n) \;.$$

Then the projection $Z \to X_0$ is an equivalence.

Proof. Let G be the functor defined by $GY = 1 + Y$. Fix an element $z : Z$. Then z and l together define a G-algebra structure on X, regarded as a fibration over $\mathbb{N}$. Since $\mathbb{N}$ is the homotopy initial algebra of G, the type Sz of algebra sections of X is contractible. But Z is equivalent to

$$\sum_{(z:X_0)} \sum_{(x:\prod_{(n:\mathbb{N})} X_n)} (x_0 = z) \times \left(\prod_{n:\mathbb{N}} x_{n+1} = l_n(x_n) \right) \;,$$

which is exactly $\sum_{(z:X_0)} Sz \simeq X_0$. □

Lemma 12. *Let (X, π) be a chain, and let (X', π') be the* shifted *chain, defined by $X'_n := X_{n+1}$ and $\pi'_n := \pi_{n+1}$. Then the two chains have equivalent limits.*

Proof. Let L and L' be the limits of (X, π) and (X', π'), respectively. We have

$$\begin{aligned}
L' &\overset{(1)}{\simeq} \sum_{(y:\prod_{(n:\mathbb{N})} X_{n+1})} \prod_{(n:\mathbb{N})} \pi_{n+1} y_{n+1} = y_n \\
&\overset{(2)}{\simeq} \sum_{(x_0:X_0)} \sum_{(y:\prod_{(n:\mathbb{N})} X_{n+1})} (\pi_0 y_0 = x_0) \times \left(\prod_{n:\mathbb{N}} \pi_{n+1} y_{n+1} = y_n \right) \\
&\overset{(3)}{\simeq} \sum_{x:\prod_{(n:\mathbb{N})} X_n} (\pi_0 x_0 = x_0) \times \left(\prod_{n:\mathbb{N}} \pi_{n+1} x_{n+2} = x_{n+1} \right) \\
&\overset{(4)}{\simeq} \sum_{(x:\prod_{(n:\mathbb{N})} X_n)} \prod_{(n:\mathbb{N})} \pi_n x_{n+1} = x_n \\
&\overset{(5)}{\simeq} L \;,
\end{aligned}$$

where **(1)** and **(5)** are by definition. Equivalence **(2)** is given by adding a path with a free end point x_0 from $\pi_0 y_0$, which is a contractible type. Equivalence **(3)** is given by joining the first two components, and similarly in **(4)** the last two components are joined. □

The next lemma says that polynomial functors (see Definition 2) commute with limits of chains. Let (A, B) be a container with associated polynomial functor $P = P_{A,B}$. Let (X, π) be a chain with limit (L, p). Define the chain $(PX, P\pi)$ with $PX_n := P(X_n)$ and likewise for $P\pi$, and let L^P be its limit. The family of maps $Pp_n : PL \to PX_n$ determines a function $\alpha : PL \to L^P$.

Lemma 13. *The function α is an isomorphism.*

Proof. By "equational" reasoning we have

$$
\begin{aligned}
L^P &\overset{(6)}{\simeq} \sum_{(w:\prod_{(n:\mathbb{N})} \sum_{(a:A)} B(a) \to X_n)} \prod_{(n:\mathbb{N})} (P\pi_n) w_{n+1} = w_n \\
&\overset{(7)}{\simeq} \sum_{(a:\prod_{(n:\mathbb{N})} A)} \sum_{(u:\prod_{(n:\mathbb{N})} B(a_n) \to X_n)} \prod_{(n:\mathbb{N})} (a_{n+1}, \pi_n \circ u_{n+1}) = (a_n, u_n) \\
&\overset{(8)}{\simeq} \sum_{(a:\prod_{(n:\mathbb{N})} A)} \sum_{(p:\prod_{(n:\mathbb{N})} a_{n+1} = a_n)} \sum_{(u:\prod_{(n:\mathbb{N})} B(a_n) \to X_n)} \prod_{(n:\mathbb{N})} (p_n)_*(\pi_n \circ u_{n+1}) = u_n \\
&\overset{(9)}{\simeq} \sum_{(a:A)} \sum_{(u:\prod_{(n:\mathbb{N})} B(a) \to X_n)} \prod_{(n:\mathbb{N})} \pi_n \circ u_{n+1} = u_n \\
&\overset{(10)}{\simeq} \sum_{a:A} B(a) \to L \\
&\overset{(11)}{\simeq} PL
\end{aligned}
$$

where (6) and (11) are by definition, (7) is by swapping Π and Σ, (8) by expanding equality of pairs, (9) by applying Lemma 11 and (10) by universal property of L. Verifying that the composition of these isomorphisms is α is straightforward. □

Proof of Theorem 7. We now construct a terminal coalgebra for a container (A, B). Let $P = P_{A,B}$ the polynomial functor associated to the container. By recursion on $\mathbb{N}$ we define the chain

$$1 \xleftarrow{!} P1 \xleftarrow{P!} P^2 1 \xleftarrow{P^2!} P^3 \xleftarrow{P^3!} \dots$$

which for brevity we call (W, π), that is, $W_n := P^n 1$ and $\pi_n := P^n !$.

Let (L, p) be the limit of (W, π). If L' is the limit of the shifted chain, we have a sequence of equivalences

$$PL \overset{(12)}{\simeq} L' \overset{(13)}{\simeq} L$$

where (12) is given by Lemma 13 and (13) by Lemma 12. We denote this equivalence by $\mathsf{in} : PL \to L$, and its inverse by $\mathsf{out} : L \to PL$.

It is worth noting that the construction of L "does not raise the universe level", i.e., if A and B are contained in some universe $\mathcal{U}$, then L is contained in $\mathcal{U}$ as well. In other words, we only need one universe to carry out our construction of the final coalgebra.

We will now show that (L, out) is a final (A, B)-coalgebra. For this, let (C, γ) be any coalgebra, i.e., $\gamma : C \to PC$. The type of coalgebra morphisms $(C, \gamma) \Rightarrow (L, \mathsf{out})$ is given by

$$U := \sum_{f:C \to L} \mathsf{out} \circ f = Pf \circ \gamma \ .$$

We need to show that U is contractible. We compute as follows (see below the math display for intermediate definitions):

$$
\begin{aligned}
U &\overset{(14)}{\simeq} \sum_{f:C\to L} \mathsf{out}\circ f = Pf\circ\gamma \\
&\overset{(15)}{\simeq} \sum_{f:C\to L} \mathsf{out}\circ f = \mathsf{step}(f) \\
&\overset{(16)}{\simeq} \sum_{f:C\to L} \mathsf{in}\circ\mathsf{out}\circ f = \mathsf{in}\circ\mathsf{step}(f) \\
&\overset{(17)}{\simeq} \sum_{f:C\to L} f = \Psi(f) \\
&\overset{(18)}{\simeq} \sum_{c:\mathsf{Cone}} e(c) = \Psi(e(c)) \\
&\overset{(19)}{\simeq} \sum_{c:\mathsf{Cone}} e(c) = e(\Phi(c)) \\
&\overset{(20)}{\simeq} \sum_{c:\mathsf{Cone}} c = \Phi(c) \\
&\overset{(21)}{\simeq} \sum_{((u,q):\mathsf{Cone})} \sum_{(p:u=\Phi_0(u))} p_*(q) = \Phi_1(u)(q) \\
&\overset{(22)}{\simeq} \sum_{(u:\mathsf{Cone}_0)} \sum_{(p:u=\Phi_0 u)} \sum_{q:\mathsf{Cone}_1 u} p_*(q) = \Phi_1(u)(q) \\
&\overset{(23)}{\simeq} \sum_{t:1} 1 \\
&\simeq 1
\end{aligned}
$$

where we use the following definitions: The function $\mathsf{step}_Y : (C \to Y) \to (C \to PY)$ is defined as $\mathsf{step}_Y(f) := Pf \circ \gamma$ and $\Psi : (C \to L) \to (C \to L)$ is defined as $\Psi(f) := \mathsf{in} \circ \mathsf{step}_L(f)$. The map $\Phi : \mathsf{Cone} \to \mathsf{Cone}$ is the counterpart of Ψ on the side of cones. We define $\Phi(u, g) = (\Phi_0 u, \Phi_1 u(g)) : \mathsf{Cone} \to \mathsf{Cone}$ with

$$
\begin{aligned}
(\Phi_0 u)_0 &:= x \mapsto tt : C \to 1 = W_0 \\
(\Phi_0 u)_{n+1} &:= \mathsf{step}_{W_n}(u_n) : C \to W_{n+1} = PW_n
\end{aligned}
$$

and analogously for Φ_1 on paths. By e we denote the equivalence of Lemma 10 from right to left, and $\mathsf{Cone} = \sum_{(u:\mathsf{Cone}_0)} \mathsf{Cone}_1(u)$ is short for $\mathsf{Cone}(C)$. The equivalence **(16)** follows from in being an equivalence, and **(17)** follows from in and out being inverse to each other. We pass from maps into L to cones in **(18)**, using the equivalence of Lemma 10, while **(19)** uses the commutativity of the following square:

$$
\begin{array}{ccc}
\mathsf{Cone} & \xrightarrow{e} & (C \to L) \\
{\scriptstyle\Phi}\downarrow & & \downarrow{\scriptstyle\Psi} \\
\mathsf{Cone} & \xrightarrow{e} & (C \to L).
\end{array}
$$

In (**21**), identity in a sigma type is reduced to identity of the components, and in (**22**) the components are rearranged. Finally, step (**23**) consists of two applications of Lemma 11.

Altogether, this shows that for any coalgebra (C, γ), the type of coalgebra morphisms $(C, \gamma) \Rightarrow (L, \mathsf{out})$ is contractible. This concludes the construction of a final coalgebra associated to the signature (A, B), and thus, combined with Lemma 5, the proof of Theorem 7. □

From the construction of L we get the following corollary about the homotopy level of M-types:

Lemma 14. *The homotopy level of the (carrier of the)* M*-type asssociated to the signature* (A, B) *is bounded by that of the type of nodes* A*, that is,*

$$\mathsf{isofhlevel}_n(A) \to \mathsf{isofhlevel}_n(\mathsf{M}(A, B)) \ .$$

Example 15. We continue the example of streams of Example 6, with $A = A_0$ the type of nodes. In that case, the chain considered in the proof of Theorem 7 is given by $W_n = P^n(1) = A^n$, and the map $\pi_n : A^{n+1} \to A^n$ chops of the $(n+1)$th element of any $(n+1)$-tuple. The limit L is hence given by $A^{\mathbb{N}} = (\mathbb{N} \to A)$. The type of streams over A has the same homotopy level as the type A of nodes.

We conclude this section with a proof of the *principle of coinduction*:

Definition 16 (Bisimulation)**.** Let (C, γ) be an coalgebra for some signature (A, B) and let $\mathcal{R} : C \to C \to \mathcal{U}$ a binary relation. Define $\overline{\mathcal{R}} := \sum_{(a:C)} \sum_{(b:C)} \mathcal{R}(a)(b)$ along with two projections $\pi_1^{\overline{\mathcal{R}}}(a, b, p) := a$ and $\pi_2^{\overline{\mathcal{R}}}(a, b, p) := b$.

An (A, B)-*bisimulation* is a relation $\mathcal{R}$, together with a map $\alpha_{\mathcal{R}} : \overline{\mathcal{R}} \to P(\overline{\mathcal{R}})$ such that both $\pi_1^{\overline{\mathcal{R}}}$ and $\pi_2^{\overline{\mathcal{R}}}$ are P-coalgebra morphisms:

$$\begin{array}{ccccc}
C & \xleftarrow{\pi_1^{\overline{\mathcal{R}}}} & \overline{R} & \xrightarrow{\pi_2^{\overline{\mathcal{R}}}} & C \\
\downarrow{\scriptstyle \gamma} & & \downarrow{\scriptstyle \alpha_{\mathcal{R}}} & & \downarrow{\scriptstyle \gamma} \\
P(C) & \xleftarrow[P(\pi_1^{\overline{\mathcal{R}}})]{} & P(\overline{R}) & \xrightarrow[P(\pi_2^{\overline{\mathcal{R}}})]{} & P(C)
\end{array}$$

We say that bisimulation is an *equivalence bisimulation* when the underlying relation is an equivalence relation.

Lemma 17. *The identity relation* $\cdot = \cdot$ *over an* (A, B)*-coalgebra* C *is an equivalence bisimulation. We write* Δ_C *for* $\overline{\cdot = \cdot}$*.*

Theorem 18 (Coinduction proof principle)**.** *Let* (L, out) *be the final coalgebra. For every bisimulation* $\overline{\mathcal{R}}$ *over* M*,* $\overline{\mathcal{R}} \subseteq \Delta_M$*. Put differently, for all* m *and* m' *in* M*,*

$$\mathcal{R}(m, m') \to m = m'.$$

Proof. Since (L, out) is the final coalgebra, for any coalgebra (C, γ) there exists a unique coalgebra morphism $\mathsf{unfold}_C : C \to L$. It follows that $\pi_1^{\overline{\mathcal{R}}} = \mathsf{unfold}_{\overline{\mathcal{R}}} = \pi_2^{\overline{\mathcal{R}}}$. Finally, given $r : \mathcal{R}(m, m')$ we have $m = \pi_1^{\overline{\mathcal{R}}}(m, m', r) = \pi_2^{\overline{\mathcal{R}}}(m, m', r) = m'$. □

5. Indexed M-types

In this section, we briefly state the main definitions for *indexed* M-types. The difference to plain M-types is that the type of nodes of an indexed M-type is actually given by a family of types, indexed by a type of sorts I.

Definition 19. An **indexed container** is given by a quadruple (I, A, B, r) such that $I : \mathcal{U}$ is a type, $i : I \vdash A(i) : \mathcal{U}$ is a family of types dependent on I, B is a family $i : I, a : A(i) \vdash B_i(a) : \mathcal{U}$ and r specifies the "sort" of the subtrees, i.e., $r : \prod_{(i:I)} \prod_{(a:A(i))} B_i(a) \to I$.

Definition 20. The polynomial functor P associated to an indexed container (I, A, B, r) is an endofunction on the type $I \to \mathcal{U}$:

$$(PX)(i) := \sum_{(a:A(i))} \prod_{(b:B_i(a))} X(r_{i,a}(b)) \ .$$

The functorial action on morphisms is, analogously to Definition 2, given by post-composition.

Coalgebras for indexed containers, and their morphisms, are defined completely analogously to Definition 3. Again, we prove that terminal coalgebras for indexed containers exist uniquely:

Theorem 21. *Let (I, A, B, r) be an indexed container. Then the associated indexed* M*-type is uniquely specified and can be constructed in the type theory described in Section 2.*

An example of a coinductive type that needs indices to be expressed as an M-type is a coinductive formulation of *equivalence of types*, to our knowledge due to T. Altenkirch:

Example 22. Let $I := \mathcal{U} \times \mathcal{U}$, and let $A : I \to \mathcal{U}$ be defined by $A(X, Y) := (X \to Y) \times (Y \to X)$. Define B as $B(X, Y)(f, g) := X \times Y$ and $r(X, Y)(f, g)(x, y) := (f(x) = y) \times (x = g(y))$. Then the associated M-type is a family $\mathsf{M} : I \to \mathcal{U}$ and $M(A, B)$ is equivalent to $A \simeq B$.

6. Formalization

The proofs contained in this paper have been formalised in the proof assistant Agda in a self-contained development forked off of the HoTT library agda-base. The proof has been type-checked by version 2.4.2.2 of Agda, and can be found in the m-types repository at https://github.com/HomotopyTypeTheory/m-types/. The relevant modules are under `container/m/from-nat`. A browsable version can be found at http://homotopytypetheory.github.io/m-types/master/.

The formalised proofs deal with the indexed case (Section 5) directly, but apart from that, they correspond closely to the informal proofs presented here. In particular, they make heavy use of the "equational reasoning" technique to prove equivalences between types.

In fact, it is often the case that proving an equivalence between types A and B "directly", i.e. by defining functions $A \to B$ and $B \to A$, and then proving that they compose to identities in both directions, is unfeasibly hard, due to the complexity of the terms involved.

However, in most cases, we can construct an equivalence between A and B by composition of several simple equivalences. Those simple building blocks range from certain ad-hoc equivalences that make specific use of the features of the two types involved, to very general and widely applicable "rewriting rules", like the fact that we can swap a Σ-type with a Π-type (sometimes called the *constructive axiom of choice* (Univalent Foundations Program [12])).

By assembling elementary equivalences, then, we automatically get both a function $A \to B$ *and* a proof that it is an equivalence. However, sometimes care is needed to ensure that the resulting function has the desired computational properties.

An important consideration during the formalisation of the proof of Theorem 7 was keeping the size of the terms reasonably short. For example, in one early attempt, the innocent-looking term in was being normalised into a term spanning more than 12000 lines.

The explosion in term size was clearly causing performance issues during type-checking, which resulted in Agda running out of memory while checking apparently trivial proofs.

We solved this problem by moving certain definitions (like that of in itself) into an *abstract* block, thereby preventing Agda from expanding it at all. Of course, this means that we lost all the computational properties of certain functions, so we had to abstract out their computational behaviour in the form of propositional equalities, and manually use them in the proof of Theorem 7. This work-around is the source of most of the complications in the formal proof.

7. Conclusion and future work

We have shown how to construct a class of coinductive types from the basic type constructors and natural numbers in Homotopy Type Theory. Our construction follows a well-known pattern, that is known to work in type theory with identity reflection rule.

We work in a univalent universe, but we make minimal use of univalence itself: Lemma 5 is the only result that uses it directly, and elsewhere univalence only appears indirectly through the use of functional extensionality. We intend to make these dependencies more explicit in future developments of the formalisation.

Finally, the coinductive types we construct do not satisfy the expected computation rules *judgmentally*, but only *propositionally*. This fact would justify adding coinduction as a primitive rather than a derived notion—provided that judgmental computation rules are validated by the intended semantics. Those semantic questions are left for future work.

References

[1] Discussion on coinductive types on HoTT mailing list. URL: https://groups.google.com/d/msg/homotopytypetheory/tYRTcI2Opyo/PIrI6t5me-oJ.

[2] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. "Containers: Constructing strictly positive types". In: *Theor. Comput. Sci.* 342.1 (2005), pp. 3–27. DOI: 10.1016/j.tcs.2005.06.002.

[3] Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. "Univalent categories and the Rezk completion". In: *Mathematical Structures in Computer Science* FirstView (Jan. 2015), pp. 1–30. DOI: 10.1017/S0960129514000486. URL: http://journals.cambridge.org/article_S0960129514000486.

[4] Thorsten Altenkirch, Neil Ghani, Peter Hancock, Conor McBride, and Peter Morris. *Indexed Containers*. available at http://www.cs.nott.ac.uk/~txa/publ/jcont.pdf.

[5] Jeremy Avigad, Krzysztof Kapulkin, and Peter LeFanu Lumsdaine. "Homotopy limits in type theory". In: *Mathematical Structures in Computer Science* FirstView (Jan. 2015), pp. 1–31. DOI: 10.1017/S0960129514000498. URL: http://journals.cambridge.org/article_S0960129514000498.

[6] Steve Awodey, Nicola Gambino, and Kristina Sojakova. "Inductive Types in Homotopy Type Theory". In: *LICS*. IEEE, 2012, pp. 95–104.

[7] Benno van den Berg and Federico De Marchi. "Non-well-founded trees in categories". In: *Ann. Pure Appl. Logic* 146.1 (2007). arXiv:math/0409158, pp. 40–59.

[8] James Cranch. *Concrete Categories in Homotopy Type Theory*. 2013. arXiv:1311.1852.

[9] Chris Kapulkin, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. *The Simplicial Model of Univalent Foundations*. 2014. arXiv:1211.2851v2.

[10] Per Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.

[11] Kristina Sojakova. "Higher Inductive Types as Homotopy-Initial Algebras". In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. Ed. by Sriram K. Rajamani and David Walker. ACM, 2015, pp. 31–42. DOI: 10.1145/2676726.2676983. URL: http://doi.acm.org/10.1145/2676726.2676983.

[12] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: http://homotopytypetheory.org/book, 2013.