# Alternative Impredicative Encodings of Inductive Types

**MSc Thesis** *(Afstudeerscriptie)*

written by

**Xavier Ripoll Echeveste**
(born May 19, 1997 in Tarragona, Catalonia)

under the supervision of **Dr Benno van den Berg** and **Daniël Otten MSc**, and submitted to the Examinations Board in partial fulfillment of the requirements for the degree of

**MSc in Logic**

at the *Universiteit van Amsterdam.*

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

## Abstract

Intensional dependent type theories with an impredicative universe admit an encoding of inductive types similar to the one developed for System F. This technique suffers from one fatal flaw: its recursion principle (non-dependent elimination) does not satisfy the propositional $\eta$-rule. Equivalently, it does not have an induction principle (dependent elimination). Awodey, Frey, and Speight (2018) make a proposal to construct inductive types using an impredicative universe by taking the subset of the System F definition satisfying a property named *naturality*, expanding the theory with $\Sigma$-types, identity types, and function extensionality. There is a catch: it can only eliminate into 0-types inside the impredicative universe.

In this thesis, we introduce the alternative notion of *inductivity*. We first prove that it is as strong as naturality—by constructing coproducts and then $\mathsf{W}$-types with their full elimination principles with respect to sets. Later on, we consider Shulman's claim (2018) that one can lift the h-level restriction by using an intermediate result of his own and including natural numbers in the system. We adapt inductivity (by applying multiple layers thereof) and manage to allow elimination to arbitrary types in the universe, again for coproducts and $\mathsf{W}$-types. Finally, we start studying a categorical generalization of this problem, namely finding the initial algebra of an endofunctor of the impredicative universe, by means of introducing one last property—*initiality*.

**Acknowledgements**

I would like to give thanks:

To both of my co-supervisors. To Benno van den Berg, for spending time in finding an adequate topic for me, for a constant follow-up, and for closing some proofs when they needed a push and dismantling others when they were too good to be true. To Daniël Otten, for investing so much thought and time even before being officially involved in the thesis, for his sharpness in recalling all sorts of results instantly, and for his excitement for all my proposals. And, to both, for the long meetings, where most of the good ideas came up and we could empathize over the frustration of being stuck. It feels nice to have academic support, but it feels much better to share academic enthusiasm.

To the members of my committee, Malvin Gattinger, Herman Geuvers, and Katia Shutova, for taking the time and putting the effort to read my thesis, and the clear interest they showed through their questions.

To my flatmates, Sabina, Dario, and Elias, for the camaraderie, which upheld each other through cold, rain, and logic.

To my mom and my sister, for their constant support despite their distance from both the author and the topic.

Finally, to Debbie, who has seen me through this whole project and whose good influence she can rest assured can be found on every word of this thesis, and without whose infinite patience, understanding, and love I would have never managed to write so much as a paragraph.

# Contents

# Chapter 1

# Introduction

## 1.1 Inductive types

Set theory characterizes entities by what they contain and how they are structured internally. Category theory uses the morphisms between objects to describe how they behave. On the other hand, type theory uses the ways elements can be constructed (**introduction rules**) and used (**elimination rules**) and the compatibility between the two (**computation** and **uniqueness** rules) to describe the type they belong to; together with **formation rules** to construct the type itself. For example, the coproduct type can be characterized by the following set of rules:

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash B : \mathcal{U}_i}{\Gamma \vdash A + B : \mathcal{U}_i}\ +\text{-FORM}$$

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash B : \mathcal{U}_i}{\Gamma \vdash \mathsf{inl} : A \to A + B}\ +\text{-INTR}_1 \qquad \frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash B : \mathcal{U}_i}{\Gamma \vdash \mathsf{inr} : B \to A + B}\ +\text{-INTR}_2$$

$$\frac{\Gamma \vdash X : \mathcal{U}_i \qquad \Gamma \vdash \mathsf{inl}_X : A \to X \qquad \Gamma \vdash \mathsf{inr}_X : B \to X}{\Gamma \vdash \mathsf{rec}(X, \mathsf{inl}_X, \mathsf{inr}_X) : A + B \to X}\ +\text{-ELIM}$$

$$\frac{\Gamma \vdash X : \mathcal{U}_i \qquad \Gamma \vdash \mathsf{inl}_X : A \to X \qquad \Gamma \vdash \mathsf{inr}_X : B \to X}{\Gamma \vdash \mathsf{rec}(X, \mathsf{inl}_X, \mathsf{inr}_X) \circ \mathsf{inl} \equiv \mathsf{inl}_X : A \to X}\ +\text{-}\beta_1$$

$$\frac{\Gamma \vdash X : \mathcal{U}_i \qquad \Gamma \vdash \mathsf{inl}_X : A \to X \qquad \Gamma \vdash \mathsf{inr}_X : B \to X}{\Gamma \vdash \mathsf{rec}(X, \mathsf{inl}_X, \mathsf{inr}_X) \circ \mathsf{inr} \equiv \mathsf{inr}_X : B \to X}\ +\text{-}\beta_2$$

$$\frac{\begin{array}{c}\Gamma \vdash X : \mathcal{U}_i \\ \Gamma \vdash \mathsf{inr}_X : B \to X \qquad \Gamma \vdash \mathsf{inl}_X : A \to X \qquad \Gamma \vdash f : A + B \to X\end{array}}{\Gamma \vdash \mathsf{rec}(X, f \circ \mathsf{inl}, f \circ \mathsf{inr}) = f : A + B \to X}\ +\text{-}\eta$$

The way this system is set up, we do not (immediately) get a statement about what shape the inhabitants of the type have, but rather one about how to use those inhabitants: to define a function out of this type, it is sufficient to define it on the canonical elements.

Inductive types are types that are freely generated by a set of constructors, i.e., functions, satisfying certain constraints, that return elements of the inductive type at hand. This covers some very big categories[1] of types, ranging widely in complexity. We can broadly classify inductive types according to two criteria.

[1]In the nonmathematical sense of the word.

First, on whether they take other types as arguments. If they do, they are strictly speaking type constructors (although our type system will allow us to encode the type constructors as types). If they do not, they are just constants in some universe of types. For example, the coproduct type $A + B$ requires specifying types $A$ and $B$, whereas the natural numbers $\mathbb{N}$ stands on its own.

Second, whether they use induction in the traditional sense of the word. This is to say, when we want to prove statements about the terms of the type, we need to use some sort of induction hypothesis, which assumes that the statement has already been proved for some of the terms. That an inductive type require an induction hypothesis is equivalent to whether any of its constructors has elements of the type itself as arguments. For example, $\mathbb{N}$ has the constructor $\mathsf{succ} : \mathbb{N} \to \mathbb{N}$. The consequence is that, when proving a property $P$ over $\mathbb{N}$, one assumes that $P$ holds for an arbitrary element $n : \mathbb{N}$ and then proves that $P$ also holds for $\mathsf{succ}\, n$. This makes $\mathbb{N}$ a "proper" inductive type. On the other hand, the unit type has a single constructor $\star : \mathbb{1}$, which does not take any arguments from $\mathbb{1}$ itself, so it is "degenerate", in the sense that proving a property does not require assuming it has been proven for any other term of the type. The most interesting cases are in the former class, but we will study all kinds.

The following table illustrates this classification, including some types that we will see throughout this work.

| | Degenerate | Proper |
|---|---|---|
| Constants | $\mathbb{0}$, $\mathbb{1}$, $\mathbb{2}$ | $\mathbb{N}$ |
| Constructors | $A + B$ | $\mathsf{W}_A B$ |

Table 1.1: A classification of inductive types according to two properties.

Let us see a complete example of the rules governing an inductive type. We will use the type $\mathbb{N}$ of natural numbers.

- First we need a **formation** rule. In the case of the natural numbers, this rule just states that $\mathbb{N}$ is a type. For the inductive types that are type constructors, this rule will require other inputs.

- For the **introduction** principle, we have two constructors:

$$0 : \mathbb{N}$$
$$\mathsf{succ} : \mathbb{N} \to \mathbb{N}$$

  This means that the canonical terms of $\mathbb{N}$ are 0, and the application of $\mathsf{succ}$ to other canonical terms of the type. The intended model of $\mathbb{N}$ contains *only* the canonical terms, this is to say, those constructed by the repeated application of 0 and $\mathsf{succ}$.

- The **elimination** principle: for any type family $C : \mathbb{N} \to \mathcal{U}_i$ (where $\mathcal{U}_i$ is some universe of types) such that we have proofs of $C(0)$ and $\forall n\; C(n) \to C(\mathsf{succ}\, n)$, we have a dependent function $f : \prod_{n:N} C(n)$ (i.e. one such that $f(n) : C(n)$ for a given $n : \mathbb{N}$), which implies that $C$ is inhabited for *all* $n : \mathbb{N}$. As stated, this is known as the **induction principle** or dependent eliminator. The degenerate non-dependent version for constant type families is known as the **recursion principle** or non-dependent eliminator. It can be stated as: given another type $X$ with associated functions similar to those of $\mathbb{N}$, i.e.:

$$\begin{aligned} z &: X \\ s &: X \to X \end{aligned}$$

There is a function $f : \mathbb{N} \to X$. Both versions of this principle are usually embodied by a function, called the eliminator (or recursor in its non-dependent case).

- The elimination principle comes hand in hand with a **computation** principle, which states that the functions $f$ constructed with the elimination principle commute with both constructors:

$$f0 = z \tag{1.1}$$
$$f(\mathsf{succ}\, n) = s(fn) \tag{1.2}$$

  The definitional versions of these equations are known as **$\beta$-equalities** or **$\beta$-rules**.

- Besides the introduction and elimination principles, we have a **uniqueness** principle: the functions out of $\mathbb{N}$ that respect the constructors are unique. Or, in other words, if $f, g : \prod_{n:\mathbb{N}} C(n)$ both satisfy 1.1, then $f = g$. This last condition ensures that the type has only one model (up to isomorphism): there is only one way to obtain elements of $\mathbb{N}$ (namely, through the constructors), and to determine a function out of the inductive type, it is sufficient and necessary to determine its action on these canonical terms. This is sometimes refered to as **$\eta$-rule**. As we will see, this rule can be deduced from the dependent elimination principle.

The names recursion and induction are highly suggestive in the contexts of computer science and mathematics, respectively. Recursion is often thought of as a tool used to "build" functions: by just providing the output on 0 and on succ, we get a function $\mathbb{N} \to X$. On the other hand, induction is often used to prove properties about the naturals, encoded as types $C : \mathbb{N} \to \mathcal{U}$. Indeed, the induction principle of the natural numbers as an inductive type corresponds with the induction of the naturals in mathematics.

Inductive types cover a wide range of mathematical and computational utilities. It is very desirable to have them available in our type systems. A priori, a simple way of achieving that is by including their formation, introduction, elimination, computation, and uniqueness rules as part of the judgmental rules that make up our type system.

A more minimalistic—but arduous—approach is to build those types inside the theory out of more basic type constructors, such as sum and product types, and prove that they satisfy the desired rules. A most famous example of this method is the one used in System F.

## 1.2 Inductive types in System F

System F, or polymorphic lambda calculus, was invented by Girard [5] and Reynlods [7]. It is the extension of simply typed lambda calculus with functions that can also depend on types, on top of terms. For instance, a function in System F can have a type $\Pi X. X \to X$, i.e. one that takes a *type* $X$ as argument, and an element of $X$, and returns another element of $X$. An inhabitant of this type is the polymorphic identity $\Lambda X.\lambda x^X.x$.

It is a remarkable property of System F that it allows some inductive types to be defined *within the system* using the signature of their intended constructors. For instance, the coproduct of two types $A$ and $B$, written as $A + B$, should have two constructors: a left injection $\mathsf{inl} : A \to A+B$, and a right injection $\mathsf{inr} : B \to A+B$. Its System F encoding would be:

$$A + B := \Pi X.(A \to X) \to (B \to X) \to X$$

There are several things to note.

First, observe that we quantify over all types $X$. One of the characteristic features of System F is that it is *impredicative*, i.e., it allows us to define types that quantify over all types, including themselves. This pattern makes use of such impredicativity to guarantee that $A + B$ can eliminate into any type (including itself), provided we supply the necessary arguments (in this case, functions $A \to X$ and $B \to X$). To make this more clear: suppose we had a term $x : A + B$. Then, impredicativity allows us to compute $x(A + B)$, which would have the type $(A \to A+B) \to (B \to A+B) \to A+B$. Predicative type systems usually implement a hierarchy of type universes, so that functions that quantify over a universe lie in a strictly higher one, so as to disallow this form of self-application.

Second, the type $A + B$ is essentially its own eliminator. The elimination rule of the coproduct states that, to define a function out of the coproduct of $A$ and $B$ into a type $X$, it suffices to provide functions from $A$ into $X$ and from $B$ into $X$. This is exactly what each term of the type $A + B$ defined above does. A function $A + B \to Y$, given $l : A \to Y$ and $r : A \to Y$, can be obtained as $\lambda x^{A+B}.xYlr$.

Third, and most critically, inductive types defined in this manner lack a uniqueness principle [9]. As already mentioned, one important feature of type theory is the ability to characterize the functions out of a type based only on how they act on the constructors for said type. System F-style inductive types allow us to *construct* functions out of them, but these are not guaranteed to be unique. In this thesis we will work on a type system with impredicativity to try to improve the System F encoding of inductive types to satisfy the uniqueness principle.

## 1.3 Our system

For the work done in this thesis, we will enrich System F in several ways. We will assume not only $\Pi$-types, but also $\Sigma$-types and intensional identity types, as well as a cumulative hierarchy of universes. This system is taken from [12]—wherein its rules are laid out explicitly and its consistency is proven—, but it closely resembles the standard homotopy type theory system of [6]. Nonetheless, we will give here a justification of all the ways that our system deviates from it.

Firstly, and most importantly, we add at the bottom of the tower of universes $\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \ldots$ a new impredicative universe that we shall denote by simply $\mathcal{U}$, such that $\mathcal{U} : \mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \ldots$. Hence, we will not partake in the abuse of notation done within the book: for us, $\mathcal{U}$ will always mean the impredicative universe, and $\mathcal{U}_i$ will be used where necessary to refer to an unspecified universe above $\mathcal{U}$. Remember that this is a cumulative hierarchy: if $A : \mathcal{U}_i$, then $A : \mathcal{U}_{i+1}$. We call the types in $\mathcal{U}$ small types, and the rest large types.

We inherit the standard formation rule of Π:

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma, x : A \vdash B[x] : \mathcal{U}_i}{\Gamma \vdash \prod_{x:A} B[x] : \mathcal{U}_i}\ \Pi\text{-}\textsc{form}_1$$

To which we add an analogous rule for $\mathcal{U}$:

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma, x : A \vdash B[x] : \mathcal{U}}{\Gamma \vdash \prod_{x:A} B[x] : \mathcal{U}}\ \Pi\text{-}\textsc{form}_2$$

But we also add another one that makes $\mathcal{U}$ impredicative:

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma, x : A \vdash B[x] : \mathcal{U}}{\Gamma \vdash \prod_{x:A} B[x] : \mathcal{U}}\ \Pi\text{-}\textsc{form}_3 \tag{1.3}$$

This reads: a type of dependent functions that eliminate into small types is also itself small, no matter the size of the domain type.

Secondly, we do not adopt the univalence axiom. Even though it is not incompatible with the system, we do not make use of its full power and hence do without it. We do, nonetheless, accept the axiom of function extensionality, which asserts that, whenever two functions are propositionally equal pointwise, they are propositionally equal to each other. The objective of this axiom is to make the type $f \sim g :\equiv \prod_{x:X} fx = gx$ of homotopies between $f$ and $g$ equivalent to the type of propositional identifications $f = g$. To be more precise, the axiom states that the function $\mathsf{happly} : f = g \to f \sim g$ has a quasi-inverse $\mathsf{funext} : f \sim g \to f = g$. This axiom will actually be quite vital to our arguments regarding W-types and is also necessary to invoke a key theorem from [10].

For the first part of this thesis (Chapter 2), reference to types restricted by their homotopy level (or h-level) will be quite important, hence we also use Contr to refer to the types with h-level $-2$ in $\mathcal{U}$, Prop for those with h-level $-1$, and Set for those with h-level 0. In other words:

$$\mathsf{Contr} :\equiv \sum_{X:\mathcal{U}} \mathsf{isContr}(X)$$
$$\mathsf{Prop} :\equiv \sum_{X:\mathcal{U}} \mathsf{isProp}(X)$$
$$\mathsf{Set} :\equiv \sum_{X:\mathcal{U}} \mathsf{isSet}(X)$$

where $\mathsf{isContr}(X) :\equiv \sum_{x:X}\prod_{y:X} x = y$, $\mathsf{isProp}(X) :\equiv \prod_{x,y:X} \mathsf{isContr}(x = y)$, and $\mathsf{isSet}(X) :\equiv \prod_{x,y:X} \mathsf{isProp}(x = y)$. In general, by h-level we mean:

$$\mathsf{isType}_{-2}(X) :\equiv \sum_{x:X}\prod_{y:X} x = y$$
$$\mathsf{isType}_{n+1}(X) :\equiv \prod_{x,y:X} \mathsf{isType}_n(x = y)$$

and then

$$\mathsf{Type}_{n+1} :\equiv \sum_{X:\mathcal{U}} \mathsf{isType}_{n+1}(X)$$

With the special names Contr, Prop, and Set for $n = -2, -1$, and 0, correspondingly. The numeration starting at $-2$ is due to historical reasons. We will almost always abuse notation and write $X : \mathsf{Set}$ when we mean $(X, s) : \mathsf{Set}$, etc.

The justification for these choices of nomenclature is widely discussed in [6, Chapter 7], but briefly, contractible types are those that contain exactly one element (up to propositional equality), propositions those that contain at most one, and sets those which may contain any amount of elements, but whose identifications are unique, i.e. there is at most one proof that two elements are equal.

It is useful, in this context, to remember from [8, Corollary 10.3.8] that:

**Lemma 1.1.** *Let $X : \mathcal{U}$ and $Y : X \to \mathsf{Prop}$. Then, $\mathsf{pr}_1 : \sum_{x:X} Y(x) \to X$ is an embedding.*

Where $\mathsf{pr}_1$ is the polymorphic first projection taking $(x, y)$ to $x$. An embedding is a function $f : A \to B$ such that $\mathsf{ap}_f : (x = y) \to (fx = fy)$ is an equivalence for all $x, y : A$. That $\mathsf{pr}_1$ is an embedding essentially warrants that for all $(x, y), (x', y') : \sum_{x:X} Y(x)$, $(x, y) = (x', y')$ if and only if $x = x'$, or, in other words, the side property $Y(x)$ is irrelevant when comparing terms of $\sum_{x:X} Y(x)$ for equality.

We will also use:

**Lemma 1.2.** *Let $X : \mathcal{U}$, $Y : X \to \mathsf{Type}_n$. Then, $\prod_{x:X} Y(x) : \mathsf{Type}_n$.*

*Proof.* $\prod_{x:X} Y(x)$ is in $\mathcal{U}$ by impredicativity, and it has h-level $n$ by [6, Theorem 7.1.9]. □

Finally, given the heavy use of $\Sigma$-types that we will make, we ease the reading by writing $\lambda(x, y).f(x)$ instead of $\lambda z.f(\mathsf{pr}_1(z))$, and similarly for longer tuples and other projections. This notation does not alter the validity of proofs, by virtue of the elimination rule for product types, which states that defining a function out of a product amounts to defining a function for its canonical elements—the tuples. This is called **pattern matching**.

## 1.4 Characterization of inductive types

The System F technique allows us to encode a wide range of inductive types. The general pattern is as follows:

$$\Pi X.(A_0^0 \to \cdots \to A_{k_0}^0 \to X) \to \cdots \to (A_0^n \to \cdots \to A_{k_n}^n \to X) \to X$$

Each of the intermediate types $A_0^i \to \cdots \to A_{k_0}^i \to X$ is the signature of each of the constructors of the intended type, wherein the $A_j^i$ are other types that may or may not involve $X$ again. If they do, though, it must always appear in a positive position (i.e. to the left of an even number of arrows $\to$). In our system, though, we need to raise this restriction to only *strict* positive positions (i.e. never to the left of an arrow $\to$). This is a quirk of dependent type theories, described in [4][6, Section 5.6], which would allow to reproduce Curry's paradox, making our type system inconsistent.

For the time being, we will focus on the example of the coproduct. Later on we will generalize, first to W-types in Chapters 2 and 3, and then to algebras for arbitrary endofunctors in Chapter 4.

Take, as seen earlier, the type of the (almost) coproduct $A + B$ in System F:

$$\Pi X.(A \to X) \to (B \to X) \to X$$

Translated into our system, we can rewrite it as:

$$\prod_{X:\mathcal{U}} (A \to X) \to (B \to X) \to X$$

The choice of domain $\mathcal{U}$, instead of any higher universe, is to warrant the use of impredicativity.

By uncurrying the chain of products:

$$A +_0 B :\equiv \prod_{(X, \mathsf{inl}_X, \mathsf{inr}_X) : \sum_{X:\mathcal{U}} (A \to X) \times (B \to X)} X$$

We define this to be our base type $A +_0 B$. For it, we can define the left and right injections:

$$\begin{aligned} \mathsf{inl}_0 &: A \to A +_0 B \\ \mathsf{inl}_0\, a &:\equiv \lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\, \mathsf{inl}_X\, a \end{aligned}$$

$$\begin{aligned} \mathsf{inr}_0 &: B \to A +_0 B \\ \mathsf{inr}_0\, a &:\equiv \lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\, \mathsf{inr}_X\, a \end{aligned}$$

Which act as the constructors for our inductive type.

Here comes the trick. Because the codomain of $A +_0 B$ is a type in $\mathcal{U}$ for every input, the $\Pi$-FORM$_3$ rule (1.3) guarantees that $A +_0 B$ stays also inside of $\mathcal{U}$. By furnishing it with functions $\mathsf{inl}_0 : A \to A +_0 B$ and $\mathsf{inr}_0 : B \to A +_0 B$, it becomes elegible to be an argument to itself: $x : A +_0 B \vdash x(A +_0 B, \mathsf{inl}_0, \mathsf{inr}_0) : A +_0 B$. This simple form of self-application will be the focus of most developments we do.

Conceptually, we always consider the constructors together with their codomain:

**Definition 1.1.** *The type of (coproduct of A and B) algebras is* $\mathcal{U}_{A+B} :\equiv \sum_{X:\mathcal{U}} (A \to X) \times (B \to X)$.

This type contains all the possible types that have associated coproduct-like constructors. Among these, if it exists, will be the *actual* coproduct of $A$ and $B$.

Associated to each pair of algebras $(X, \mathsf{inl}_X, \mathsf{inr}_X)$ and $(Y, \mathsf{inl}_Y, \mathsf{inr}_Y)$ comes a notion of "morphism":

**Definition 1.2.** *A morphism between algebras* $(X, \mathsf{inl}_X, \mathsf{inr}_X)$ *and* $(Y, \mathsf{inl}_Y, \mathsf{inr}_Y)$ *is a function* $f : X \to Y$ *such that* $f(\mathsf{inl}_X\, a) = \mathsf{inl}_Y\, a$ *and* $f(\mathsf{inr}_X\, b) = \mathsf{inr}_Y\, b$ *for every* $a : A$, $b : B$. *If* $f$ *is a morphism, we write* $f : (X, \mathsf{inl}_X, \mathsf{inr}_X) \to (Y, \mathsf{inl}_Y, \mathsf{inr}_Y)$.

The word morphism comes from the categorical underpinning of the theory, that we will detail in Chapter 4. In fact, the algebras form a category, if we take the morphisms as defined above under propositional equality. It is useful and easy to notice now that morphisms are indeed closed under composition and identities. The equations in the definition of morphism basically state that the underlying function is compatible with the constructors on both sides.

Let us state the elimination principle:

**Definition 1.3.** *A weak coproduct of A and B is* $(I, \mathsf{inl}_I, \mathsf{inr}_I) : \mathcal{U}_{A+B}$ *such that there is a recursor* $\mathsf{rec}_I : \prod_{(X, \mathsf{inl}_X, \mathsf{inr}_X) : \mathcal{U}_{A+B}} I \to X$, *such that* $\mathsf{rec}_I(X, \mathsf{inl}_X, \mathsf{inr}_X) \circ \mathsf{inl}_I \sim \mathsf{inl}_X$ *and* $\mathsf{rec}_I(X, \mathsf{inl}_X, \mathsf{inr}_X) \circ \mathsf{inr}_I \sim \mathsf{inr}_X$ *for any* $(X, \mathsf{inl}_X, \mathsf{inr}_X) : \mathcal{U}_{A+B}$.

The two side conditions

$$\begin{aligned} \mathsf{rec}_I(X, \mathsf{inl}_X, \mathsf{inr}_X) \circ \mathsf{inl}_I \sim \mathsf{inl}_X \\ \mathsf{rec}_I(X, \mathsf{inl}_X, \mathsf{inr}_X) \circ \mathsf{inr}_I \sim \mathsf{inr}_X \end{aligned}$$

are the $\beta$-equalities of the coproduct. Usually, for $(I, \mathsf{inl}_I, \mathsf{inr}_I)$ with $\mathsf{rec}_I$ to be an inductive type, they are required to be true definitionally:

$$\mathsf{rec}_I(X, \mathsf{inl}_X, \mathsf{inr}_X) \circ \mathsf{inl}_I \equiv \mathsf{inl}_X$$
$$\mathsf{rec}_I(X, \mathsf{inl}_X, \mathsf{inr}_X) \circ \mathsf{inr}_I \equiv \mathsf{inr}_X$$

When we only ask for the propositional (or, equivalently in our case, homotopical) version, we say that $(I, \mathsf{inl}_I, \mathsf{inr}_I)$ and $\mathsf{rec}_I$ form a homotopy-inductive type. For the theory developed in our thesis, we will focus only on homotopy-inductive types, and we will drop the prefix "homotopy-" for convenience. Nonetheless, many times it will be possible to (and we will) prove the definitional version of these equalities. Observe that, under our definition, the $\beta$-equalities mean precisely that $\mathsf{rec}_I(X, \mathsf{inl}_X, \mathsf{inr}_X)$ is a morphism. See [6, Section 5.5] for an extensive discussion on the advantages and drawbacks of homotopy-inductive types over inductive types with definitional $\beta$.

As highlighted earlier, $A +_0 B$ acts as its own recursor. More precisely, each term of $A +_0 B$ determines its own image for every possible type $(Y, \mathsf{inl}_Y, \mathsf{inr}_Y)$.

**Proposition 1.3.** *$(A +_0 B, \mathsf{inl}_0, \mathsf{inr}_0)$ is a weak coproduct.*

*Proof.* For the recursor, we define

$$\mathsf{rec}_0(X, \mathsf{inl}_X, \mathsf{inr}_X) :\equiv \lambda\alpha.\alpha(X, \mathsf{inl}_X, \mathsf{inr}_X)$$

We show that $\mathsf{rec}_0(X, \mathsf{inl}_X, \mathsf{inr}_X)$ is a morphism by function extensionality. Let $a : A$:

$$\begin{aligned}
&(\mathsf{rec}_0(X, \mathsf{inl}_X, \mathsf{inr}_X) \circ \mathsf{inl}_0)a && \equiv \quad \text{(defn. of } \mathsf{rec}_0 \text{ and } \mathsf{inl}_0\text{)}\\
&((\lambda\alpha.\alpha(X, \mathsf{inl}_X, \mathsf{inr}_X)) \circ (\lambda a.\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\,\mathsf{inl}_X\, a))a && \equiv \quad \text{(function application)}\\
&(\lambda\alpha.\alpha(X, \mathsf{inl}_X, \mathsf{inr}_X))(\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\,\mathsf{inl}_X\, a) && \equiv \quad \text{(function application)}\\
&(\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\,\mathsf{inl}_X\, a)(X, \mathsf{inl}_X, \mathsf{inr}_X) && \equiv \quad \text{(function application)}\\
&\mathsf{inl}_X\, a
\end{aligned}$$

Similarly for $b : B$. □

Being a weak coproduct is not enough for our purposes. Remember that we want our inductive types to satisfy the uniqueness principle as well, known as the propositional $\eta$-rule:

**Definition 1.4.** *A (strict) coproduct of $A$ and $B$ is a weak coproduct $(I, \mathsf{inl}_I, \mathsf{inr}_I) : \mathcal{U}_{A+B}$ with a recursor $\mathsf{rec}_I$ such that, for every $X : \mathcal{U}$ and $f : I \to X$, $f \sim \mathsf{rec}_I(X, f \circ \mathsf{inl}_I, f \circ \mathsf{inr}_I)$.*

Our end goal is to construct a strict coproduct of $A$ and $B$.

But one may wonder: what about dependent functions? After all, we are working within a dependent type theory. The eliminator seen so far (the recursor) is only capable of generating non-dependent morphisms.

The dependent eliminator (the induction principle) of a coproduct $A+B$ should generate, given a type family $Y : A+B \to \mathcal{U}$ together with appropriate constructors, a dependent function $\prod_{x:A+B} Y(x)$. How does the type of a constructor for a type family look like? We need—in the left case—, for every $a : A$, an element $x$ of $A + B$, with which to choose a fiber $Y(x)$. The only canonical choice is to use the injections of $A + B$ itself. Let us generalize:

**Definition 1.5.** *Given $(X, \mathsf{inl}_X, \mathsf{inr}_X) : \mathcal{U}_{A+B}$, a dependent algebra over $(X, \mathsf{inl}_X, \mathsf{inr}_X)$ is a type family $Y : X \to \mathcal{U}$ together with constructors $\mathsf{inl}_Y : \prod_{a:A} Y(\mathsf{inl}_X\, a)$ and $\mathsf{inr}_Y : \prod_{b:B} Y(\mathsf{inr}_X\, b)$. We write*

$$\mathcal{U}_{A+B}^{(X,\mathsf{inl}_X,\mathsf{inr}_X)} :\equiv \sum_{Y:X\to\mathcal{U}} \left(\prod_{a:A} Y(\mathsf{inl}_X\, a)\right) \times \left(\prod_{b:B} Y(\mathsf{inr}_X\, b)\right)$$

*for the type of all such dependent algebras.*

Our choice of typing for the constructors of a dependent algebra is the only one that allows to define a dependent morphism in the following manner:

**Definition 1.6.** *A dependent morphism between algebras $(X, \mathsf{inl}_X, \mathsf{inr}_X) : \mathcal{U}_{A+B}$ and $(Y, \mathsf{inl}_Y, \mathsf{inr}_Y) : \mathcal{U}_{A+B}^{(X,\mathsf{inl}_X,\mathsf{inr}_X)}$ is a function $f : \prod_{x:X} Y(x)$ such that $f \circ \mathsf{inl}_X \sim \mathsf{inl}_Y$ and $f \circ \mathsf{inr}_X \sim \mathsf{inr}_Y$.*

When appropriate, we will write $f : \prod_{(X,\mathsf{inl}_X,\mathsf{inr}_X)}(Y, \mathsf{inl}_Y, \mathsf{inr}_Y)$ whenever $f$ is a dependent morphism between $(X, \mathsf{inl}_X, \mathsf{inr}_X) : \mathcal{U}_{A+B}$ and $(Y, \mathsf{inl}_Y, \mathsf{inr}_Y) : \mathcal{U}_{A+B}^{(X,\mathsf{inl}_X,\mathsf{inr}_X)}$ (i.e. such that $f \circ \mathsf{inl}_X \sim \mathsf{inl}_Y$ and $f \circ \mathsf{inr}_X \sim \mathsf{inr}_Y$).

Observe that a normal algebra $(Y, \mathsf{inl}_Y, \mathsf{inr}_Y)$ can be defined as a dependent algebra over the final algebra $(\mathbb{1}, \lambda a.\star, \lambda b.\star)$, and taking the type family to be constant on $Y$. Following this point of view, a non-dependent morphism $f : (X, \mathsf{inl}_X, \mathsf{inr}_X) \to (Y, \mathsf{inl}_Y, \mathsf{inr}_Y)$ is just a dependent one from $(X, \mathsf{inl}_X, \mathsf{inr}_X)$ to a dependent algebra defined in this way.

Finally, an induction principle for a type $(I, \mathsf{inl}_I, \mathsf{inr}_I)$ can be written as:

$$\mathsf{ind}_I : \prod_{(X,\mathsf{inl}_X,\mathsf{inr}_X):\mathcal{U}_{A+B}^{(I,\mathsf{inl}_I,\mathsf{inr}_I)}} \prod_{i:I} X(i)$$

such that $\mathsf{ind}_I(X, \mathsf{inl}_X, \mathsf{inr}_X)$ satisfies the propositional $\beta$-rules, or, equivalently, it is a dependent morphism.

Naturally, we desire our inductive type to also have an induction principle. Luckily for us, this comes for free with every strict coproduct:

**Theorem 1.4.** *Let $(I, \mathsf{inl}_I, \mathsf{inr}_I) : \mathcal{U}_{A+B}$. Then, the following are equivalent:*

1. *$(I, \mathsf{inl}_I, \mathsf{inr}_I)$ has a dependent eliminator $\mathsf{ind}_I$ that satisfies the $\beta$-equalities.*

2. *$(I, \mathsf{inl}_I, \mathsf{inr}_I)$ has a non-dependent eliminator $\mathsf{rec}_I$ that satisfies the $\beta$-equalities, and for every $(X, \mathsf{inl}_X, \mathsf{inr}_X) : \mathcal{U}_{A+B}$ there is a unique morphism $(I, \mathsf{inl}_I, \mathsf{inr}_I) \to (X, \mathsf{inl}_X, \mathsf{inr}_X)$.*

3. *$(I, \mathsf{inl}_I, \mathsf{inr}_I)$ has a non-dependent eliminator $\mathsf{rec}_I$ that satisfies the $\beta$-equalities, as well as the propositional $\eta$-equality (i.e. $(I, \mathsf{inl}_I, \mathsf{inr}_I)$ is a coproduct).*

*Proof.* For (1) $\implies$ (2), first take $\mathsf{rec}_I(X, \mathsf{inl}_X, \mathsf{inr}_X) :\equiv \mathsf{ind}_I(\lambda i.X, \mathsf{inl}_X, \mathsf{inr}_X)$; the $\beta$-rules are the same. Assume $f, g : (I, \mathsf{inl}_I, \mathsf{inr}_I) \to (X, \mathsf{inl}_X, \mathsf{inr}_X)$. We will use the dependent eliminator $\mathsf{ind}_I$ to prove that $f \sim g$. By definition, this means proving $\prod_{i:I} fi = gi$. If we define a family $X : I \to \mathcal{U}$ as $I(i) :\equiv fi = gi$, then we just need to see that $C(i)$ is inhabited for every $i : I$. Observe again the type of $\mathsf{ind}_I$:

$$\prod_{(X,\mathsf{inl}_X,\mathsf{inr}_X):\mathcal{U}_{A+B}^{(I,\mathsf{inl}_I,\mathsf{inr}_I)}} \prod_{i:I} X(i)$$

If we can provide $\mathsf{inl}_X : \prod_{a:A} X(\mathsf{inl}_I\, a)$ and $\prod_{b:B} X(\mathsf{inr}_I\, b)$, then we can supply $(X, \mathsf{inl}_X, \mathsf{inr}_X)$ to $\mathsf{ind}_I$ to obtain a term of $\prod_{i:I} X(i)$, exactly as desired.

For $\prod_{a:A} X(\mathsf{inl}_I\, a)$, observe that this means:

$$\prod_{a:A} f(\mathsf{inl}_I\, a) = g(\mathsf{inl}_I\, a)$$

But, because of $f$ and $g$ are morphisms, $f(\mathsf{inl}_I\, a) = \mathsf{inl}_X\, a = g(\mathsf{inl}_I\, a)$. A similar reasoning can be applied to $b : B$.

For (2) $\implies$ (3), we just set $\mathsf{rec}_I(X, \mathsf{inl}_X, \mathsf{inr}_X)$ to be the function underlying such unique morphism, and the $\beta$-equalities are satisfied as part of the morphism. Now, we prove the $\eta$-equality for such $\mathsf{rec}_I$. Let $f : I \to X$ and consider $g :\equiv \mathsf{rec}_I(X, f \circ \mathsf{inl}_I, f \circ \mathsf{inr}_I) : I \to X$. Because $g$ is a morphism, $g \circ \mathsf{inl}_I \sim f \circ \mathsf{inl}_I$ and $g \circ \mathsf{inr}_I \sim f \circ \mathsf{inr}_I$, and because $g$ is the *unique* function with these properties, and $f$ also satisfies them ($f \circ \mathsf{inl}_I \sim f \circ \mathsf{inl}_I$ and $f \circ \mathsf{inr}_I \sim f \circ \mathsf{inr}_I$), then $f \sim g$.

(3) $\implies$ (1) is more complex; a proof is seen on [12, Theorem 2.3.1], and the general theory around this fact is developed in [3] and [2]. □

These characterizations will be key to all developments in this thesis: instead of proving an induction principle, which can be cumbersome (due to the presence of dependent algebras and morphisms), we just prove the recursion principle and then the $\eta$-rule, or the recursion principle and the uniqueness of the morphisms. With this in mind, we venture to see how we can obtain either of these conditions.

# Chapter 2

# Inductive types in $\mathsf{Set}$

## 2.1 Naturality

The starting point for this thesis is the paper by Awodey, Frey, and Speight [1] (or see [12] for a comprehensive explanation), in which they attempt to encode *strict* inductive types à la System F in a type system like the one described in Section 1.3.

In their approach, inductive types are defined as a particular refinement of the System F encoding. By refinement, it is meant a subtype (i.e. a $\Sigma$-type of the System F encoding together with a proposition on it). For instance, in the study case of coproducts, their base type is one equivalent to

$$A +_0 B :\equiv \prod_{(X, \mathsf{inl}_X, \mathsf{inr}_X) : \mathsf{Set}_{A+B}} X$$

where $\mathsf{Set}_{A+B} :\equiv \sum_{X:\mathsf{Set}} (A \to X) \times (B \to X)$, and $A$ and $B$ are also in $\mathsf{Set}$. One can observe that this is just the System F construction seen in Chapter 1, with the exception that all involved types are now in $\mathsf{Set}$ instead of the whole $\mathcal{U}$. The reasons for this restriction will be seen later on, but this is the main caveat of Awodey, Frey, and Speight's approach that we will try to overcome. For the rest of this chapter, we assume all the definitions of Section 1.4, replacing $\mathcal{U}$ for $\mathsf{Set}$.

As with the full $\mathcal{U}$ version, this type comes with canonical constructors:

$$\begin{aligned}
&\mathsf{inl}_0 : A \to A +_0 B \\
&\mathsf{inl}_0 :\equiv \lambda a. \lambda (X, \mathsf{inl}_X, \mathsf{inr}_X).\, \mathsf{inl}_X\, a \\
\\
&\mathsf{inr}_0 : B \to A +_0 B \\
&\mathsf{inr}_0 :\equiv \lambda b. \lambda (X, \mathsf{inl}_X, \mathsf{inr}_X).\, \mathsf{inr}_X\, a
\end{aligned}$$

And a recursor:

$$\begin{aligned}
&\mathsf{rec}_0 : \prod_{(X, \mathsf{inl}_X, \mathsf{inr}_X) : \mathsf{Set}_{A+B}} A +_0 B \to X \\
&\mathsf{rec}_0 :\equiv \lambda (X, \mathsf{inl}_X, \mathsf{inr}_X). \lambda \alpha. \alpha(X, \mathsf{inl}_X, \mathsf{inr}_X)
\end{aligned}$$

Then, they introduce the notion of **naturality** of an element $\alpha : A +_0 B$:

$$\mathrm{Nat}(\alpha) :\equiv \prod_{(X, \mathsf{inl}_X, \mathsf{inr}_X) : \mathsf{Set}_{A+B}} \prod_{Y : \mathsf{Set}} \prod_{f : X \to Y} f(\alpha(X, \mathsf{inl}_X, \mathsf{inr}_X)) = \alpha(Y, f \circ \mathsf{inl}_X, f \circ \mathsf{inr}_X)$$

This can be read as $\alpha$ being compatible with every function $f : X \to Y$. [1] and [12] offer an extensive justification of this construction.

From here, they build the refinement of $A +_0 B$ as its natural subtype, i.e.

$$A +_N B :\equiv \sum_{\alpha : A +_0 B} \mathrm{Nat}(\alpha)$$

Which also admits injections:

$$\begin{aligned}
&\mathsf{inl}_N : A \to A +_N B\\
&\mathsf{inl}_N :\equiv \lambda a.(\mathsf{inl}_0\, a, \lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\lambda Y.\lambda f.\mathsf{refl}_{f(\mathsf{inl}_X\, a)})
\end{aligned}$$

$$\begin{aligned}
&\mathsf{inr}_N : B \to A +_N B\\
&\mathsf{inr}_N :\equiv \lambda b.(\mathsf{inr}_0\, b, \lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\lambda Y.\lambda f.\mathsf{refl}_{f(\mathsf{inr}_X\, b)})
\end{aligned}$$

It is easy to check that the proofs of naturality of each constructor are well typed:

$$\begin{aligned}
&f((\mathsf{inl}_0\, a)(X, \mathsf{inl}_X, \mathsf{inr}_X)) &&\equiv &&\text{(definition of } \mathsf{inl}_0)\\
&f((\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\, \mathsf{inl}_X\, a)(X, \mathsf{inl}_X, \mathsf{inr}_X)) &&\equiv &&\text{(function application)}\\
&f(\mathsf{inl}_X\, a)
\end{aligned}$$

$$\begin{aligned}
&(\mathsf{inl}_0\, a)(X, f \circ \mathsf{inl}_X, f \circ \mathsf{inr}_X) &&\equiv &&\text{(definition of } \mathsf{inl}_0)\\
&(\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\, \mathsf{inl}_X\, a)(X, f \circ \mathsf{inl}_X, f \circ \mathsf{inr}_X) &&\equiv &&\text{(function application)}\\
&f(\mathsf{inl}_X\, a)
\end{aligned}$$

And it also admits a recursor, which just uses the one for the base type:

$$\begin{aligned}
&\mathsf{rec}_N : \prod_{(X, \mathsf{inl}_X, \mathsf{inr}_X) : \mathsf{Set}_{A+B}} A +_N B \to X\\
&\mathsf{rec}_N :\equiv \lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\lambda(\alpha, n).\, \mathsf{rec}_0(X, \mathsf{inl}_X, \mathsf{inr}_X)\alpha
\end{aligned}$$

Or, equivalently, $\mathsf{rec}_N \equiv \lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\lambda(\alpha, n).\alpha(X, \mathsf{inl}_X, \mathsf{inr}_X)$.

**Proposition 2.1.** $(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N)$ *together with* $\mathsf{rec}_N$ *is a weak coproduct (with respect to* $\mathsf{Set}$*).*

*Proof.* It is sufficient to prove $(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N)$ satisfies the $\beta$-equalities $\mathsf{rec}_N(X, \mathsf{inl}_X, \mathsf{inr}_X)(\mathsf{inl}_N\, a) = \mathsf{inl}_X\, a$ and $\mathsf{rec}_N(X, \mathsf{inl}_X, \mathsf{inr}_X)(\mathsf{inr}_N\, b) = \mathsf{inr}_X\, b$ for every $(X, \mathsf{inl}_X, \mathsf{inr}_X) : \mathsf{Set}_{A+B}$, $a : A$, and $b : B$.

$$\begin{aligned}
&\mathsf{rec}_N(X, \mathsf{inl}_X, \mathsf{inr}_X)(\mathsf{inl}_N\, a) &&\equiv &&\text{(definition of } \mathsf{rec}_N)\\
&(\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\lambda(\alpha, n).\, \mathsf{rec}_0(X, \mathsf{inl}_X, \mathsf{inr}_X)\alpha)\\
&\quad (X, \mathsf{inl}_X, \mathsf{inr}_X)(\mathsf{inl}_N\, a) &&\equiv &&\text{(function application)}\\
&(\lambda(\alpha, n).\, \mathsf{rec}_0(X, \mathsf{inl}_X, \mathsf{inr}_X)\alpha)(\mathsf{inl}_N\, a) &&\equiv &&\text{(definition of } \mathsf{inl}_N)\\
&(\lambda(\alpha, n).\, \mathsf{rec}_0(X, \mathsf{inl}_X, \mathsf{inr}_X)\alpha)(\mathsf{inl}_0\, a, \dots) &&\equiv &&\text{(function application)}\\
&\mathsf{rec}_0(X, \mathsf{inl}_X, \mathsf{inr}_X)(\mathsf{inl}_0\, a) &&\equiv &&\text{(definition of } \mathsf{rec}_0)\\
&(\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\alpha.\alpha(X, \mathsf{inl}_X, \mathsf{inr}_X))\\
&\quad (X, \mathsf{inl}_X, \mathsf{inr}_X)(\mathsf{inl}_0\, a) &&\equiv &&\text{(function application)}\\
&(\alpha.\alpha(X, \mathsf{inl}_X, \mathsf{inr}_X))(\mathsf{inl}_0\, a) &&\equiv &&\text{(function application)}\\
&(\mathsf{inl}_0\, a)(X, \mathsf{inl}_X, \mathsf{inr}_X) &&\equiv &&\text{(defintion of } \mathsf{inl}_0)\\
&(\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\, \mathsf{inl}_X\, a)(X, \mathsf{inl}_X, \mathsf{inr}_X) &&\equiv &&\text{(function application)}\\
&\mathsf{inl}_X\, a
\end{aligned}$$

And analogously for $b : B$. $\square$

**Lemma 2.2.** *$\mathrm{Nat}(\alpha)$ is a $\mathsf{Prop}$ for every $\alpha : A +_0 B$.*

*Proof.* By definition:

$$\mathrm{Nat}(\alpha) :\equiv \prod_{(X,\mathsf{inl}_X,\mathsf{inr}_X):\mathsf{Set}_{A+B}} \prod_{Y:\mathsf{Set}} \prod_{f:X\to Y} f(\alpha(X,\mathsf{inl}_X,\mathsf{inr}_X)) = \alpha(Y, f\circ \mathsf{inl}_X, f\circ \mathsf{inr}_X)$$

The codomain of both sides of the equality is $Y$, a $\mathsf{Set}$. Hence the equality itself is a $\mathsf{Prop}$. The result is obtained by triple application of Lemma 1.2. $\square$

With this lemma, it can be proved:

**Theorem 2.3** (Awodey, Frey, and Speight)**.** *$(A+_N B, \mathsf{inl}_N, \mathsf{inr}_N)$ together with $\mathsf{rec}_N$ is a strict coproduct (with respect to $\mathsf{Set}$).*

*Proof.* We already know from Proposition 2.1 that it is a weak coproduct. We have to prove that $(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N)$ satisfies the propositional $\eta$-rule: $\mathsf{rec}_N(X, f \circ \mathsf{inl}_N, f \circ \mathsf{inr}_N) \sim f$ for every $X : \mathsf{Set}$ and $f : A +_N B \to X$. To this end, one first proves that $\mathsf{rec}_N(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N) \sim \mathsf{id}_{A+_N B}$.

$$\begin{aligned}
&\mathsf{rec}_N(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N)(\alpha, n) && \equiv && \text{(definition of } \mathsf{rec}_N) \\
&\mathsf{rec}_0(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N)\alpha && \equiv && \text{(definition of } \mathsf{rec}_0) \\
&\alpha(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N)
\end{aligned}$$

We must then see whether $\alpha(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N) = (\alpha, n)$ for every $\alpha : A +_0 B$ and $n : \mathrm{Nat}(\alpha)$. Using Lemma 1.1 and Lemma 2.2, it is sufficient to prove equality of the first coordinate of the pair. For that, remember that both $\mathsf{pr}_1(\alpha(A+_N B, \mathsf{inl}_N, \mathsf{inr}_N))$ and $\alpha$ belong to $A +_0 B :\equiv \prod_{(X,\mathsf{inl}_X,\mathsf{inr}_X):\mathsf{Set}_{A+B}} X$, which is a function type, so by function extensionality we can just prove $\mathsf{pr}_1(\alpha(A+_N B, \mathsf{inl}_N, \mathsf{inr}_N))(X, \mathsf{inl}_X, \mathsf{inr}_X) = \alpha(X, \mathsf{inl}_X, \mathsf{inr}_X)$ for every $(X, \mathsf{inl}_X, \mathsf{inr}_X) : \mathsf{Set}_{A+B}$.

$$\begin{aligned}
&\mathsf{pr}_1(\alpha(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N))(X, \mathsf{inl}_X, \mathsf{inr}_X) && \equiv && \text{(definition of } \mathsf{rec}_N) \\
&\mathsf{rec}_N(X, \mathsf{inl}_X, \mathsf{inr}_X)(\alpha(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N)) && = && (\mathsf{pr}_2(\alpha(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N))) \\
&\alpha(X, \mathsf{rec}_N(X, \mathsf{inl}_X, \mathsf{inr}_X) \circ \mathsf{inl}_N, \\
&\quad \mathsf{rec}_N(X, \mathsf{inl}_X, \mathsf{inr}_X) \circ \mathsf{inr}_N) && = && (\beta\text{-equalities}) \\
&\alpha(X, \mathsf{inl}_X, \mathsf{inr}_X)
\end{aligned}$$

From the fact that $\mathsf{rec}_N(A+_N B, \mathsf{inl}_N, \mathsf{inr}_N) \sim \mathsf{id}_{A+_N B}$ and using function extensionality again, we derive the desired, more general result $\mathsf{rec}_N(X, f\circ\mathsf{inl}_N, f\circ\mathsf{inr}_N) \sim f$:

$$\begin{aligned}
&\mathsf{rec}_N(X, f \circ \mathsf{inl}_N, f \circ \mathsf{inr}_N)(\alpha, n) && \equiv && \text{(definition of } \mathsf{rec}_N) \\
&\alpha(X, f \circ \mathsf{inl}_N, f \circ \mathsf{inr}_N) && = && (n) \\
&f(\alpha(X, \mathsf{inl}_N, \mathsf{inr}_N)) && \equiv && \text{(defintion of } \mathsf{rec}_N) \\
&f(\mathsf{rec}_N(X, \mathsf{inl}_N, \mathsf{inr}_N)(\alpha, n)) && = && (\mathsf{rec}_N(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N) \sim \mathsf{id}_{A+_N B}) \\
&f(\alpha, n)
\end{aligned}$$

$\square$

As seen in the proof, the restriction to $\mathsf{Set}$ serves two purposes: on the one hand, to establish the equality between two elements of $A +_N B$ that agree on the first component; and, on the other, so that the subtype $A +_N B$ stays in $\mathsf{Set}$, and hence we can use impredicativity to evaluate $\mathsf{rec}_N$ with $(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N)$.

## 2.2 Inductivity

Our first attempt to improve on Awodey et al.'s result is to replace this naturality principle for another condition that we call inductivity. Our hope is that this new property allows us to forgo the h-level restriction that Nat has. On the other hand, inductivity is encoded as a very natural property to define given the signature of the inductive type that we desire to construct.

**Definition 2.1.** *We define **inductivity** for a term $\alpha : A +_0 B$ as:*

$$\mathrm{Ind}(\alpha) :\equiv \prod_{(C,\mathsf{inl}_C,\mathsf{inr}_C):\sum_{C:A+_0B\to\mathsf{Prop}}(\prod_{a:A} C(\mathsf{inl}_0\, a))\times(\prod_{b:B} C(\mathsf{inr}_0\, b))} C(\alpha)$$

The shape of the type under the $\Pi$ symbol is almost identical to the dependent algebras defined in the previous chapter, with the subtle difference that we eliminate into types in $\mathsf{Prop}$ instead of $\mathcal{U}$ (or the expected $\mathsf{Set}$). This is a technical requirement for the coming proofs. Nevertheless, this is not a real downgrade, as from it we will be able to obtain the $\eta$-rule which is equivalent to the full induction principle into $\mathsf{Set}$.

Compare also the a term of this type to one of $A+_0B$: the latter takes an algebra and returns an element of it, the former takes a *dependent* algebra and returns an element of it. More specifically, $\mathrm{Ind}(\alpha)$ can be read as: "given a proposition $C$ over $A+_0B$, if it can be proven for every $\mathsf{inl}_0\, a$ and every $\mathsf{inr}_0\, b$, then it can be proven for $\alpha$". Indeed, this is what makes this type inductive: to prove a statement about all its elements, it suffices to prove it for every canonical element. In fact, in the same way one defines $\mathsf{rec}(X,\mathsf{inl}_X,\mathsf{inr}_X)$ as $\lambda\alpha : A+_0B.\alpha(X,\mathsf{inl}_X,\mathsf{inr}_X)$, one could imagine the induction principle $\mathsf{ind}(C,\mathsf{inl}_C,\mathsf{inr}_C)$ for a dependent algebra $(C,\mathsf{inl}_C,\mathsf{inr}_C)$ as $\lambda i : \mathrm{Ind}(\alpha).i(C,\mathsf{inl}_C,\mathsf{inr}_C)$.

Our candidate to a coproduct in $\mathsf{Set}$ of $A$ and $B$ is then

$$A +_1 B :\equiv \sum_{\alpha:A+_0B} \mathrm{Ind}(\alpha)$$

with the inherited constructors

$$\begin{aligned}&\mathsf{inl}_1 : A \to A +_1 B\\ &\mathsf{inl}_1 :\equiv \lambda a.(\mathsf{inl}_0\, a, \lambda(C,\mathsf{inl}_C,\mathsf{inr}_C).\, \mathsf{inl}_C\, a)\end{aligned}$$

$$\begin{aligned}&\mathsf{inr}_1 : B \to A +_1 B\\ &\mathsf{inr}_1 :\equiv \lambda b.(\mathsf{inr}_0\, b, \lambda(C,\mathsf{inl}_C,\mathsf{inr}_C).\, \mathsf{inr}_C\, b)\end{aligned}$$

These constructors, similarly to those of $A+_N B$, are just the System F constructors $\mathsf{inl}_0$ and $\mathsf{inr}_0$ accompanied by canonical witnesses that they are always inductive. And, as with the previous examples, we can define a recursor:

$$\begin{aligned}&\mathsf{rec}_1 : \prod_{(X,\mathsf{inl}_X,\mathsf{inr}_X):\mathsf{Set}_{A+B}} A +_1 B \to X\\ &\mathsf{rec}_1 :\equiv \lambda(X,\mathsf{inl}_X,\mathsf{inr}_X).\lambda(\alpha,i).\, \mathsf{rec}_0(X,\mathsf{inl}_X,\mathsf{inr}_X)\alpha\end{aligned}$$

This does the exact same thing as $\mathsf{rec}_N$: drop the proof of inductivity, and apply $\mathsf{rec}_0$.

**Proposition 2.4.** $(A +_1 B, \mathsf{inl}_1, \mathsf{inr}_1)$ *together with* $\mathsf{rec}_1$ *is a weak coproduct.*

*Proof.* We prove the statement for $\mathsf{inl}_1$, up to definitional equality. The one for $\mathsf{inr}_1$ is totally analogous. Given any $a : A$:

$$
\begin{array}{lcr}
\mathsf{rec}_1(X, \mathsf{inl}_X, \mathsf{inr}_X)(\mathsf{inl}_1\, a) & \equiv & \text{(definition of } \mathsf{rec}_1\text{)} \\
(\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\lambda(\alpha, i).\, \mathsf{rec}_0(X, \mathsf{inl}_X, \mathsf{inr}_X)\alpha) & & \\
\quad (X, \mathsf{inl}_X, \mathsf{inr}_X)(\mathsf{inl}_1\, a) & \equiv & \text{(function application)} \\
(\lambda(\alpha, i).\, \mathsf{rec}_0(X, \mathsf{inl}_X, \mathsf{inr}_X)\alpha)(\mathsf{inl}_1\, a) & \equiv & \text{(definition of } \mathsf{inl}_1\text{)} \\
(\lambda(\alpha, i).\, \mathsf{rec}_0(X, \mathsf{inl}_X, \mathsf{inr}_X)\alpha)(\mathsf{inl}_0\, a, \dots) & \equiv & \text{(function application)} \\
\mathsf{rec}_0(X, \mathsf{inl}_X, \mathsf{inr}_X)(\mathsf{inl}_0\, a) & \equiv & \text{(definition of } \mathsf{rec}_0\text{)} \\
(\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\alpha.\alpha(X, \mathsf{inl}_X, \mathsf{inr}_X)) & & \\
\quad (X, \mathsf{inl}_X, \mathsf{inr}_X)(\mathsf{inl}_0\, a) & \equiv & \text{(function application)} \\
(\alpha.\alpha(X, \mathsf{inl}_X, \mathsf{inr}_X))(\mathsf{inl}_0\, a) & \equiv & \text{(function application)} \\
(\mathsf{inl}_0\, a)(X, \mathsf{inl}_X, \mathsf{inr}_X) & \equiv & \text{(defintion of } \mathsf{inl}_0\text{)} \\
(\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\, \mathsf{inl}_X\, a)(X, \mathsf{inl}_X, \mathsf{inr}_X) & \equiv & \text{(function application)} \\
\mathsf{inl}_X\, a & &
\end{array}
$$

□

**Lemma 2.5.** $\mathrm{Ind}(\alpha)$ *is a* $\mathsf{Prop}$ *for every* $\alpha : A +_0 B$.

*Proof.* Remember that:

$$\mathrm{Ind}(\alpha) \equiv \prod_{(C, \mathsf{inl}_C, \mathsf{inr}_C) : \sum_{C : A +_0 B \to \mathsf{Prop}} (\prod_{a:A} C(\mathsf{inl}_0\, a)) \times (\prod_{b:B} C(\mathsf{inr}_0\, b))} C(\alpha)$$

We imposed that $C : A +_0 B \to \mathsf{Prop}$, so $C(\alpha) : \mathsf{Prop}$. Again by Lemma 1.2, the whole type $\mathrm{Ind}(\alpha)$ is also a $\mathsf{Prop}$. □

**Theorem 2.6.** $(A +_1 B, \mathsf{inl}_1, \mathsf{inr}_1)$ *together with* $\mathsf{rec}_1$ *is the coproduct of the sets* $A$ *and* $B$ *with respect to other sets in* $\mathcal{U}$.

*Proof.* Let $h, k : (A +_1 B, \mathsf{inl}_1, \mathsf{inr}_1) \to (X, \mathsf{inl}_X, \mathsf{inr}_X)$, we want to see $h \sim k$. By function extensionality, we just need to see $\prod_{(\alpha,i):A+_1B} h(\alpha, i) = k(\alpha, i)$. Equivalently, because $A +_1 B \equiv \sum_{\alpha:A+_0B} \mathrm{Ind}(\alpha)$, we can rewrite our goal as $\prod_{\alpha:A+_0B} \prod_{i:\mathrm{Ind}(\alpha)} h(\alpha, i) = k(\alpha, i)$.

Our notion of induction works on families of propositions over $A +_0 B$, but we want to prove a statement for all $\alpha : A +_0 B$ *and* all $i : \mathrm{Ind}(\alpha)$. This is one of the main cruxes of our approach: we define inductivity over a type, but we want to use induction over that type *plus* its proof of inductivity. A priori this problem might seem insurmountable, but there is a possible fix that makes use of impredicativity and the fact that $\mathrm{Ind}(\alpha)$ is a $\mathsf{Prop}$ (Lemma 2.5). Let us start by fixing arbitrary $\alpha : A +_0 B$ and $i : \mathrm{Ind}(\alpha)$ for the rest of the proof.

Define then a type family $C$ over $A +_0 B$ as $C(\beta) :\equiv \prod_{j:\mathrm{Ind}(\beta)} h(\beta, j) = k(\beta, j)$. As $X$ is a $\mathsf{Set}$, all equality types between elements of $X$ are propositions in $\mathcal{U}$. In particular, $h(\beta, j) = k(\beta, j)$ is always a $\mathsf{Prop}$, hence (by Lemma 1.2) so is $C(\beta) \equiv \prod_{j:\mathrm{Ind}(\beta)} h(\beta, j) = k(\beta, j)$. So $C$ has type $A +_0 B \to \mathsf{Prop}$. Our goal is to obtain a term $c : C(\alpha)$, from which we can get $ci : h(\alpha, i) = k(\alpha, i)$ which is our ultimate objective.

Now we can finally apply induction: we want to prove $C$ for a given $\alpha$, for which we have a witness of inductivity $i : \mathrm{Ind}(\alpha)$. We are only missing the proofs of $C(\mathsf{inl}_0\, a)$ and $C(\mathsf{inr}_0\, b)$ for arbitrary $a : A$, $b : B$. Let us do the case for $A$, the other is analogous.

We want to see $\prod_{j:\mathrm{Ind}(\mathsf{inl}_0\, a)} h(\mathsf{inl}_0\, a, j) = k(\mathsf{inl}_0\, a, j)$. We already have a similar result, namely that $\sum_{j:\mathrm{Ind}(\mathsf{inl}_0\, a)} h(\mathsf{inl}_0\, a, j) = k(\mathsf{inl}_0\, a, j)$, with $j :\equiv \lambda(C, \mathsf{inl}_C, \mathsf{inr}_C)$. $\mathsf{inl}_C\, a$, because with this choice $(\mathsf{inl}_0\, a, j) \equiv \mathsf{inl}_1\, a$. But then $h(\mathsf{inl}_0\, a, j) = k(\mathsf{inl}_0\, a, j) \equiv h(\mathsf{inl}_1\, a) = k(\mathsf{inl}_1\, a)$, and each of $h$ and $k$, when composed with $\mathsf{inl}_1$, are equal to $\mathsf{inl}_X$ by hypothesis.

The final piece is to observe that, for any family of types $C$ over a proposition $P$, $\sum_{p:P} C(p)$ implies $\prod_{p:P} C(p)$: given $(p, c) : \sum_{p:P} C(p)$ and $p' : P$, we can construct an element of $C(p')$ as $\mathsf{transport}^C(q, c)$, where $q : p = p'$ by virtue of $P$ being a proposition. Hence, our proof of $\sum_{j:\mathrm{Ind}(\mathsf{inl}_0\, a)} h(\mathsf{inl}_0\, a, j) = k(\mathsf{inl}_0\, a, j)$ gives us a proof of $\prod_{j:\mathrm{Ind}(\mathsf{inl}_0\, a)} h(\mathsf{inl}_0\, a, j) = k(\mathsf{inl}_0\, a, j)$, as wanted, because Ind is a proposition. $\square$

As with naturality, the proof for inductivity makes heavy use of specific properties of Set: in our case, on top of using it to apply impredicativity, we also need it to make certain equalities propositions so that we can then use the final trick of the proof.

## 2.3 W-types

The result given for coproducts can be generalized to virtually any inductive type, by constructing them as W-types.

W-types are a family of inductive types parametrized by two other types, $A : \mathcal{U}$ and $B : A \to \mathcal{U}$. The W-type on $A$ and $B$ is usually denoted by $\mathsf{W}_{a:A} B(a)$, although we will use the notation $\mathsf{W}_A B$ for simplicity's sake. A $\mathsf{W}_A B$-like algebra is a type $X : \mathsf{Set}$ together with a single constructor $\mathsf{sup} : (\sum_{a:A} B(a) \to X) \to X$.

**Definition 2.2.** *The type of $\mathsf{W}_A B$-like algebras is:*

$$\mathsf{Set}_{\mathsf{W}_A B} :\equiv \sum_{X:\mathsf{Set}} \left( \left( \sum_{a:A} B(a) \to X \right) \to X \right)$$

The idea is that, based on our choice of $A$ and $B$, we can encode any System F-style inductive type (and more). For instance, we can take $A :\equiv \mathbb{2}$ (the type of booleans $0_{\mathbb{2}}$ and $1_{\mathbb{2}}$) and $B(0_{\mathbb{2}}) :\equiv \mathbb{0}$, $B(1_{\mathbb{2}}) :\equiv \mathbb{1}$, which results in:

$$\begin{aligned} &\left( \sum_{a:A} B(a) \to X \right) \to X && \simeq \\ &((\mathbb{0} \to X) + (\mathbb{1} \to X)) \to X && \simeq \\ &(\mathbb{1} + X) \to X && \simeq \\ &(\mathbb{1} \to X) \times (X \to X) \end{aligned}$$

Which is the conjunction of the two the constructors for the naturals.

Each $a : A$ classifies the different constructors, and the corresponding $B(a)$ indexes the arguments for each constructor. W-types are also thought of as types of well-founded trees: each $a : A$ is a node of the tree, and the node labeled by $a$

has $B(a)$ subtrees. Given a subtree, we can construct its supremum, which is a new tree. See also [6, Section 5.3].

The proof for $\mathsf{W}$-types brings in more complexity, as they are, in general, proper inductive types (in the sense of Table 1.1), which means that we have to take into account an induction hypothesis in every proof by induction. Throughout the whole section we fix a type $A : \mathcal{U}$ in our impredicative universe $\mathcal{U}$, and a type family $B : A \to \mathcal{U}$ of $\mathcal{U}$-types over $A$. We also inherit, mutatis mutandis, the definitions of (strict) $\mathsf{W}$-type from those of (strict) coproduct—with the corresponding $\eta$- and $\beta$-rules that we will soon see.

We start, as always, by defining the System F-style base type:

$$\mathsf{W}^0_A B :\equiv \prod_{(X, \mathsf{sup}_X) : \mathsf{Set}_{\mathsf{W}_A B}} X$$

and its associated constructor:

$$\begin{aligned}
\mathsf{sup}_0 &: \left( \sum_{a:A} B(a) \to \mathsf{W}^0_A B \right) \to \mathsf{W}^0_A B \\
\mathsf{sup}_0 &:\equiv \lambda(a, f).\lambda(X, \mathsf{sup}_X).\, \mathsf{sup}_X(a, \lambda b. fb(X, \mathsf{sup}_X))
\end{aligned}$$

For convenience, we introduce the "reverse application" helper function $\mathsf{rev}\, x :\equiv \lambda f. fx$ for all suitably typed $f$ and $x$, such that $\mathsf{rev}\, x f \equiv fx$. This allows us to rewrite $\mathsf{sup}_0(a, f)(X, \mathsf{sup}_X)$ as $\mathsf{sup}_X(a, \mathsf{rev}(X, \mathsf{sup}_X) \circ f)$.

We also translate the concept of morphism to $\mathsf{W}$-types:

**Definition 2.3.** *A morphism between algebras $(X, \mathsf{sup}_X)$ and $(Y, \mathsf{sup}_Y)$ is a function $f : X \to Y$ such that $f(\mathsf{sup}_X(a, l) = \mathsf{sup}_Y(a, f \circ l)$ for every $a : A$, $l : B(a) \to X$. If $f$ is a morphism, we write $f : (X, \mathsf{sup}_X) \to (Y, \mathsf{sup}_Y)$.*

First, we need a bit of notational aid. For a family of propositions $C : \mathsf{W}^0_A B \to \mathsf{Prop}$, let us write $C^{\mathsf{W}_A B}(a, f) :\equiv \prod_{b:B(a)} C(fb)$. This makes it easier to define the inductivity of $w$ as

$$\mathrm{Ind}(w) :\equiv \prod_{(C, \mathsf{sup}_C) : \sum_{C : \mathsf{W}^0_A B \to \mathsf{Prop}} \left( \prod_{(a,f) : \sum_{a:A} f : B(a) \to \mathsf{W}^0_A B} C^{\mathsf{W}_A B}(a,f) \to C(\mathsf{sup}_0(a,f)) \right)} C(w)$$

As with the coproduct, we could refer to $(C, \mathsf{sup}_C)$ as a dependent algebra over $(\mathsf{W}^0_A B, \mathsf{sup}_0)$ and write

$$\mathsf{Set}^{(\mathsf{W}^0_A B, \mathsf{sup}_0)}_{\mathsf{W}_A B} :\equiv \sum_{C : \mathsf{W}^0_A B \to \mathsf{Prop}} \left( \prod_{(a,f) : \sum_{a:A} f : B(a) \to \mathsf{W}^0_A B} C^{\mathsf{W}_A B}(a, f) \to C(\mathsf{sup}_0(a, f)) \right)$$

and then

$$\mathrm{Ind}(w) \equiv \prod_{(C, \mathsf{sup}_C) : \mathsf{Set}^{(\mathsf{W}^0_A B, \mathsf{sup}_0)}_{\mathsf{W}_A B}} C(w)$$

The recursor is identical to the one for $A + B$:

$$\begin{aligned}
\mathsf{rec}_0 &: \prod_{(X, \mathsf{sup}_X) : \mathsf{Set}_{\mathsf{W}_A B}} \mathsf{W}^0_A B \to X \\
\mathsf{rec}_0 &:\equiv \lambda(X, \mathsf{sup}_X).\lambda w. w(X, \mathsf{sup}_X)
\end{aligned}$$

And once again we take the inductive subtype:

$$\mathsf{W}^1_A B :\equiv \sum_{w:\mathsf{W}^0_A B} \mathrm{Ind}(w)$$

This type can also be given an algebra structure, but this time it will require substantially more work.

**Proposition 2.7.** *There is a term* $\mathsf{sup}_1$ *of type* $\left(\sum_{a:A} B(a) \to \mathsf{W}^1_A B\right) \to \mathsf{W}^1_A B$.

*Proof.* Let $(a, f) : \sum_{a:A} B(a) \to \mathsf{W}^1_A B$. We want to find a term of type $\mathsf{W}^1_A B \equiv \sum_{w:\mathsf{W}^0_A B} \mathrm{Ind}(w)$, this is, a term $w : \mathsf{W}^0_A B$ and a term of type $\mathrm{Ind}(w)$. For $w$, we choose $\mathsf{sup}_0(a, \mathsf{pr}_1 \circ f)$. Now we need something of type $\mathrm{Ind}(\mathsf{sup}_0(a, \mathsf{pr}_1 \circ f))$.

This is, given a type family $C : \mathsf{W}^0_A B \to \mathsf{Prop}$, and a function

$$\mathsf{sup}_C : \prod_{(a,f):\sum_{a:A} B(a)\to \mathsf{W}^0_A B} C^{\mathsf{W}_A B}(a, f) \to C(\mathsf{sup}_0(a, f))$$

we need to obtain a term of type $C(\mathsf{sup}_0(a, \mathsf{pr}_1 \circ f))$. By choosing $a$ and $\mathsf{pr}_1 \circ f$ as the first two arguments to $\mathsf{sup}_C$, we are left with finding a term of $C^{\mathsf{W}_A B}(a, \mathsf{pr}_1 \circ f) \equiv \prod_{b:B(a)} C(\mathsf{pr}_1(fb))$ as our last requirement.

Let $b : B(a)$ arbitrary, then $fb : \mathsf{W}^1_A B$ and $\mathsf{pr}_2(fb) : \mathrm{Ind}(\mathsf{pr}_1(fb))$. Observe that we now have the following situation: we are looking for a term of type $C(\mathsf{pr}_1(fb))$ and we have a witness $\mathsf{pr}_2(fb)$ of the property $\mathrm{Ind}(\mathsf{pr}_1(fb))$. So, we can just apply $(C, \mathsf{sup}_C)$ to $\mathsf{pr}_2(fb)$ to obtain the desired result.

$$\begin{aligned}
&\mathsf{sup}_1 : \left(\sum_{a:A} B(a) \to \mathsf{W}^1_A B\right) \to \mathsf{W}^1_A B \\
&\mathsf{sup}_1 :\equiv \lambda(a, f).( \\
&\qquad \mathsf{sup}_0(a, \mathsf{pr}_1 \circ f), \\
&\qquad \lambda(C, \mathsf{sup}_C).\, \mathsf{sup}_C(a, (\mathsf{pr}_1 \circ f))(\lambda b.\, \mathsf{pr}_2(fb)(C, \mathsf{sup}_C)) \\
&\quad )
\end{aligned}$$

□

**Lemma 2.8.** $\mathrm{Ind}(\alpha)$ *is a* $\mathsf{Prop}$ *for every* $\alpha : A +_0 B$.

*Proof.* As in the proof for Lemma 2.5, we just need to observe that $C(\alpha) : \mathsf{Prop}$ and use Lemma 1.2. □

**Lemma 2.9.** *Given* $a : A$, $f : B(a) \to \mathsf{W}^0_A B$, *and a proof of inductivity of each* $fb$, *i.e.* $g : \prod_{b:B(a)} \mathrm{Ind}(fb)$, *there is a proof of the inductivity of* $\mathsf{sup}_0(a, f)$. *In other words, there exists a function* $\mathsf{sup}_{\mathrm{Ind}} : \prod_{(a,f):\sum_{a:A} B(a)\to \mathsf{W}^0_A B} \mathrm{Ind}^{\mathsf{W}_A B}(a, f) \to \mathrm{Ind}(\mathsf{sup}_0(a, f))$.

*Proof.* Let $a$, $f$, and $g$ be the hypotheses as described above. We need to find:

$$\mathrm{Ind}(\mathsf{sup}_0(a, f)) \equiv \prod_{(C,\mathsf{sup}_C):\mathsf{Set}^{(\mathsf{W}^0_A B,\mathsf{sup}_0)}_{\mathsf{W}_A B}} C(\mathsf{sup}_0(a, f))$$

Take hence $C$ and $\mathsf{sup}_C$ as the bound variables, with types:

$$\begin{aligned} C &: \mathsf{W}^0_A B \to \mathsf{Prop} \\ \mathsf{sup}_C &: \prod_{(a',f'):\sum_{a':A} B(a')\to \mathsf{W}^0_A B} C^{\mathsf{W}_A B}(a',f') \to C(\mathsf{sup}_0(a',f')) \end{aligned}$$

and let us find a proof of $C(\mathsf{sup}_0(a, f))$. Observe that this is the return type of $\mathsf{sup}_C$, given that we set $(a', f') :\equiv (a, f)$, and we provide a proof that $C(fb)$ holds for every $b : B(a)$.

This proof can be constructed by using $gb : \mathrm{Ind}(fb)$. A term of type $\mathrm{Ind}(fb)$ takes a type family (such as $C$), and then a proof that if the property $C$ holds for every branch of the W-type, then it holds for the supremum as well. But that is exactly what $\mathsf{sup}_C$ is, so $gb(C, \mathsf{sup}_C) : C(fb)$. Therefore we can construct $\mathsf{sup}_{\mathrm{Ind}} :\equiv \lambda(a, f).\lambda g.\lambda(C, \mathsf{sup}_C).\, \mathsf{sup}_C(a, f)(\lambda b.gb(C, \mathsf{sup}_C))$, so that $\mathsf{sup}_{\mathrm{Ind}}(a, f)g : \mathrm{Ind}(\mathsf{sup}_0(a, f))$. □

We continue with the usual development: for every algebra $X$, we can construct a function from $\mathsf{W}^1_A B$ to it:

$$\begin{aligned} \mathsf{rec}_1 &: \prod_{(X,\mathsf{sup}_X):\mathsf{Set}_{\mathsf{W}_A B}} \mathsf{W}^1_A B \to X \\ \mathsf{rec}_1 &:\equiv \lambda(X, \mathsf{sup}_X).\lambda(w, i).\, \mathsf{rec}_0(X, \mathsf{sup}_X)w \end{aligned}$$

We also introduce the $\mathsf{W}_A B$-like algebra morphisms:

**Definition 2.4.** *A morphism between algebras $(X, \mathsf{sup}_X)$ and $(Y, \mathsf{sup}_Y)$ is a function $f : X \to Y$ such that $f \circ \mathsf{sup}_X(a, f) = \mathsf{sup}_Y(a, f \circ l)$. We write $(X, \mathsf{sup}_X) \to (Y, \mathsf{sup}_Y) :\equiv \sum_{f:X\to Y} f \circ \mathsf{sup}_X \sim \mathsf{sup}_Y \circ (\mathsf{id}_A \times (f \circ -))$.*

The higher-order function $\mathsf{id}_A \times (f \circ -)$ takes $(a, l)$ to $(a, f \circ l)$. It is technically a slight abuse of notation, as $(f \circ -) : (B(a) \to X) \to (B(a) \to Y)$ as a function depends on $a$; but it is convenient and cannot be accidentally mistyped.

We can visualize a morphism as a function $f$ together with a proof that the following diagram commutes (up to homotopy):

$$\begin{tikzcd} \sum_{a:A}(B(a) \to X) \arrow[r, "\mathsf{id}_A \times (f\circ -)"] \arrow[d, "\mathsf{sup}_X"'] & \sum_{a:A}(B(a) \to Y) \arrow[d, "\mathsf{sup}_Y"] \\ X \arrow[r, "f"'] & Y \end{tikzcd}$$

Finally, we prove that our candidate does the job of the W-type on $A$ and $B$. For this, we prove the analogous properties of a strict coproduct:

**Proposition 2.10.** $(\mathsf{W}^1_A B, \mathsf{sup}_1)$ *satisfies the definitional $\beta$-equality for $\mathsf{W}_A B$, namely, that for every $a : A$, $l : B(a) \to \mathsf{W}^1_A B$, and $(X, \mathsf{sup}_X) : \mathsf{Set}_{\mathsf{W}_A B}$:*

$$\mathsf{rec}_1(X, \mathsf{sup}_X)(\mathsf{sup}_1(a, l)) \equiv \mathsf{sup}_X(a, \mathsf{rec}_1(X, \mathsf{sup}_X) \circ l)$$

*Proof.* On the one hand:

$$\begin{aligned}
&(\mathsf{rec}_1(X, \mathsf{sup}_X) \circ \mathsf{sup}_1)(a, l) && \equiv && \text{(definition of } \mathsf{sup}_1\text{)}\\
&(\mathsf{rec}_1(X, \mathsf{sup}_X) \circ (\lambda(a, l).(\mathsf{sup}_0(a, \mathsf{pr}_1 \circ l), \dots)))(a, l) && \equiv && \text{(function application)}\\
&\mathsf{rec}_1(X, \mathsf{sup}_X)(\mathsf{sup}_0(a, \mathsf{pr}_1 \circ l), \dots) && \equiv && \text{(definition of } \mathsf{rec}_1\text{)}\\
&(\lambda(w, i).w(X, \mathsf{sup}_X))(\mathsf{sup}_0(a, \mathsf{pr}_1 \circ l), \dots) && \equiv && \text{(function application)}\\
&\mathsf{sup}_0(a, \mathsf{pr}_1 \circ l)(X, \mathsf{sup}_X) && \equiv && \text{(definition of } \mathsf{sup}_0\text{)}\\
&(\lambda(a', l').\lambda(X, \mathsf{sup}_X).\, \mathsf{sup}_X(a', \mathsf{rev}(X, \mathsf{sup}_X) \circ l'))\\
&\quad (a, \mathsf{pr}_1 \circ l)(X, \mathsf{sup}_X) && \equiv && \text{(function application)}\\
&\mathsf{sup}_X(a, \mathsf{rev}(X, \mathsf{sup}_X) \circ \mathsf{pr}_1 \circ l)
\end{aligned}$$

On the other hand:

$$\begin{aligned}
&\mathsf{sup}_X((\mathsf{id}_A \times (\mathsf{rec}_1(X, \mathsf{sup}_X) \circ -))(a, l)) && \equiv && \text{(function application)}\\
&\mathsf{sup}_X(a, \mathsf{rec}_1(X, \mathsf{sup}_X) \circ l) && \equiv && \text{(definition of } \mathsf{rec}_1\text{)}\\
&\mathsf{sup}_X(a, (\lambda(X, \mathsf{sup}_X).\lambda(w, i).w(X, \mathsf{sup}_X))(X, \mathsf{sup}_X) \circ l) && \equiv && \text{(function application)}\\
&\mathsf{sup}_X(a, (\lambda(w, i).w(X, \mathsf{sup}_X)) \circ l) && \equiv && \text{(definition of } \mathsf{pr}_1\text{)}\\
&\mathsf{sup}_X(a, \mathsf{rev}(X, \mathsf{sup}_X) \circ \mathsf{pr}_1 \circ l)
\end{aligned}$$

as desired. □

**Theorem 2.11.** *For every* $(X, \mathsf{sup}_X) : \mathsf{Set}_{\mathsf{W}_A B}$*, there is a unique morphism* $(\mathsf{W}^1_A B, \mathsf{sup}_1) \to (X, \mathsf{sup}_X)$.

*Proof.* Assume that, $h, k : \mathsf{W}^1_A B \to X$ are such that for all $a : A$ and $l : B(a) \to \mathsf{W}^1_A B$, $\mathsf{sup}_X(a, h \circ l) = h(\mathsf{sup}_1(a, l))$, and let us prove $h = k$. Using function extensionality, it is enough to show that $\prod_{(w,i):\mathsf{W}^1_A B} h(w, i) = k(w, i)$, or, equivalently, $\prod_{w:\mathsf{W}^0_A B} \prod_{i:\mathrm{Ind}(w)} h(w, i) = k(w, i)$.

Let then $w : \mathsf{W}^0_A B$ and $i : \mathrm{Ind}(w)$ be arbitrary, and let us prove $h(w, i) = k(w, i)$. As we did with the coproducts, we strengthen the goal to $\prod_{j:\mathrm{Ind}(w)} h(w, j) = k(w, j)$, which can be then applied to the case $j :\equiv i$ to obtain our result. Now, because of Lemma 2.8, the type $\sum_{j:\mathrm{Ind}(w)} h(w, j) = k(w, j)$ implies $\prod_{j:\mathrm{Ind}(w)} h(w, j) = k(w, j)$.

Define then the type family $C(w) :\equiv \sum_{j:\mathrm{Ind}(w)} h(w, j) = k(w, j)$. Because the codomain of $h$ and $k$ is $X$, which is a $\mathsf{Set}$, $C(w)$ is always a $\mathsf{Prop}$. Hence, we can use the induction principle $i$ to construct a term of type $C(w)$, as desired. We just need to find a term of type $\prod_{(a,f):\sum_{a:A} B(a) \to \mathsf{W}^0_A B} C^{\mathsf{W}_A B}(a, f) \to C(\mathsf{sup}_0(a, f))$.

So, assume we have $a : A$, $f : B(a) \to \mathsf{W}^0_A B$, and a proof $g : \prod_{b:B(a)} \sum_{j:\mathrm{Ind}(fb)} h(fb, j) = k(fb, j)$. Given this, we need a term of type $C(\mathsf{sup}_0(a, f))$, which translates to a term $s : \mathrm{Ind}(\mathsf{sup}_0(a, f))$ together with a path $h(\mathsf{sup}_0(a, f), s) = k(\mathsf{sup}_0(a, f), s)$.

For the former, we take $s :\equiv \mathsf{sup}_{\mathrm{Ind}}(a, f)g_1$, where $g_1 :\equiv \lambda b.\, \mathsf{pr}_1(gb)$ is the first component of $gb : \sum_{j:\mathrm{Ind}(fb)} h(fb, j) = k(fb, j)$ for each $b$.

For the latter, namely $h(\mathsf{sup}_0(a, f), \mathsf{sup}_{\mathrm{Ind}}(a, f)g_1) = k(\mathsf{sup}_0(a, f), \mathsf{sup}_{\mathrm{Ind}}(a, f)g_1)$, we need more work. Remember that, by hypothesis, we know that $\mathsf{sup}_X(a, h \circ l) = h(\mathsf{sup}(a, l))$ and $\mathsf{sup}_X(a, k \circ l) = k(\mathsf{sup}(a, l))$ for every $l : B(a) \to \mathsf{W}^1_A B$. If we can find some $l$ such that $\mathsf{sup}_1(a, l) = (\mathsf{sup}_0(a, f), \mathsf{sup}_{\mathrm{Ind}}(a, f)g_1)$, then we can apply those hypotheses and reduce our problem to $\mathsf{sup}_X(a, h \circ l) = \mathsf{sup}_X(a, k \circ l)$.

A candidate for $l$ is $\lambda b.(fb, g_1 b)$. Let us prove that it does, indeed, satisfy the equality stated above. First, observe that $\mathsf{pr}_1 \circ (\lambda b.(fb, g_1 b)) \equiv f$. Then:

$$\begin{aligned}
&\mathsf{sup}_1(a, l) && \equiv \\
&\mathsf{sup}_1(a, \lambda b.(fb, g_1 b)) && \equiv \\
&(\mathsf{sup}_0(a, f), \lambda(C, \mathsf{sup}_C).\, \mathsf{sup}_C(a, f, (\lambda b. g_1 b(C, \mathsf{sup}_C))))
\end{aligned}$$

So, the first component of $\mathsf{sup}_1(a, l)$ is identical to to the first component of $(\mathsf{sup}_0(a, f), \mathsf{sup}_{\mathrm{Ind}}(a, f) g_1)$, and the second one is also equal by definition of $\mathsf{sup}_{\mathrm{Ind}}$.

We have reduced our goal to $\mathsf{sup}_X(a, h \circ (\lambda b.(fb, g_1 b))) = \mathsf{sup}_X(a, k \circ (\lambda b.(fb, g_1 b)))$, i.e. $\mathsf{sup}_X(a, \lambda b.h(fb, g_1 b)) = \mathsf{sup}_X(a, \lambda b.k(fb, g_1 b))$. For that, it is sufficient to prove $\lambda b.h(fb, g_1 b) = \lambda b.k(fb, g_1 b)$, or, by function extensionality, $\prod_{b:B(a)} h(fb, g_1 b) = k(fb, g_1 b)$.

Because $\mathrm{Ind}(fb) : \mathsf{Prop}$, from $g : \prod_{b:B(a)} \sum_{j:\mathrm{Ind}(fb)} h(fb, j) = k(fb, j)$ we can obtain a $g' : \prod_{b:B(a)} \prod_{j:\mathrm{Ind}(fb)} h(fb, j) = k(fb, j)$. So, given a $b : B(a)$ we have $g' b(g_1 b) : h(fb, g_1 b) = k(fb, g_1 b)$. Abstracting away the $b$, we get $\lambda b.g' b(g_1 b) : \prod_{b:B(a)} h(fb, g_1 b) = k(fb, g_1 b)$, as wanted. $\square$

By this theorem and Theorem 1.4, we have that $(\mathsf{W}^1_A B, \mathsf{sup}_1)$ has a full induction principle, as we wanted to see. We have shown that our Ind is at least as powerful as Nat. Can we go a bit further?

# Chapter 3

# Inductive types in $\mathcal{U}$

## 3.1 Splitting of idempotents

The approaches seen so far suffer from the key limitation of only allowing elimination into $\mathsf{Set}$. Unfortunately, this restriction has been proven vital for the proofs as they stand, and in order to lift them we will have to take quite a different route.

In their proof that $A +_N B$ is a coproduct (Theorem 2.3), Awodey, Frey, and Speight prove the $\eta$-equality $\mathsf{rec}_N(X, f \circ \mathsf{inl}_N, f \circ \mathsf{inr}_N) \sim f$ via the more specific case with $f :\equiv \mathsf{id}_{A+_N B}$, i.e. $\mathsf{rec}_N(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N) \sim \mathsf{id}_{A+_N B}$. In fact, the generalization from the latter to the former is quite abstract and does not make use of any of the specific assumptions that they impose, such as the restriction to $\mathsf{Set}$.

Shulman [11] notices this and tries to avoid having to prove this equality altogether. In this section we will lay Shulman's observations and how he proposes to apply them in order to construct inductive types not restricted to $\mathsf{Set}$. The first realization is that—even without the size restriction—the function $\varphi :\equiv \mathsf{rec}_N(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N)$ is an idempotent, i.e. $\varphi \circ \varphi \sim \varphi$. He proposes trying to split it. A splitting of an idempotent such as $\varphi : A +_N B \to A +_N B$ is a pair of functions $r : A +_N B \to J$ and $s : J \to A +_N B$ through some intermediate type $J$, such that $s \circ r \sim \varphi$ and $r \circ s \sim \mathsf{id}_J$.

If it were possible to split $\varphi$, we would have a new type $J$, with its own injections:

$$\mathsf{inl}_J :\equiv r \circ \mathsf{inl}_N$$
$$\mathsf{inr}_J :\equiv r \circ \mathsf{inr}_N$$

And its own recursor:

$$\mathsf{rec}_J : \prod_{(X, \mathsf{inl}_X, \mathsf{inr}_X)} J \to X$$
$$\mathsf{rec}_J(X, \mathsf{inl}_X, \mathsf{inr}_X) :\equiv \mathsf{rec}_N(X, \mathsf{inl}_X, \mathsf{inr}_X) \circ s$$

It turns out that this type can be proven to be the strict coproduct:

**Proposition 3.1** (Shulman). $\mathsf{rec}_J(J, \mathsf{inl}_J, \mathsf{inr}_J) \sim \mathsf{id}_J$.

*Proof.* We use two auxiliary computations. First:

$$\begin{array}{llr}
s \circ \mathsf{rec}_N(J, \mathsf{inl}_J, \mathsf{inr}_J) & \sim & \text{(naturality)} \\
\mathsf{rec}_N(A +_N B, s \circ \mathsf{inl}_J, s \circ \mathsf{inr}_J) & \sim & \text{(definition of } \mathsf{inl}_J \text{ and } \mathsf{inr}_J) \\
\mathsf{rec}_N(A +_N B, s \circ r \circ \mathsf{inl}_N, s \circ r \circ \mathsf{inr}_N) & \sim & (s \circ r \sim \varphi) \\
\mathsf{rec}_N(A +_N B, \mathsf{rec}_N(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N) \circ & & \\
\quad \mathsf{inl}_N, \mathsf{rec}_N(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N) \circ \mathsf{inr}_N) & \sim & \text{(the } \beta\text{-equalities)} \\
\mathsf{rec}_N(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N) & & (3.1)
\end{array}$$

Then:

$$\begin{array}{llr}
\mathsf{rec}_N(J, \mathsf{inl}_J, \mathsf{inr}_J) & \sim & (r \circ s \sim \mathsf{id}_J) \\
r \circ s \circ \mathsf{rec}_N(J, \mathsf{inl}_J, \mathsf{inr}_J) & \sim & (3.1) \\
r \circ \mathsf{rec}_N(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N) & \sim & (s \circ r \sim \varphi) \\
r \circ s \circ r & \sim & (r \circ s \sim \mathsf{id}_J) \\
r & & (3.2)
\end{array}$$

To finally obtain the desired result:

$$\begin{array}{llr}
\mathsf{rec}_J(J, \mathsf{inl}_J, \mathsf{inr}_J) & \sim & \text{(definition of } \mathsf{rec}_J) \\
\mathsf{rec}_N(J, \mathsf{inl}_J, \mathsf{inr}_J) \circ s & \sim & (3.2) \\
r \circ s & \sim & (r \circ s \sim \mathsf{id}_J) \\
\mathsf{id}_J & &
\end{array}$$

□

Note that the proof is completely abstract over the specific computations; the only requirements are the $\beta$-equalities, the splitting equations, use of naturality, and the relationship between $\mathsf{rec}_J$, $\mathsf{inl}_J$, and $\mathsf{inr}_J$ with their analogs for $A +_N B$.

Now it all boils down to splitting $\mathsf{rec}_N(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N)$. Very conveniently, Shulman also proved a very handy collection of results about splitting idempotents in a setting similar to the present one [10]. In particular, it turns out that finding a splitting of an idempotent $\varphi$ only requires us to find two things:

**Definition 3.1.** *A **pre-idempotent** is a function $\varphi : I \to I$ together with a proof $I : \varphi \circ \varphi \sim \varphi$.*

**Definition 3.2.** *A **quasi-idempotent** is a pre-idempotent $(\varphi, I)$ together with a proof $J : \mathsf{ap}_\varphi \circ I \sim I \circ \varphi$.*

**Theorem 3.2** (Shulman)**.** *All quasi-idempotents split.*

For a proof, see [10, Section 5]. It is important to know that, in order for these results to work, it is necessary to assume that our system contains the natural numbers $\mathbb{N}$ with their usual rules.

$I$ tells us that we can compose $\varphi$ with itself and reduce it down to $\varphi$ again. Due to associativity, $(\varphi \circ \varphi) \circ \varphi \sim \varphi \circ (\varphi \circ \varphi)$, so if we want to apply $I$ to reduce a concatenation of *three* $\varphi$ down to one, there are two ways to do it: first applying $I$, and then concatenating $\varphi$, or first applying $\varphi$ and then $I$. $J$ makes these two ways equivalent, this is, it makes the following diagram commute:

$$\begin{tikzcd}
\varphi \circ \varphi \circ \varphi \arrow[r, equal, "I \circ \varphi"] \arrow[d, equal, "\mathsf{ap}_\varphi \circ I"'] & \varphi \circ \varphi \arrow[d, equal, "I"] \arrow[dl, Rightarrow, shorten=4mm, "\mathsf{ap}_I \circ J"'] \\
\varphi \circ \varphi \arrow[r, equal, "I"'] & \varphi
\end{tikzcd}$$

In [11], Shulman claims to be able to split the idempotent for the coproduct under the definition given by Awodey et al., although a complete proof has never been published. This also requires imposing, on top of naturality, a second layer of compatibility, which we will not elaborate further upon, but can be seen in [3, Section 5]. For the rest of this chapter we will try to achieve this result—without restricting to $\mathsf{Set}$—and then generalize it to all $\mathsf{W}$-types. We will also proceed using our own construction, inductivity, instead of naturality.

## 3.2 Multiple layers of inductivity

Before we embark on this task, we will play a bit more with the concept of inductivity.

As pointed out in Chapter 2, a key drawback in our strategy of adding an inductivity property to the base System F type is that it allows us to use induction on the base type, but not on the derived subtype, which is what we would like.

Ideally, we want to reach a type $(A + B, \mathsf{inl}, \mathsf{inr})$ together with an induction rule

$$\prod_{(X,\mathsf{inl}_X,\mathsf{inr}_X):\mathcal{U}_{A+B}^{(A+B,\mathsf{inl},\mathsf{inr})}} \prod_{\alpha:A+B} X(\alpha)$$

which is equivalent to

$$\prod_{\alpha:A+B} \prod_{(X,\mathsf{inl}_X,\mathsf{inr}_X):\mathcal{U}_{A+B}^{(A+B,\mathsf{inl},\mathsf{inr})}} X(\alpha)$$

and hence amounts to proving

$$\mathrm{Ind}(\alpha) :\equiv \prod_{(X,\mathsf{inl}_X,\mathsf{inr}_X):\mathcal{U}_{A+B}^{(A+B,\mathsf{inl},\mathsf{inr})}} X(\alpha)$$

for every $\alpha : A+B$. In a way, we would want $A+B \simeq \sum_{\alpha:A+B} \mathrm{Ind}(\alpha)$, i.e. that the type $A+B$ be equivalent to the subtype of itself that has the inductivity property. Of course, we could introduce a second layer of inductivity that applies to the whole $A +_1 B \equiv \sum_{\alpha:A+_0B} \mathrm{Ind}(\alpha)$, but then that would suffer from the same problems, and so on and so forth.

Nonetheless, the idea of incorporating more than one layer could be useful. To begin with, we extrapolate our idea of inductivity so it can be applied to *any* algebra, not only the base $(A +_0 B, \mathsf{inl}_0, \mathsf{inr}_0)$:

**Definition 3.3.** *The **inductivity** of an algebra $(X, \mathsf{inl}_X, \mathsf{inr}_X) : \mathcal{U}_{A+B}$ is a property $\mathrm{Ind}_{(X,\mathsf{inl}_X,\mathsf{inr}_X)}$:*

$$\mathrm{Ind} : \prod_{(X,\mathsf{inl}_X,\mathsf{inr}_X):\mathcal{U}_{A+B}} X \to \mathcal{U}$$

$$\mathrm{Ind}_{(X,\mathsf{inl}_X,\mathsf{inr}_X)}(\alpha) :\equiv \prod_{(C,\mathsf{inl}_C,\mathsf{inr}_C):\mathcal{U}_{A+B}^{(X,\mathsf{inl}_X,\mathsf{inr}_X)}} C(\alpha)$$

Note that, in general, the property of inductivity is not a proposition, i.e. there might be propositionally distinct proofs of $\mathrm{Ind}_{(X,\mathsf{inl}_X,\mathsf{inr}_X)}(\alpha)$. So, a priori, we cannot continue as we had before.

Our original proposal would be akin to defining a base type

$$\begin{aligned} A +_0 B &:\equiv \prod_{(X,\mathsf{inl}_X,\mathsf{inr}_X):\mathcal{U}_{A+B}} X \\ \mathsf{inl}_0 &:\equiv \lambda(X,\mathsf{inl}_X,\mathsf{inr}_X).\,\mathsf{inl}_X\, a \\ \mathsf{inr}_0 &:\equiv \lambda(X,\mathsf{inl}_X,\mathsf{inr}_X).\,\mathsf{inr}_X\, b \end{aligned}$$

And our first attempt at "inductivizing" $A +_0 B$ would be

$$\begin{aligned} A +_1 B &:\equiv \sum_{\alpha:A+_0 B} \mathrm{Ind}_{(A+_0 B,\mathsf{inl}_0,\mathsf{inr}_0)}(\alpha) \\ \mathsf{inl}_1 &:\equiv \lambda a.(\mathsf{inl}_0\, a, \lambda(C,\mathsf{inl}_C,\mathsf{inr}_C).\,\mathsf{inl}_C\, a) \\ \mathsf{inr}_1 &:\equiv \lambda b.(\mathsf{inr}_0\, b, \lambda(C,\mathsf{inl}_C,\mathsf{inr}_C).\,\mathsf{inr}_C\, b) \end{aligned}$$

We can extend this indefinitely, by building a hierarchy of reversed cumulative types, each one embedded in the previous one. For readability, henceforth we will write $\mathrm{Ind}_n$ to represent $\mathrm{Ind}_{(A+_n B,\mathsf{inl}_n,\mathsf{inr}_n)}$.

**Definition 3.4.** *We define the nth approximation to the coproduct of A and B as:*

$$\begin{aligned} A +_0 B &:\equiv \prod_{(X,\mathsf{inl}_X,\mathsf{inr}_X):\mathcal{U}_{A+B}} X \\ \mathsf{inl}_0 &:\equiv \lambda(X,\mathsf{inl}_X,\mathsf{inr}_X).\,\mathsf{inl}_X\, a \\ \mathsf{inr}_0 &:\equiv \lambda(X,\mathsf{inl}_X,\mathsf{inr}_X).\,\mathsf{inr}_X\, b \end{aligned}$$

$$\begin{aligned} A +_{n+1} B &:\equiv \sum_{\alpha:A+_n B} \mathrm{Ind}_n(\alpha) \\ \mathsf{inl}_{n+1} &:\equiv \lambda a.(\mathsf{inl}_n\, a, \lambda(C,\mathsf{inl}_C,\mathsf{inr}_C).\,\mathsf{inl}_C\, a) \\ \mathsf{inr}_{n+1} &:\equiv \lambda b.(\mathsf{inr}_n\, b, \lambda(C,\mathsf{inl}_C,\mathsf{inr}_C).\,\mathsf{inr}_C\, b) \end{aligned}$$

We also define the following helper functions:

**Definition 3.5.** *Projection to the immediately underlying type:*

$$\begin{aligned} \pi_n &: A +_{n+1} B \to A +_n B \\ \pi_n &:\equiv \lambda(\alpha, i).\alpha \end{aligned}$$

*Projection to the bottom of the hierarchy:*

$$\begin{aligned} \rho &: \prod_{n:\mathbb{N}} A +_n B \to A +_0 B \\ \rho_0 &:\equiv \lambda\alpha : A +_0 B.\alpha \\ \rho_{n+1} &:\equiv \lambda(\alpha, i) : A +_{n+1} B.\rho_n(\alpha) \end{aligned}$$

*Elimination into arbitrary types of $\mathcal{U}_{A+B}$:*

$$\begin{aligned} \mathsf{rec} &: \prod_{n:\mathbb{N}} \prod_{(X,\mathsf{inl}_X,\mathsf{inr}_X):\mathcal{U}_{A+B}} A +_n B \to X \\ \mathsf{rec} &:\equiv \lambda n.\lambda(X,\mathsf{inl}_X,\mathsf{inr}_X).\lambda\alpha.\rho_n(\alpha)(X,\mathsf{inl}_X,\mathsf{inr}_X) \end{aligned}$$

Or, equivalently, $\pi_n \equiv \mathsf{pr}_1$ and $\rho_{n+1} \equiv \pi_0 \circ \pi_1 \circ \cdots \circ \pi_n$.
This way, one has:

$$\begin{aligned} \alpha &: A +_0 B \\ (\alpha, i_0) &: A +_1 B \equiv \sum_{\alpha : A +_0 B} \mathrm{Ind}_0(\alpha) \\ ((\alpha, i_0), i_1) &: A +_2 B \equiv \sum_{(\alpha, i_0) : \sum_{\alpha : A +_0 B} \mathrm{Ind}_0(\alpha)} \mathrm{Ind}_1(\alpha, i_0) \\ &\dots \end{aligned}$$

Nonetheless, it turns out that we will not need to venture down too much into this hierarchy of types; with two layers it is enough.

## 3.3 Coproducts

Before we start splitting any idempotents, we observe that Shulman's method uses naturality in the proof of Proposition 3.1. As we do not have naturality anymore, we need to fill this gap. In particular, we want to prove $s \circ \mathsf{rec}_N(J, \mathsf{inl}_J, \mathsf{inr}_J) \sim \mathsf{rec}_N(A +_N B, \mathsf{inl}_N, \mathsf{inr}_N)$ (Equation 3.1), where $s : J \to A +_N B$ is one of the functions that make up the splitting. In our context, we will prove that this equation holds for any morphism, and then we will see that $s$ is indeed a morphism.

**Lemma 3.3.** *For any $n$, and any morphism $f : (X, \mathsf{inl}_X, \mathsf{inr}_X) \to (Y, \mathsf{inl}_Y, \mathsf{inr}_Y)$, $f \circ \mathsf{rec}_{n+1}(X, \mathsf{inl}_X, \mathsf{inr}_X) \sim \mathsf{rec}_{n+1}(Y, \mathsf{inl}_Y, \mathsf{inr}_Y)$*

*Proof.* The goal is, by extensionality, to prove $(f \circ \mathsf{rec}_{n+1}(X, \mathsf{inl}_X, \mathsf{inr}_X))(\alpha, i) = \mathsf{rec}_{n+1}(Y, \mathsf{inl}_Y, \mathsf{inr}_Y)(\alpha, i)$ for all $(\alpha, i) : A +_{n+1} B$. The left-hand side can be reduced as:

$$\begin{aligned} & (f \circ \mathsf{rec}_{n+1}(X, \mathsf{inl}_X, \mathsf{inr}_X))(\alpha, i) && \equiv \quad \text{(definition of } \mathsf{rec}_{n+1}) \\ & (f \circ (\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\lambda(\alpha, i).\rho_{n+1}(\alpha, i)(X, \mathsf{inl}_X, \mathsf{inr}_X)) \\ & \quad (X, \mathsf{inl}_X, \mathsf{inr}_X))(\alpha, i) && \equiv \quad \text{(definition of } \rho_{n+1}) \\ & (f \circ (\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\lambda(\alpha, i).\rho_n(\alpha)(X, \mathsf{inl}_X, \mathsf{inr}_X)) \\ & \quad (X, \mathsf{inl}_X, \mathsf{inr}_X))(\alpha, i) && \equiv \quad \text{(function application)} \\ & (f \circ (\lambda(\alpha, i).\rho_n(\alpha)(X, \mathsf{inl}_X, \mathsf{inr}_X)))(\alpha, i) && \equiv \quad \text{(function application)} \\ & f(\rho_n(\alpha)(X, \mathsf{inl}_X, \mathsf{inr}_X)) \end{aligned}$$

And the right-hand side:

$$\begin{aligned} & \mathsf{rec}_{n+1}(Y, \mathsf{inl}_Y, \mathsf{inr}_Y)(\alpha, i) && \equiv && \text{(definition of } \mathsf{rec}_{n+1}) \\ & \rho_{n+1}(\alpha, i)(Y, \mathsf{inl}_Y, \mathsf{inr}_Y) && \equiv && \text{(definition of } \rho_{n+1}) \\ & \rho_n(\alpha)(Y, \mathsf{inl}_Y, \mathsf{inr}_Y) \end{aligned}$$

Our objective became $\prod_{(\alpha, i) : A +_{n+1} B} f(\rho_n(\alpha)(X, \mathsf{inl}_X, \mathsf{inr}_X)) = \rho_n(\alpha)(Y, \mathsf{inl}_Y, \mathsf{inr}_Y)$.
We will do this by induction on $A +_{n+1} B$. As stated earlier, induction on this hierarchy of weak coproducts only allows us to prove things on a strictly lower level. There is caveat to this restriction: we can prove statements that do not depend on the proof of inductivity. This lemma is an example.

Because neither side of the equality contains the term $i : \mathrm{Ind}(\alpha)$, we can *use* $i$ to prove the statement by inductivity. So, if we define a type family $C : A +_n B \to \mathcal{U}$ as:

$$C(\alpha) :\equiv f(\rho_n(\alpha)(X, \mathsf{inl}_X, \mathsf{inr}_X)) = \rho_n(\alpha)(Y, \mathsf{inl}_Y, \mathsf{inr}_Y)$$

We are left with supplying the proofs of $\prod_{a:A} C(\mathsf{inl}_n\, a)$ and $\prod_{b:B} C(\mathsf{inr}_n\, b)$ to $i$. The proof for $a : A$ is provided; the other is analogous. The left-hand side reduces to:

$$\begin{aligned}
& f(\rho_n(\mathsf{inl}_n\, a)(X, \mathsf{inl}_X, \mathsf{inr}_X)) && \equiv && \text{(definition of } \rho_n \text{ and } \mathsf{inl}_n\text{)}\\
& f((\mathsf{inl}_0\, a)(X, \mathsf{inl}_X, \mathsf{inr}_X)) && \equiv && \text{(definition of } \mathsf{inl}_0\text{)}\\
& f((\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\, \mathsf{inl}_X\, a)(X, \mathsf{inl}_X, \mathsf{inr}_X)) && \equiv && \text{(function application)}\\
& f(\mathsf{inl}_X\, a)
\end{aligned}$$

And the right-hand side to:

$$\begin{aligned}
& \rho_n(\mathsf{inl}_n\, a)(Y, \mathsf{inl}_Y, \mathsf{inr}_Y) && \equiv && \text{(definition of } \rho_n \text{ and } \mathsf{inl}_n\text{)}\\
& (\mathsf{inl}_0\, a)(Y, \mathsf{inl}_Y, \mathsf{inr}_Y) && \equiv && \text{(definition of } \mathsf{inl}_0\text{)}\\
& (\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\, \mathsf{inl}_X\, a)(Y, \mathsf{inl}_Y, \mathsf{inr}_Y) && \equiv && \text{(function application)}\\
& \mathsf{inl}_Y\, a
\end{aligned}$$

Because $f$ is a morphism, $f(\mathsf{inl}_X\, a) = \mathsf{inl}_Y\, a$. □

**Lemma 3.4.** *Suppose we have a coproduct* $(X, \mathsf{inl}_X, \mathsf{inr}_X)$ *with recursor* $\mathsf{rec}_X$*, and* $Y : \mathcal{U}$*. Let* $r : X \to Y$ *and* $s : Y \to X$ *be such that* $s \circ r \sim \mathsf{rec}_X(X, \mathsf{inl}_X, \mathsf{inr}_X)$*. Then* $s$ *is a morphism with respect to* $(Y, \mathsf{inl}_Y, \mathsf{inr}_Y)$*, where* $\mathsf{inl}_Y :\equiv r \circ \mathsf{inl}_X$ *and* $\mathsf{inr}_Y :\equiv r \circ \mathsf{inr}_X$*.*

*Proof.* The objective is to prove $s \circ \mathsf{inl}_Y \sim \mathsf{inl}_X$. By definition of $\mathsf{inl}_Y$, that is the same as $s \circ r \circ \mathsf{inl}_X \sim \mathsf{inl}_X$, and by hypothesis on $s$ and $r$, we get $\mathsf{rec}_X(X, \mathsf{inl}_X, \mathsf{inr}_X) \circ \mathsf{inl}_X \sim \mathsf{inl}_X$, or, equivalently, that $\mathsf{rec}_X(X, \mathsf{inl}_X, \mathsf{inr}_X)$ is a morphism, which it is because $(X, \mathsf{inl}_X, \mathsf{inr}_X)$ is a weak coproduct. The same reasoning applies to $\mathsf{inr}$. □

With this, we have replaced all the pieces of Shulman's argument that hinged on naturality. Let our candidate idempotent $\varphi : A +_2 B \to A +_2 B$ be defined as $\varphi :\equiv \mathsf{rec}_2(A +_2 B, \mathsf{inl}_2, \mathsf{inr}_2)$. We just need to prove that there are terms $I : \varphi \circ \varphi \sim \varphi$ and $J : \mathsf{ap}_\varphi \circ I \sim I \circ \varphi$. Most of the proofs in this section will only be laid out for $\mathsf{inl}_2$, as the ones for $\mathsf{inr}_2$ are completely analogous.

**Lemma 3.5.** $\varphi$ *is a pre-idempotent.*

*Proof.* Remember that [10] defines a function $\varphi : A +_2 B \to A +_2 B$ to be a pre-idempotent if it has a witness of idempotency $I : \varphi \circ \varphi \sim \varphi$, i.e. $\prod_{\alpha : A +_2 B} (\varphi \circ \varphi)(\alpha) = \varphi(\alpha)$.

As $\varphi$ first applies $\rho_2$, the statement $\varphi(\varphi((\alpha, i), j)) = \varphi(\alpha, i), j)$ does actually not involve any of the two proofs of inductivity that the term $((\alpha, i), j)$ contains, so we can repeat the strategy from the previous lemma. For the right-hand side of the equality:

$$\begin{aligned}
& \varphi((\alpha, i), j) && \equiv && \text{(definition of } \varphi\text{)}\\
& \rho_2((\alpha, i), j)(A +_2 B, \mathsf{inl}_2, \mathsf{inr}_2) && \equiv && \text{(definition of } \rho\text{)}\\
& \alpha(A +_2 B, \mathsf{inl}_2, \mathsf{inr}_2) && && 
\end{aligned} \tag{3.3}$$

And for the left-hand side:

$$\begin{aligned}
&\varphi(\varphi((\alpha,i),j)) && \equiv && (3.3)\\
&\varphi(\alpha(A+_2 B,\mathsf{inl}_2,\mathsf{inr}_2)) && \equiv && (\text{definition of } \varphi)\\
&\rho_2(\alpha(A+_2 B,\mathsf{inl}_2,\mathsf{inr}_2))(A+_2 B,\mathsf{inl}_2,\mathsf{inr}_2) && && (3.4)
\end{aligned}$$

Let us see how this proof will work. We want to prove the following statement:

$$\prod_{((\alpha,i),j):A+_2B} \varphi(\varphi((\alpha,i),j)) = \varphi((\alpha,i),j)$$

which, putting together 3.4 and 3.3, is equivalent to:

$$\prod_{((\alpha,i),j):A+_2B} \rho_2(\alpha(A+_2 B,\mathsf{inl}_2,\mathsf{inr}_2))(A+_2 B,\mathsf{inl}_2,\mathsf{inr}_2) = \alpha(A+_2 B,\mathsf{inl}_2,\mathsf{inr}_2)$$

We can write the equality as a type family on $A+_0 B$:

$$\begin{aligned}
C &: A+_0 B \to \mathcal{U}\\
C(\alpha) &:\equiv \rho_2(\alpha(A+_2 B,\mathsf{inl}_2,\mathsf{inr}_2))(A+_2 B,\mathsf{inl}_2,\mathsf{inr}_2) = \alpha(A+_2 B,\mathsf{inl}_2,\mathsf{inr}_2)
\end{aligned}$$

Our goal is to prove $C$ for all $((\alpha,i),j) : A+_0 B$. As $C$ does not make use of $i$ or $j$, we can prove this by $A+_0 B$-induction on $\alpha$, conveniently using the proof of inductivity $i$ that is also provided. $j$ will be ignored altogether for this lemma.

For this task, we need witnesses $\mathsf{inl}_C : \prod_{a:A} C(\mathsf{inl}_0\, a)$ and $\mathsf{inr}_C : \prod_{b:B} C(\mathsf{inr}_0\, b)$. It is readily seen that $C(\mathsf{inl}_0\, a)$ is equivalent to $\mathsf{inl}_2\, a = \mathsf{inl}_2\, a$. By definition of $\mathsf{inl}_0$:

$$(\mathsf{inl}_0\, a)(A+_2 B,\mathsf{inl}_2,\mathsf{inr}_2) \equiv \mathsf{inl}_2\, a \tag{3.5}$$

And, from here:

$$\begin{aligned}
&\rho_2((\mathsf{inl}_0\, a)(A+_2 B,\mathsf{inl}_2,\mathsf{inr}_2))(A+_2 B,\mathsf{inl}_2,\mathsf{inr}_2) && \equiv && (3.5)\\
&\rho_2(\mathsf{inl}_2\, a)(A+_2 B,\mathsf{inl}_2,\mathsf{inr}_2) && \equiv && (\text{definition of } \rho_n \text{ and } \mathsf{inl}_n)\\
&(\mathsf{inl}_0\, a)(A+_2 B,\mathsf{inl}_2,\mathsf{inr}_2) && \equiv && (3.5 \text{ again})\\
&\mathsf{inl}_2\, a
\end{aligned}$$

So we have our witnesses:

$$\begin{aligned}
\mathsf{inl}_C &: \prod_{a:A} C(\mathsf{inl}_1\, a) & \mathsf{inr}_C &: \prod_{b:B} C(\mathsf{inr}_1\, b)\\
\mathsf{inl}_C &:\equiv \lambda a.\mathsf{refl}_{\mathsf{inl}_2\, a} & \mathsf{inr}_C &:\equiv \lambda b.\mathsf{refl}_{\mathsf{inr}_2\, b}
\end{aligned}$$

Finally, we can construct our proof term:

$$\begin{aligned}
I &: \varphi\circ\varphi \sim \varphi\\
I &:\equiv \lambda((\alpha,i),j) : A+_2 B.i(C,\mathsf{inl}_C,\mathsf{inr}_C)
\end{aligned}$$

□

**Theorem 3.6.** *$\varphi$ is a quasi-idempotent.*

*Proof.* This means that we now need a proof $J$ of $\mathsf{ap}_\varphi \circ I \sim I \circ \varphi$. Again, we will do this by induction. This time the task is trickier: unlike before, we need to prove a statement that involves the inductivity term $i$, as that is how we built our witness of idempotency $I$. This is the reason why we are working with *two* layers of inductivity.

$$\prod_{((\alpha,i),j):A+_2 B} \mathsf{ap}_\varphi(I((\alpha,i),j)) = I(\varphi((\alpha,i),j))$$

On the left hand side of the identity type:

$$\begin{aligned}
&\mathsf{ap}_\varphi(I((\alpha,i),j)) && \equiv && \text{(definition of } I\text{)}\\
&\mathsf{ap}_\varphi((\lambda((\alpha,i),j) : A +_2 B.i(C, \mathsf{inl}_C, \mathsf{inr}_C))((\alpha,i),j)) && \equiv && \text{(function application)}\\
&\mathsf{ap}_\varphi(i(C, \mathsf{inl}_C, \mathsf{inr}_C)) && \equiv && \text{(definition of } \varphi\text{)}\\
&\mathsf{ap}_{p_2(A+_2B,\mathsf{inl}_2,\mathsf{inr}_2)}(i(C, \mathsf{inl}_C, \mathsf{inr}_C))
\end{aligned}$$

Whereas the right hand side:

$$\begin{aligned}
&I(\varphi((\alpha,i),j)) && \equiv && \text{(definition of } I \text{ and } \varphi\text{)}\\
&(\lambda((\alpha,i),j) : A +_2 B.i(C, \mathsf{inl}_C, \mathsf{inr}_C))\\
&\quad(p_2(A +_2 B, \mathsf{inl}_2, \mathsf{inr}_2)((\alpha,i),j)) && \equiv && \text{(definition of } p_2\text{)}\\
&(\lambda((\alpha,i),j) : A +_2 B.i(C, \mathsf{inl}_C, \mathsf{inr}_C))\\
&\quad(\alpha(A +_2 B, \mathsf{inl}_2, \mathsf{inr}_2)) && \equiv && \text{(function application)}\\
&\pi_1(\pi_0(\alpha(A +_2 B, \mathsf{inl}_2, \mathsf{inr}_2)))(C, \mathsf{inl}_C, \mathsf{inr}_C)
\end{aligned}$$

Our goal in this lemma is to prove $\mathsf{ap}_{p_2(A+_2B,\mathsf{inl}_2,\mathsf{inr}_2)}(i(C, \mathsf{inl}_C, \mathsf{inr}_C)) = \pi_1(\pi_0(\alpha(A+_2 B, \mathsf{inl}_2, \mathsf{inr}_2)))(C, \mathsf{inl}_C, \mathsf{inr}_C)$ for every $((\alpha,i),j) : A +_2 B$. This time, the statement *does* involve the first proof of inductivity $i$, so we can set:

$$\begin{aligned}
D(\alpha,i) :\equiv \mathsf{ap}_{p_2(A+_2B,\mathsf{inl}_2,\mathsf{inr}_2)}(i(C, \mathsf{inl}_C, \mathsf{inr}_C)) =&\\
&\pi_1(\pi_0(\alpha(A +_2 B, \mathsf{inl}_2, \mathsf{inr}_2)))(C, \mathsf{inl}_C, \mathsf{inr}_C)
\end{aligned}$$

and rewrite the goal as:

$$\prod_{((\alpha,i),j):A+_2 B} D(\alpha,i)$$

and then use the previous lemma's trick and prove $D$ by induction via $j$. For $(\alpha,i) :\equiv \mathsf{inl}_1\, a \equiv (\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\, \mathsf{inl}_X\, a, \lambda(C, \mathsf{inl}_C, \mathsf{inr}_C).\, \mathsf{inl}_C\, a)$, we have that in the left hand side of $D$:

$$\begin{aligned}
&\mathsf{ap}_{p_2(A+_2B,\mathsf{inl}_2,\mathsf{inr}_2)}(i(C, \mathsf{inl}_C, \mathsf{inr}_C)) && \equiv && \text{(definition of } i\text{)}\\
&\mathsf{ap}_{p_2(A+_2B,\mathsf{inl}_2,\mathsf{inr}_2)}((\lambda(C, \mathsf{inl}_C, \mathsf{inr}_C).\, \mathsf{inl}_C\, a)(C, \mathsf{inl}_C, \mathsf{inr}_C)) && \equiv && \text{(function appl.)}\\
&\mathsf{ap}_{p_2(A+_2B,\mathsf{inl}_2,\mathsf{inr}_2)}(\mathsf{inl}_C\, a) && \equiv && \text{(definition of } \mathsf{inl}_C\text{)}\\
&\mathsf{ap}_{p_2(A+_2B,\mathsf{inl}_2,\mathsf{inr}_2)}(\mathsf{refl}_{\mathsf{inl}_2\, a}) && \equiv && \text{(definition of } \mathsf{ap}\text{)}\\
&\mathsf{refl}_{p_2(A+_2B,\mathsf{inl}_2,\mathsf{inr}_2)(\mathsf{inl}_2\, a)} && \equiv && \text{(definition of } p_2\text{)}\\
&\mathsf{refl}_{(\mathsf{inl}_0\, a)(A+_2B,\mathsf{inl}_2,\mathsf{inr}_2)} && \equiv && \text{(definition of } \mathsf{inl}_0\text{)}\\
&\mathsf{refl}_{\mathsf{inl}_2\, a}
\end{aligned}$$

And in the right hand side:

$$
\begin{aligned}
&\pi_1(\pi_0(\alpha(A +_2 B, \mathsf{inl}_2, \mathsf{inr}_2)))(C, \mathsf{inl}_C, \mathsf{inr}_C) && \equiv && \text{(definition of } \alpha\text{)}\\
&\pi_1(\pi_0((\lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\, \mathsf{inl}_X\, a)(A +_2 B, \mathsf{inl}_2, \mathsf{inr}_2))) && && \\
&\quad (C, \mathsf{inl}_C, \mathsf{inr}_C) && \equiv && \text{(function application)}\\
&\pi_1(\pi_0(\mathsf{inl}_2\, a))(C, \mathsf{inl}_C, \mathsf{inr}_C) && \equiv && \text{(definition of } \mathsf{inl}_2\text{)}\\
&(\lambda(C, \mathsf{inl}_C, \mathsf{inr}_C).\, \mathsf{inl}_C\, a)(C, \mathsf{inl}_C, \mathsf{inr}_C) && \equiv && \text{(function application)}\\
&\mathsf{inl}_C\, a && \equiv && \text{(definition of } \mathsf{inl}_C\text{)}\\
&\mathsf{refl}_{\mathsf{inl}_2\, a} && &&
\end{aligned}
$$

So a valid witness of $D(\mathsf{inl}_1\, a)$ is $\mathsf{refl}_{\mathsf{refl}_{\mathsf{inl}_2\, a}}$; and so is $\mathsf{refl}_{\mathsf{refl}_{\mathsf{inr}_2\, b}}$ for $D(\mathsf{inr}_1\, b)$. Finally, we get the term:

$$
\begin{aligned}
&J : \mathsf{ap}_\varphi \circ I \sim I \circ \varphi\\
&J :\equiv \lambda((\alpha, i), j).j(D, \lambda a.\mathsf{refl}_{\mathsf{refl}_{\mathsf{inl}_2\, a}}, \lambda b.\mathsf{refl}_{\mathsf{refl}_{\mathsf{inr}_2\, b}})
\end{aligned}
$$

□

Now, by application of Shulman's theorem, we get a splitting of $\varphi$, and thus the coproduct of $A$ and $B$ with (dependent) elimination into the full type $\mathcal{U}$.

## 3.4 W-types

In this section we generalize the previous result to all W-types. We will mostly use the same technique of proving the intermediate results by induction, but it will be trickier as we are now dealing with an induction principle that does actually require an inductive hypothesis.

Remember definition of the type of $\mathsf{W}_A B$-like algebras (generalized from the one for Set):

$$\mathcal{U}_{\mathsf{W}_A B} :\equiv \sum_{X:\mathcal{U}} \left( \left( \sum_{a:A} B(a) \to X \right) \to X \right)$$

And $\mathsf{W}_A B$-like dependent algebras:

$$\mathcal{U}^{(X,\mathsf{sup}_X)}_{\mathsf{W}_A B} :\equiv \sum_{C:X\to\mathcal{U}} \left( \prod_{(a,f):\sum_{a:A} f:B(a)\to X} \left( \prod_{b:B(a)} C(fb) \right) \to C(\mathsf{sup}_X(a,f)) \right)$$

Which will be convenient to write our inductivity property more concisely:

**Definition 3.6.** *The **inductivity** of an algebra $(X, \mathsf{sup}_X) : \mathcal{U}_{\mathsf{W}_A B}$ is a property $\mathrm{Ind}_{(X,\mathsf{sup}_X)}$:*

$$
\begin{aligned}
\mathrm{Ind} &: \prod_{(X,\mathsf{sup}_X):\mathcal{U}_{\mathsf{W}_A B}} X \to \mathcal{U}\\
\mathrm{Ind}_{(X,\mathsf{sup}_X)}(\alpha) &:\equiv \prod_{(C,\mathsf{sup}_C):\mathcal{U}^{(X,\mathsf{sup}_X)}_{\mathsf{W}_A B}} C(\alpha)
\end{aligned}
$$

And from there, we introduce our tower of proto-inductive types.

**Definition 3.7.** *We define the nth approximation of* $\mathsf{W}_A B$ *as:*

$$\mathsf{W}^0_A B :\equiv \prod_{(X,\mathsf{sup}_X):\mathcal{U}_{\mathsf{W}_A B}} X$$

$$\mathsf{sup}_0 :\equiv \lambda(a,f).\lambda(X,\mathsf{sup}_X).\,\mathsf{sup}_X(a,\mathsf{rev}(X,\mathsf{sup}_X)\circ f)$$

$$\mathsf{W}^{n+1}_A B :\equiv \sum_{\alpha:\mathsf{W}^n_A B} \mathrm{Ind}_{(\mathsf{W}^n_A B,\mathsf{sup}_n)}(\alpha)$$

$$\begin{aligned}\mathsf{sup}_{n+1} :\equiv {} & \lambda(a,f).( \\ & \quad \mathsf{sup}_n(a,\mathsf{pr}_1 \circ f), \\ & \quad \lambda(C,\mathsf{sup}_C).\,\mathsf{sup}_C(a,\mathsf{pr}_1\circ f)(\mathsf{rev}(C,\mathsf{sup}_C)\circ \mathsf{pr}_2 \circ f) \\ & )\end{aligned}$$

As with the coproducts, we will write $\mathrm{Ind}_n$ to mean $\mathrm{Ind}_{(\mathsf{W}^n_A B,\mathsf{sup}_n)}$. We also define a few helper functions.

**Definition 3.8.** *Projection to the immediately underlying type:*

$$\pi : \prod_{n:\mathbb{N}} W^{n+1}_A B \to W^n_A B$$

$$\pi_n :\equiv \lambda(\alpha,i).\alpha$$

*Projection to the bottom of the hierarchy:*

$$\rho : \prod_{n:\mathbb{N}} W^n_A B \to W^0_A B$$

$$\rho_0 :\equiv \lambda\alpha : W^n_A B.\alpha$$

$$\rho_{n+1} :\equiv \lambda(\alpha,i) : W^{n+1}_A B.\rho_n(\alpha)$$

*Elimination into arbitrary types of* $\mathcal{U}_{\mathsf{W}_A B}$*:*

$$\mathsf{rec} : \prod_{n:\mathbb{N}} \prod_{(X,\mathsf{sup}_X):\mathcal{U}_{\mathsf{W}_A B}} W^n_A B \to X$$

$$\mathsf{rec}_n :\equiv \lambda(X,\mathsf{sup}_X).\,\mathsf{rev}(X,\mathsf{sup}_X)\circ \rho_n$$

Finally, let $\varphi :\equiv \mathsf{rec}_2(\mathsf{W}^2_A B,\mathsf{sup}_2)$ be our candidate of endomorphism to split.

For easier reference, the types and functions that we will be working explicitly with are:

$$\mathsf{W}^0_A B :\equiv \prod_{(X,\mathsf{sup}_X):\mathcal{U}_{\mathsf{W}_A B}} X$$

$$\mathsf{sup}_0 :\equiv \lambda(a,f).\lambda(X,\mathsf{sup}_X).\,\mathsf{sup}_X(a,\mathsf{rev}(X,\mathsf{sup}_X)\circ f)$$

$$\mathsf{W}^1_A B :\equiv \sum_{\alpha:\mathsf{W}^0_A B} \mathrm{Ind}_0(\alpha)$$

$$\mathsf{sup}_1 :\equiv \lambda(a,f).(\mathsf{sup}_0(a,\pi_0\circ f),\lambda(C,\mathsf{sup}_C).\,\mathsf{sup}_C(a,\pi_0\circ f)(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ f))$$

$$\mathsf{W}^2_A B :\equiv \sum_{\alpha:\mathsf{W}^1_A B} \mathrm{Ind}_1(\alpha)$$

$$\mathsf{sup}_2 :\equiv \lambda(a,f).(\mathsf{sup}_1(a,\pi_1\circ f),\lambda(C,\mathsf{sup}_C).\,\mathsf{sup}_C(a,\pi_1\circ f)(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ f))$$

We now prove that all of the functions defined so far satisfy the $\beta$-rule $f \circ \mathsf{sup}_X \equiv \mathsf{sup}_Y \circ (\mathsf{id}_A \times (f \circ -))$:

**Lemma 3.7.** *The following functions satisfy the definitional $\beta$-equality:*

- $\pi_n : \mathsf{W}^{n+1}_A B \to \mathsf{W}^n_A B$ *for all* $n : \mathbb{N}$,
- $\rho_n : \mathsf{W}^n_A B \to \mathsf{W}^0_A B$ *for all* $n : \mathbb{N}$,
- $\mathsf{rev}(X, \mathsf{sup}_X) : \mathsf{W}^0_A B \to X$ *for all* $(X, \mathsf{sup}_X) : \mathcal{U}_{\mathsf{W}_A B}$, *and*
- $\varphi : \mathsf{W}^2_A B \to \mathsf{W}^2_A B$.

*Proof.* One by one:

- For $\pi$, let $n : \mathbb{N}$, $a : A$, and $f : B(a) \to \mathsf{W}^{n+1}_A B$ be arbitrary. We must show that $\pi_n(\mathsf{sup}_{n+1}(a, f)) \equiv \mathsf{sup}_n(a, \pi_n \circ f)$.

$$\begin{aligned}
&\pi_n(\mathsf{sup}_{n+1}(a, f)) && \equiv && \text{(definition of } \pi_n\text{)}\\
&\mathsf{pr}_1(\mathsf{sup}_{n+1}(a, f)) && \equiv && \text{(definition of } \mathsf{sup}_{n+1}\text{)}\\
&\mathsf{pr}_1(\mathsf{sup}_n(a, \mathsf{pr}_1 \circ f), \dots) && \equiv && \text{(application of } \mathsf{pr}_1\text{)}\\
&\mathsf{sup}_n(a, \mathsf{pr}_1 \circ f) && \equiv && \text{(definition of } \pi_n\text{)}\\
&\mathsf{sup}_n(a, \pi_n \circ f)
\end{aligned}$$

- For $\rho$, we just remark that $\rho_0$ is the identity and $\rho_{n+1} \equiv \pi_0 \circ \cdots \circ \pi_n$.
- For $\mathsf{rev}(X, \mathsf{sup}_X)$:

$$\begin{aligned}
&\mathsf{rev}(X, \mathsf{sup}_X)(\mathsf{sup}_0(a, f)) && \equiv && \text{(definition of } \mathsf{rev}\text{)}\\
&\mathsf{sup}_0(a, f)(X, \mathsf{sup}_X) && \equiv && \text{(definition of } \mathsf{sup}_0\text{)}\\
&(\lambda(X, \mathsf{sup}_X).\, \mathsf{sup}_X(a, \mathsf{rev}(X, \mathsf{sup}_X) \circ f))(X, \mathsf{sup}_X) && \equiv && \text{(function application)}\\
&\mathsf{sup}_X(a, \mathsf{rev}(X, \mathsf{sup}_X) \circ f)
\end{aligned}$$

- Finally, for $\varphi$, it is sufficient to see that $\varphi \equiv \mathsf{rec}_2(\mathsf{W}^2_A B, \mathsf{sup}_2) \equiv \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ \rho_2$, which both satisfy the $\beta$-rule on their own as just seen.

□

As before, we need the following technicality so that the splitting argument works:

**Lemma 3.8.** *For any $n$, and any morphism $f : (X, \mathsf{sup}_X) \to (Y, \mathsf{sup}_Y)$, $f \circ \mathsf{rec}_{n+1}(X, \mathsf{sup}_X) \sim \mathsf{rec}_{n+1}(Y, \mathsf{sup}_Y)$.*

*Proof.* We prove $\prod_{(\alpha,i):\mathsf{W}^{n+1}_A B}(f \circ \mathsf{rec}_{n+1}(X, \mathsf{sup}_X))(\alpha, i) = \mathsf{rec}_{n+1}(Y, \mathsf{sup}_Y)(\alpha, i)$ by induction. We reduce both sides:

$$\begin{aligned}
&(f \circ \mathsf{rec}_{n+1}(X, \mathsf{sup}_X))(\alpha, i) && \equiv && \text{(definition of } \mathsf{rec}_{n+1}\text{)}\\
&(f \circ (\lambda(X, \mathsf{inl}_X).\lambda(\alpha, i).\rho_{n+1}(\alpha, i)(X, \mathsf{sup}_X))\\
&\quad (X, \mathsf{sup}_X))(\alpha, i) && \equiv && \text{(definition of } \rho_{n+1}\text{)}\\
&(f \circ (\lambda(X, \mathsf{sup}_X).\lambda(\alpha, i).\rho_n(\alpha)(X, \mathsf{inl}_X, \mathsf{inr}_X))\\
&\quad (X, \mathsf{sup}_X))(\alpha, i) && \equiv && \text{(function application)}\\
&(f \circ (\lambda(\alpha, i).\rho_n(\alpha)(X, \mathsf{sup}_X)))(\alpha, i) && \equiv && \text{(function application)}\\
&f(\rho_n(\alpha)(X, \mathsf{sup}_X))
\end{aligned}$$

$$\begin{aligned}
&\mathsf{rec}_{n+1}(Y, \mathsf{sup}_Y)(\alpha, i) &&\equiv && \text{(definition of } \mathsf{rec}_{n+1}\text{)}\\
&\rho_{n+1}(\alpha, i)(Y, \mathsf{sup}_Y) &&\equiv && \text{(definition of } \rho_{n+1}\text{)}\\
&\rho_n(\alpha)(Y, \mathsf{sup}_Y)
\end{aligned}$$

We use the inductivity term $i$ to prove the family $C : \mathsf{W}^n_A B \to \mathcal{U}$

$$C(\alpha) :\equiv f(\rho_n(\alpha)(X, \mathsf{sup}_X)) = \rho_n(\alpha)(Y, \mathsf{sup}_Y)$$

In this case, we only need to provide a proof for one constructor, $\mathsf{sup}_0$ (instead of two as with the coproduct). In other words, we want to find:

$$\prod_{(a,l):\sum_{a:A} l:B(a)\to \mathsf{W}^n B} \left( \prod_{b:B(a)} C(lb) \right) \to C(\mathsf{sup}_0(a, l))$$

Let then $a$ and $l$ have the required types, and $g : \prod_{b:B(a)} C(lb)$. As before, we use the fact that $f$ is a morphism, i.e. that $f(\mathsf{sup}_X(a, l)) = \mathsf{sup}_Y(a, f \circ l)$ for all $(a, l) : \sum_{a:A} B(a) \to X$. The left-hand side reduces to:

$$\begin{aligned}
&f(\rho_n(\mathsf{sup}_n(a, l))(X, \mathsf{sup}_X)) &&\equiv && \text{(Lemma 3.7)}\\
&f((\mathsf{sup}_0(a, \rho_n \circ l))(X, \mathsf{sup}_X)) &&\equiv && \text{(definition of } \mathsf{sup}_0\text{)}\\
&f((\lambda(X, \mathsf{sup}_X).\, \mathsf{sup}_X(a, \mathsf{rev}(X, \mathsf{sup}_X) \circ \rho_n \circ l))(X, \mathsf{sup}_X)) &&\equiv && \text{(function appl.)}\\
&f(\mathsf{sup}_X(a, \mathsf{rev}(X, \mathsf{sup}_X) \circ \rho_n \circ l)) &&= && \text{(}f\text{ is a morphism)}\\
&\mathsf{sup}_Y(a, f \circ \mathsf{rev}(X, \mathsf{sup}_X) \circ \rho_n \circ l)
\end{aligned}$$

And the right-hand side to:

$$\begin{aligned}
&\rho_n(\mathsf{sup}_n(a, l))(Y, \mathsf{sup}_Y) &&\equiv && \text{(Lemma 3.7)}\\
&(\mathsf{sup}_0(a, \rho_n \circ l))(Y, \mathsf{sup}_Y) &&\equiv && \text{(definition of } \mathsf{sup}_0\text{)}\\
&(\lambda(X, \mathsf{sup}_X).\, \mathsf{sup}_X(a, \mathsf{rev}(X, \mathsf{sup}_X) \circ \rho_n \circ l))(Y, \mathsf{sup}_Y) &&\equiv && \text{(function application)}\\
&\mathsf{sup}_Y(a, \mathsf{rev}(Y, \mathsf{sup}_Y) \circ \rho_n \circ l)
\end{aligned}$$

We can prove $\mathsf{sup}_Y(a, f \circ \mathsf{rev}(X, \mathsf{sup}_X) \circ \rho_n \circ l) = \mathsf{sup}_Y(a, \mathsf{rev}(Y, \mathsf{sup}_Y) \circ \rho_n \circ l)$ by using $\mathsf{ap}_{\mathsf{sup}_Y}$ and proving the equality of each component. Obviously, $a = a$ by $\mathsf{refl}_a$, so we just see that $f \circ \mathsf{rev}(X, \mathsf{sup}_X) \circ \rho_n \circ l = \mathsf{rev}(Y, \mathsf{sup}_Y) \circ \rho_n \circ l$. By function extensionality, we want to reach:

$$\prod_{b:B(a)} (f \circ \mathsf{rev}(X, \mathsf{sup}_X) \circ \rho_n \circ l)b = (\mathsf{rev}(Y, \mathsf{sup}_Y) \circ \rho_n \circ l)b$$

Or,

$$\prod_{b:B(a)} (f \circ \mathsf{rev}(X, \mathsf{sup}_X) \circ \rho_n)(lb) = (\mathsf{rev}(Y, \mathsf{sup}_Y) \circ \rho_n)(lb)$$

We finish by pointing out that $(f \circ \mathsf{rev}(X, \mathsf{sup}_X) \circ \rho_n)(lb) \equiv f(\rho_n(lb)(X, \mathsf{sup}_X))$ and $(\mathsf{rev}(Y, \mathsf{sup}_Y) \circ \rho_n)(lb) \equiv \rho_n(lb)(Y, \mathsf{sup}_Y)$, so the only missing piece is exactly our induction hypothesis $g : \prod_{b:B(a)} C(lb)$. □

**Lemma 3.9.** *For any $(X, \mathsf{sup}_X) : \mathcal{U}_{\mathsf{W}_A B}$, $a : A$, and $\psi : X \to X$, if $\psi$ satisfies the $\beta$-rule, then $\psi \circ \mathsf{sup}_X(a, -) \equiv \mathsf{sup}_X(a, -) \circ (\psi \circ -)$.*

*Proof.* It is sufficient to prove the definitional identity pointwise, i.e., for each $f : B(a) \to X$.

$$\begin{aligned}
&(\psi \circ \mathsf{sup}_X(a,-))f && \equiv && \text{(function composition)}\\
&\psi(\mathsf{sup}_X(a,-)f) && \equiv && \text{(function application)}\\
&\psi(\mathsf{sup}_X(a,f)) && \equiv && \text{(hypothesis)}\\
&\mathsf{sup}_X(a,\psi \circ f) && \equiv && \text{(undoing function application)}\\
&\mathsf{sup}_X(a,-)(\psi \circ f) && \equiv && \text{(undoing function application)}\\
&\mathsf{sup}_X(a,-)((\psi \circ -)f) && \equiv && \text{(function composition)}\\
&(\mathsf{sup}_X(a,-) \circ (\psi \circ -))f
\end{aligned}$$

□

**Lemma 3.10.** *$\varphi$ is a pre-idempotent.*

*Proof.* As before, we find a witness $I : \varphi \circ \varphi \sim \varphi$. The strategy is completely analogous to that of the coproduct, so we will skip the explanations about our usage of the proofs of inductivity and jump straight into the calculations. We structure our goal type as:

$$\prod_{((\alpha,i),j):\mathsf{W}^2_A B} (\varphi \circ \varphi)((\alpha,i),j) = \varphi((\alpha,i),j)$$

Which, unfolding $\varphi$, is definitionally identical to:

$$\prod_{((\alpha,i),j):\mathsf{W}^2_A B} (\mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ \rho_2 \circ \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2))(\alpha) = \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2)(\alpha)$$

So we define our type family $C : \mathsf{W}^0_A B \to \mathcal{U}$ as:

$$C(\alpha) :\equiv (\mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ \rho_2 \circ \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2))(\alpha) = \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2)(\alpha)$$

And restate the goal as:

$$\prod_{((\alpha,i),j):\mathsf{W}^2_A B} C(\alpha)$$

Let $a : A$ and $f : B(a) \to \mathsf{W}^0_A B$, and assume the induction hypothesis $g : \prod_{b:B(a)} C(fb)$. Applying Lemma 3.7:

$$\begin{aligned}
&(\mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ \rho_2 \circ \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2))(\mathsf{sup}_0(a,f)) \equiv\\
&\qquad\qquad \mathsf{sup}_2(a, \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ \rho_2 \circ \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ f)
\end{aligned}$$

We need to see:

$$\mathsf{sup}_2(a, \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ \rho_2 \circ \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ f) = \mathsf{sup}_2(a, \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ f)$$

Equivalently, we just find a witness of $\mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ \rho_2 \circ \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ f = \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ f$ and use $\mathsf{ap}_{\mathsf{sup}_2(a,-)}$ on it. As we want to prove equality between two functions that take an argument $b : B(a)$, we can apply function extensionality to a proof of type $\prod_{b:B(a)} (\mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ \rho_2 \circ \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ f)(b) = (\mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ f)(b)$, or equivalently $\prod_{b:B(a)} (\mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ \rho_2 \circ$

$\mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2))(fb) = \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2)(fb)$. If we observe carefully, this type turns out to be $\prod_{b:B(a)} C(fb)$, which is exactly that of our induction hypothesis $g$. So the proof by induction is given by $\lambda(a, f).\lambda g.\, \mathsf{ap}_{\mathsf{sup}_2(a,-)}(\mathsf{funext}\, g)$, and hence this lemma is proven by:

$$I : \varphi \circ \varphi \sim \varphi$$
$$I :\equiv \lambda((\alpha, i), j).i(C, \lambda(a, f).\lambda g.\, \mathsf{ap}_{\mathsf{sup}_2(a,-)}(\mathsf{funext}\, g))$$

□

For the final theorem we will need a way of computationally dealing with the function extensionality axiom funext that we have introduced in our term $I$. The way it is defined in [6, Axiom 2.9.3], the only computational rules that funext obeys are related to its quasi-inverse $\mathsf{happly} : f = f' \to \prod_{x:X} fx = f'x$:

$$\forall x : X\ \forall k : \prod_{x:X} fx = f'x \quad \mathsf{happly}(\mathsf{funext}\, k, x) = kx \tag{3.6}$$

$$\forall p : f = f' \quad \mathsf{funext}(\lambda x.\, \mathsf{happly}(p, x)) = p \tag{3.7}$$

Which can be regarded as computation and uniqueness principles for identity types of functions, correspondingly. Unfortunately, our calculations will not contain happly, but they will contain a propositionally equivalent construction:

**Lemma 3.11.** $\mathsf{happly} = \lambda p.\lambda x.\, \mathsf{ap}_{\mathsf{rev}\, x}\, p$.

*Proof.* As given by [6, Equation 2.9.2], happly is defined by path induction for the case $f \equiv f'$ as $\mathsf{happly}(\mathsf{refl}_f) :\equiv \lambda x.\mathsf{refl}_{fx}$. On the other hand, ap is also defined by induction, in such a way that $\mathsf{ap}_{\mathsf{rev}\, x}\, \mathsf{refl}_f \equiv \mathsf{refl}_{fx}$. Abstracting on $x$, we have that $\lambda x.\, \mathsf{ap}_{\mathsf{rev}\, x}\, \mathsf{refl}_f \equiv \lambda x.\mathsf{refl}_{fx}$. Then, we have by path induction that $\mathsf{happly}(p) = \lambda x.\, \mathsf{ap}_{\mathsf{rev}\, x}\, p$: we just need to prove it for $p :\equiv \mathsf{refl}_f$, using for example $\mathsf{refl}_{\lambda x.\mathsf{refl}_{fx}}$. Finally, by applying funext we obtain a proof of $\mathsf{happly} = \lambda p.\lambda x.\, \mathsf{ap}_{\mathsf{rev}\, x}\, p$. □

By applying Lemma 3.11 to 3.6 and 3.7 we obtain the following equalities:

$$\forall x : X\ \forall k : \prod_{x:X} fx = f'x \quad \mathsf{ap}_{\mathsf{rev}\, x}(\mathsf{funext}\, k) = kx \tag{3.8}$$

$$\forall p : f = f' \quad \mathsf{funext}(\lambda x.\, \mathsf{ap}_{\mathsf{rev}\, x}\, p) = p \tag{3.9}$$

with which we can arrive to:

**Lemma 3.12.** *Let $\varphi : Y \to Y$ and $k : \prod_{x:X} fx = f'x$ for any $X, Y$ types and $f, f' : X \to Y$. Then,* $\mathsf{ap}_{\varphi\circ -}(\mathsf{funext}\, k) = \mathsf{funext}(\mathsf{ap}_\varphi \circ k)$.

*Proof.*

$$\begin{array}{lcr}
\mathsf{ap}_{\varphi\circ -}(\mathsf{funext}\, k) & = & (3.9)\\
\mathsf{funext}(\lambda x.\, \mathsf{ap}_{\mathsf{rev}\, x}(\mathsf{ap}_{\varphi\circ -}(\mathsf{funext}\, k))) & = & (\text{functoriality of } \mathsf{ap})\\
\mathsf{funext}(\lambda x.\, \mathsf{ap}_{\mathsf{rev}\, x \circ(\varphi\circ -)}(\mathsf{funext}\, k)) & \equiv & \\
\mathsf{funext}(\lambda x.\, \mathsf{ap}_{(\varphi\circ -)x}(\mathsf{funext}\, k)) & \equiv & \\
\mathsf{funext}(\lambda x.\, \mathsf{ap}_{\varphi\circ \mathsf{rev}\, x}(\mathsf{funext}\, k)) & = & (\text{functoriality of } \mathsf{ap})\\
\mathsf{funext}(\lambda x.\, \mathsf{ap}_{\varphi}(\mathsf{ap}_{\mathsf{rev}\, x}(\mathsf{funext}\, k))) & = & (3.8)\\
\mathsf{funext}(\lambda x.\, \mathsf{ap}_{\varphi}(kx)) & \equiv & (\eta\text{-reduction})\\
\mathsf{funext}(\mathsf{ap}_{\varphi} \circ k) & &
\end{array}$$

□

**Theorem 3.13.** *$\varphi$ is a quasi-idempotent.*

*Proof.* By the same procedure as with the coproduct, we want $J : \mathsf{ap}_\varphi \circ I \sim I \circ \varphi$. Let us abbreviate $\lambda(a,f).\lambda g.\, \mathsf{ap}_{\mathsf{sup}_2(a,-)}(\mathsf{funext}\, g)$ as $\mathsf{sup}_C$ (as it will form a dependent algebra with our type family $C$). $I$ can be rewritten as $\mathsf{rev}(C, \mathsf{sup}_C) \circ \mathsf{pr}_2 \circ \pi_1$. First, we reduce the goal:

$$\begin{aligned}
&(\mathsf{ap}_\varphi \circ I)((\alpha,i),j) && \equiv && \text{(definition of } I\text{)}\\
&(\mathsf{ap}_\varphi \circ \mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2 \circ \pi_1)((\alpha,i),j) && \equiv && \text{(definition of } \pi_1\text{)}\\
&(\mathsf{ap}_\varphi \circ \mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2)(\alpha,i)
\end{aligned}$$

$$\begin{aligned}
&(I \circ \varphi)((\alpha,i),j) && \equiv && \text{(definition of } I \text{ and } \varphi\text{)}\\
&(\mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2 \circ \pi_1 \circ \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ \rho_2)((\alpha,i),j) && \equiv && \text{(definition of } \rho_2\text{)}\\
&(\mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2 \circ \pi_1 \circ \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ \rho_1)(\alpha,i) && \equiv && \text{(definition of } \rho_1\text{)}\\
&(\mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2 \circ \pi_1 \circ \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ \pi_0)(\alpha,i)
\end{aligned}$$

We set:

$$\begin{aligned}
D(\alpha,i) :\equiv (\mathsf{ap}_\varphi \circ \mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2)(\alpha,i) = \\
(\mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2 \circ \pi_1 \circ \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ \pi_0)(\alpha,i)
\end{aligned}$$

and prove it for all $((\alpha,i),j)$ by induction using $j$.

Let us prove $D(\mathsf{sup}_1(a,f))$ for arbitrary $a : A$, $f : B(a) \to \mathsf{W}^1_A B$, and given a proof $g : \prod_{b:B(a)} D(fb)$, i.e. $\prod_{b:B(a)} (\mathsf{ap}_\varphi \circ \mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2 \circ f)(b) = (\mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2 \circ \pi_1 \circ \mathsf{rev}(\mathsf{W}^2_A B, \mathsf{sup}_2) \circ \pi_0 \circ f)(b)$. By the definition of $\mathsf{sup}_1$, $D(\mathsf{sup}_1(a,f))$ can be simplified. For the left-hand side of the equality:

$$\begin{aligned}
&(\mathsf{ap}_\varphi \circ \mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2)(\mathsf{sup}_1(a,f)) && \equiv && \text{(definition of } \mathsf{sup}_1\text{)}\\
&(\mathsf{ap}_\varphi \circ \mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2)(\ldots,\\
&\quad \lambda(C,\mathsf{sup}_C).\, \mathsf{sup}_C(a, \pi_0 \circ f)(\mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2 \circ f)) && \equiv && \text{(application of } \mathsf{pr}_2\text{)}\\
&(\mathsf{ap}_\varphi \circ \mathsf{rev}(C,\mathsf{sup}_C))\\
&\quad (\lambda(C,\mathsf{sup}_C).\, \mathsf{sup}_C(a, \pi_0 \circ f)(\mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2 \circ f)) && \equiv && \text{(definition of } \mathsf{rev}\text{)}\\
&\mathsf{ap}_\varphi(\mathsf{sup}_C(a, \pi_0 \circ f)(\mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2 \circ f)) && \equiv && \text{(definition of } \mathsf{sup}_C\text{)}\\
&\mathsf{ap}_\varphi((\lambda(a,f).\lambda g.\, \mathsf{ap}_{\mathsf{sup}_2(a,-)}(\mathsf{funext}\, g))\\
&\quad (a, \pi_0 \circ f)(\mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2 \circ f)) && \equiv && \text{(function application)}\\
&\mathsf{ap}_\varphi(\mathsf{ap}_{\mathsf{sup}_2(a,-)}(\mathsf{funext}(\mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2 \circ f))) && = && \text{(functoriality of } \mathsf{ap}\text{)}\\
&\mathsf{ap}_{\varphi \circ \mathsf{sup}_2(a,-)}(\mathsf{funext}(\mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2 \circ f)) && \equiv && \text{(Lemma 3.9)}\\
&\mathsf{ap}_{\mathsf{sup}_2(a,-) \circ (\varphi \circ -)}(\mathsf{funext}(\mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2 \circ f)) && = && \text{(functoriality of } \mathsf{ap}\text{)}\\
&\mathsf{ap}_{\mathsf{sup}_2(a,-)}(\mathsf{ap}_{(\varphi \circ -)}(\mathsf{funext}(\mathsf{rev}(C,\mathsf{sup}_C) \circ \mathsf{pr}_2 \circ f))) && && (3.10)
\end{aligned}$$

And, for the right-hand side:

$$
\begin{aligned}
&(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ\pi_1\circ\mathsf{rev}(\mathsf{W}^2_A B,\mathsf{sup}_2)\circ\pi_0)(\mathsf{sup}_1(a,f)) && \equiv \quad \text{(Lemma 3.7)}\\
&(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ\pi_1\circ\mathsf{rev}(\mathsf{W}^2_A B,\mathsf{sup}_2))\\
&\quad(\mathsf{sup}_0(a,\rho_0\circ\pi_0\circ f)) && \equiv \quad \text{(definition of } \rho_0)\\
&(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ\pi_1\circ\mathsf{rev}(\mathsf{W}^2_A B,\mathsf{sup}_2))(\mathsf{sup}_0(a,\pi_0\circ f)) && \equiv \quad \text{(definition of } \mathsf{sup}_0)\\
&(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ\pi_1\circ\mathsf{rev}(\mathsf{W}^2_A B,\mathsf{sup}_2))\\
&\quad(\lambda(X,\mathsf{sup}_X).\,\mathsf{sup}_X(a,\mathsf{rev}(X,\mathsf{sup}_X)\circ\pi_0\circ f)) && \equiv \quad \text{(definition of } \mathsf{rev})\\
&(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ\pi_1)(\mathsf{sup}_2(a,\mathsf{rev}(\mathsf{W}^2_A B,\mathsf{sup}_2)\circ\pi_0\circ f)) && \equiv \quad \text{(Lemma 3.7)}\\
&(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2)(\mathsf{sup}_1(a,\pi_1\circ\mathsf{rev}(\mathsf{W}^2_A B,\mathsf{sup}_2)\circ\pi_0\circ f)) && \equiv \quad \text{(definition of } \mathsf{sup}_1)\\
&\mathsf{rev}(C,\mathsf{sup}_C)\\
&\quad(\lambda(C,\mathsf{sup}_C).\,\mathsf{sup}_C(a,\pi_0\circ\pi_1\circ\mathsf{rev}(\mathsf{W}^2_A B,\mathsf{sup}_2)\circ\pi_0\circ f)\\
&\quad(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ\pi_1\circ\mathsf{rev}(\mathsf{W}^2_A B,\mathsf{sup}_2)\circ\pi_0\circ f)) && \equiv \quad \text{(definition of } \mathsf{rev})\\
&\mathsf{sup}_C(a,\rho_2\circ\mathsf{rev}(\mathsf{W}^2_A B,\mathsf{sup}_2)\circ\pi_0\circ f)\\
&\quad(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ\pi_1\circ\mathsf{rev}(\mathsf{W}^2_A B,\mathsf{sup}_2)\circ\pi_0\circ f) && \equiv \quad \text{(definition of } \mathsf{sup}_C)\\
&(\lambda(a,f).\lambda g.\,\mathsf{ap}_{\mathsf{sup}_2(a,-)}(\mathsf{funext}\,g))\\
&\quad(a,\rho_2\circ\mathsf{rev}(\mathsf{W}^2_A B,\mathsf{sup}_2)\circ\pi_0\circ f)\\
&\quad(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ\pi_1\circ\mathsf{rev}(\mathsf{W}^2_A B,\mathsf{sup}_2)\circ\pi_0\circ f) && \equiv \quad \text{(function appl.)}\\
&\mathsf{ap}_{\mathsf{sup}_2(a,-)}(\mathsf{funext}(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ\\
&\quad\pi_1\circ\mathsf{rev}(\mathsf{W}^2_A B,\mathsf{sup}_2)\circ\pi_0\circ f)) && \qquad\qquad (3.11)
\end{aligned}
$$

Putting together 3.10 and 3.11, our goal is:

$$
\begin{aligned}
&\mathsf{ap}_{\mathsf{sup}_2(a,-)}(\mathsf{ap}_{(\varphi\circ-)}(\mathsf{funext}(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ f))) =\\
&\qquad\qquad \mathsf{ap}_{\mathsf{sup}_2(a,-)}(\mathsf{funext}(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ\pi_1\circ\mathsf{rev}(\mathsf{W}^2_A B,\mathsf{sup}_2)\circ\pi_0\circ f))
\end{aligned}
$$

Using $\mathsf{ap}_{\mathsf{ap}_{\mathsf{sup}_2(a,-)}}$, we can reduce it to $\mathsf{ap}_{(\varphi\circ-)}(\mathsf{funext}(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ f)) = \mathsf{funext}(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ\pi_1\circ\mathsf{rev}(\mathsf{W}^2_A B,\mathsf{sup}_2)\circ\pi_0\circ f)$.

Now comes the time to use our induction hypothesis $g$; observe that:

$$
\begin{aligned}
&\mathsf{ap}_{\mathsf{funext}}(\mathsf{funext}\,g) : \mathsf{funext}(\mathsf{ap}_\varphi\circ\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ f) =\\
&\qquad\qquad\qquad \mathsf{funext}(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ\pi_1\circ\mathsf{rev}(\mathsf{W}^2_A B,\mathsf{sup}_2)\circ\pi_0\circ f)
\end{aligned}
$$

so we can change our goal to

$$\mathsf{ap}_{(\varphi\circ-)}(\mathsf{funext}(\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ f)) = \mathsf{funext}(\mathsf{ap}_\varphi\circ\mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ f)$$

Finally, we just need to apply Lemma 3.12 with $k :\equiv \mathsf{rev}(C,\mathsf{sup}_C)\circ\mathsf{pr}_2\circ f$. $\square$

We have seen that we can use Ind to construct inductive types via their System F encoding, and then generalized via W-types. The next generalization step is to consider the categorical interpretation of inductive types.

# Chapter 4

# Initial algebras of endofunctors

As with many concepts in mathematics, inductive types can be given a characterization in the abstract framework of category theory. The reader will already have noticed the familiar usage of terms such as "algebras" and "morphisms". In this chapter we will abstract away from a specific inductive type to present a language to talk about *any* inductive type in general, and we will begin studying different possibilities on how to generalize the theory seen so far.

## 4.1 Initial algebras

First, we need to establish the base category over which we develop the rest of this theory.

**Definition 4.1.** *$\mathcal{U}$ has the structure of a category, where the objects are its types and the morphisms are the (non-dependent) functions between such types, up to homotopy, i.e. two functions $f, g : X \to Y$ are considered equal in such category if $f \sim g \equiv \prod_{x:X} fx = gx$ is inhabited. We will abusively write $\mathcal{U}$ for this category.*

That this is indeed a category follows immediately from the analogous properties of functions between types.

Under the assumption of function extensionality, $f \sim g$ is equivalent to $f = g$, and hence equality of morphisms as defined above is equivalent to their propositional equality as terms in the type system. Nonetheless, we maintain this definition for ease of use.

Between categories, one can define functors, which take objects to objects and morphisms to morphisms in a compatible manner. In particular, one has endofunctors:

**Definition 4.2.** *An endofunctor $T : \mathcal{U} \Rightarrow \mathcal{U}$ consists of:*

- *An action on objects $T_0 : \mathcal{U} \to \mathcal{U}$.*
- *An action on morphisms $T_1 : (X \to Y) \to (T_0 X \to T_0 Y)$, such that $T_1\, \mathsf{id}_X \sim \mathsf{id}_{T_0 X}$ and $T_1(f \circ g) \sim T_1 f \circ T_1 g$.*

Henceforth when we say an endofunctor, we will refer to an endofunctor of $\mathcal{U}$, unless otherwise stated.

**Definition 4.3.** *An algebra of an endofunctor $T$ (or a $T$-algebra) is a pair $(X, \alpha_X)$ where $X : \mathcal{U}$ and $\alpha_X : T_0X \to X$. We write $T$-Alg $:\equiv \sum_{X:\mathcal{U}} T_0X \to X$.*

**Definition 4.4.** *The category of algebras of an endofunctor $T$ has as objects all the algebras of $T$. A morphism $f : (X, \alpha_X) \to (Y, \alpha_Y)$ is a function $f : X \to Y$ such that $\alpha_Y \circ T_1 f \sim f \circ \alpha_X$.*

*We call the initial object of this category an initial algebra.*

We will often abuse notation and write $f : (X, \alpha_X) \to (Y, \alpha_Y)$ to mean the underlying function from $X$ to $Y$.

The property that a morphism $f$ satisfies is the commutativity of the following diagram:

$$\begin{tikzcd}
T_0X \arrow[r, "T_1 f"] \arrow[d, "\alpha_X"'] & T_0Y \arrow[d, "\alpha_Y"] \\
X \arrow[r, "f"'] & Y
\end{tikzcd}$$

That this forms a category is quite straightforward: clearly $\mathsf{id}_X : X \to X$ satisfies $\alpha_X \circ T_1\,\mathsf{id}_X \sim \alpha_X \circ \mathsf{id}_X \sim \alpha_X \sim \mathsf{id}_X \circ \alpha_X$, and for composition of morphisms we observe that, in the following diagram:

$$\begin{tikzcd}
T_0X \arrow[r, "T_1 f"] \arrow[d, "\alpha_X"'] & T_0Y \arrow[r, "T_1 g"] \arrow[d, "\alpha_Y"'] & T_0Z \arrow[d, "\alpha_Z"] \\
X \arrow[r, "f"'] & Y \arrow[r, dash, "g"'] & Z
\end{tikzcd}$$

the composition of the two commuting squares results in a commuting rectangle, and we use the fact that $T_1 f \circ T_1 g \sim T_1(f \circ g)$.

Now, the gist is to encode every "candidate" to an inductive type as an algebra of a certain endofunctor. For instance, the natural numbers have been described as having two constructors:

$$\begin{aligned} 0 &: \mathbb{N} \\ \mathsf{succ} &: \mathbb{N} \to \mathbb{N} \end{aligned}$$

This is equivalent to

$$\begin{aligned} 0 &: \mathbb{1} \to \mathbb{N} \\ \mathsf{succ} &: \mathbb{N} \to \mathbb{N} \end{aligned}$$

Which can be packed into $(0, \mathsf{succ}) : (\mathbb{1} \to \mathbb{N}) \times (\mathbb{N} \to \mathbb{N})$. Finally, this type is equivalent to $(\mathbb{1} + \mathbb{N}) \to \mathbb{N}$. Under this new shape, $\alpha : (\mathbb{1} + \mathbb{N}) \to \mathbb{N}$ can act as $0$ ($(\alpha \circ \mathsf{inl})\star$) and as $\mathsf{succ}$ ($\alpha \circ \mathsf{inr}$).

This way of molding the constructors into a single function is not specific to $\mathbb{N}$. Remember that all constructors of all inductive types have as codomain the type itself, and any collection of constructors $\alpha_0 : A^0_0 \to \cdots \to A^0_{k_0} \to X, \ldots, \alpha_n : A^n_0 \to \cdots \to A^n_{k_n} \to X$ can be glued together into one via

$$(A^i_0 \to \cdots \to A^i_{k_i} \to X) \simeq (A^i_0 \times \cdots \times A^i_{k_i}) \to X$$

and then

$$\begin{aligned} &((A^0_0 \times \cdots \times A^0_{k_0}) \to X) \times \cdots \times ((A^n_0 \times \cdots \times A^n_{k_n}) \to X) \\ &\qquad\qquad \simeq ((A^0_0 \times \cdots \times A^0_{k_0}) + \cdots + (A^n_0 \times \cdots \times A^n_{k_n})) \to X \end{aligned}$$

Ultimately, one can define an endofunctor $T_0X :\equiv ((A^0_0 \times \cdots \times A^0_{k_0}) + \cdots + (A^n_0 \times \cdots \times A^n_{k_n}))$ and consider $X$ together with the constructor $\alpha : T_0X \to X$ as a $T$-algebra. In the case of $\mathbb{N}$, we can define $T$ as:

$$T_0X :\equiv \mathbb{1} + X$$
$$T_1f :\equiv [\mathsf{id}_{\mathbb{1}}, f]$$

Where, given $f : X \to Y$, $[\mathsf{id}_{\mathbb{1}}, f] : (\mathbb{1} + X) \to (\mathbb{1} + Y)$ returns $\mathsf{inl}(\mathsf{id}_{\mathbb{1}} \star)$ for $\mathsf{inl}\,\star$ and $\mathsf{inr}(fx)$ for $\mathsf{inr}\,x$. An $\mathbb{N}$-morphism from $(X, \alpha_X)$ to $(Y, \alpha_Y)$ is then a function $f : X \to Y$ such that $\alpha_Y \circ T_1f \sim f \circ \alpha_X$, or, equivalently, $\alpha_Y \circ [\mathsf{id}_{\mathbb{1}}, f] \sim f \circ \alpha_X$:

$$\begin{tikzcd}
\mathbb{1} + X \arrow[r, "{[\mathsf{id}_{\mathbb{1}}, f]}"] \arrow[d, "\alpha_X"'] \arrow[dr, phantom, "\sim"] & \mathbb{1} + Y \arrow[d, "\alpha_Y"] \\
X \arrow[r, "f"'] & Y
\end{tikzcd}$$

For each of the two possibilities in $\mathbb{1} + X$, $\mathsf{inl}\,\star$ and $\mathsf{inr}\,n$, this means:

$$f(\alpha_X(\mathsf{inl}\,\star)) = \alpha_Y(\mathsf{inl}\,\star)$$
$$f(\alpha_X(\mathsf{inr}\,n)) = \alpha_Y(\mathsf{inr}\,fn)$$

which, can be read as

$$f(0_X) = 0_Y$$
$$f(\mathsf{succ}_X\,n) = \mathsf{succ}_Y(fn)$$

by interpreting $0_X :\equiv \alpha_X(\mathsf{inl}\,\star)$, $\mathsf{succ}_X :\equiv \alpha_X \circ \mathsf{inr}$, and similarly for $Y$.

As for our $A +_0 B$:

$$\prod_{(X, \mathsf{inl}_X, \mathsf{inr}_X) : \sum_{X:\mathcal{U}} (A \to X) \times (B \to X)} X$$

We can merge the two constructors $\mathsf{inl}_X$ and $\mathsf{inr}_X$ into one:

$$\prod_{(X, \alpha_X) : \sum_{X:\mathcal{U}} (A + B) \to X} X$$

Where $A + B$ is the coproduct of the type system[1] constructed as $\sum_{b:\mathbf{2}} S(b)$ where $S(0_{\mathbf{2}}) :\equiv A$ and $S(1_{\mathbf{2}}) :\equiv B$.

Seeing this result, a good choice for $T$ is the constant endofunctor that takes any type $X$ to the type $A + B$, and any function $f : X \to Y$ to $\mathsf{id}_{A+B}$.

Now, remember that in an arbitrary category, a weakly initial object is one that has morphisms into every other object. A (strictly) initial object is one that has exactly one morphism into every other object. As hinted at previously the System F encoding yields *weakly* initial algebras of any endofunctor $T$.

The naming is not coincidental: the (weak) coproduct corresponds with the (weakly) initial algebra for the endofunctor $TX :\equiv A + B$, as the definitions match. The weak version states existence of morphisms out of the type, and the strict version, on top of that, also guarantees uniqueness.

[1]Of course, constructing the coproduct on top of a type system that already has it is not a useful endeavor per se, but it is an easy to conceive example.

**Proposition 4.1.** *There exists a weakly initial algebra of $T$.*

*Proof.* For the carrier type $I$, we take $\prod_{(X,\alpha_X)} X$. Now we need an associated constructor $\alpha_I : T_0 I \to I$. Consider the application function $\mathsf{rev} : \prod_{a:A} \left(\prod_{a':A} B(a')\right) B(a)$, such that $\mathsf{rev}\, a f :\equiv f a$. In our case, we take $\mathsf{rev}(X, \alpha_X) : I \to X$, and we apply $T_1$ to obtain $T_1\, \mathsf{rev}(X, \alpha_X) : T_0 I \to T_0 X$. From here, we can build:

$$\begin{aligned}
\alpha_I &: T_0 \left( \prod_{(X,\alpha_X)} X \right) \to \prod_{(X,\alpha_X)} X \\
\alpha_I &: \lambda t.\lambda(X, \alpha_X).(\alpha_X \circ T_1\, \mathsf{rev}(X, \alpha_X))t
\end{aligned}$$

Now we need to prove that $(I, \alpha_I) : T\text{-Alg}$ is weakly initial. Let $(Y, \alpha_Y)$ be another $T$-algebra, and let us build a morphism of $T$-algebras from $(I, \alpha_I)$ to $(Y, \alpha_Y)$. Our candidate is $f :\equiv \mathsf{rev}(Y, \alpha_Y)$. Clearly the types match, but we need to see that it commutes with the constructors. In fact, not only does it commute with them, but it does so definitionally:

$$\begin{aligned}
&(f \circ \alpha_I)t && \equiv \quad \text{(definition of } f \text{ and } \alpha_I) \\
&(\mathsf{rev}(Y, \alpha_Y) \circ (\lambda t.\lambda(X, \alpha_X).(\alpha_X \circ T_1\, \mathsf{rev}(X, \alpha_X))t))t && \equiv \quad \text{(function application)} \\
&\mathsf{rev}(Y, \alpha_Y)(\lambda(X, \alpha_X).(\alpha_X \circ T_1\, \mathsf{rev}(X, \alpha_X))t) && \equiv \quad \text{(definition of } \mathsf{rev}) \\
&(\lambda(X, \alpha_X).(\alpha_X \circ T_1\, \mathsf{rev}(X, \alpha_X))t)(Y, \alpha_Y) && \equiv \quad \text{(function application)} \\
&(\alpha_Y \circ T_1\, \mathsf{rev}(Y, \alpha_Y))t && \equiv \quad \text{(definition of } f) \\
&(\alpha_Y \circ T_1 f)t
\end{aligned}$$

□

We can rewrite the function $f : I \to Y$ above to point to any $T$-algebra as such: $\lambda(Y, \alpha_Y).\, \mathsf{rev}(Y, \alpha_Y)$ (which, by $\eta$-reduction, is just $\mathsf{rev}$). This is our candidate to elimination principle of $(I, \alpha_I)$:

$$\begin{aligned}
&\mathsf{rec} : \prod_{(Y,\alpha_Y):T\text{-Alg}} I \to Y \\
&\mathsf{rec} :\equiv \mathsf{rev}
\end{aligned}$$

## 4.2 Adapting Ind to arbitrary endofunctors

Let us take a look again at the property of inductivity for coproducts:

$$\mathrm{Ind}_{(X,\mathsf{inl}_X,\mathsf{inr}_X)}(\alpha) :\equiv \prod_{(C,\mathsf{inl}_C,\mathsf{inr}_C):\sum_{C:X\to\mathcal{U}}\left(\prod_{a:A} C(\mathsf{inl}_X\, a)\right)\times\left(\prod_{b:B} C(\mathsf{inr}_X\, b)\right)} C(\alpha)$$

And the one for $\mathsf{W}$-types:

$$\mathrm{Ind}_{(X,\alpha_X)}(x) :\equiv \prod_{(C,\mathsf{sup}_C):\sum_{C:X\to\mathcal{U}}\left(\prod_{(a,f):\sum_{a:A} B(a)\to X} C^{\mathsf{W}_A B}(a,f)\to C(\mathsf{sup}_0(a,f))\right)} C(x)$$

where $C^{\mathsf{W}_A B}(a, f) :\equiv \prod_{b:B(a)} C(fb)$.

There is a clear pattern: we take a type family $C$ over the type for which we want to define inductivity, together with a proof, and return a member the fiber of $C$ over the inductive element. This "proof" that we supply is the core of the induction. For coproducts it is simple enough: for each $a : A$ provide an inhabitant of $C(\mathsf{inl}_X\, a)$, and for each $b : B$ one of $C(\mathsf{inr}_X\, b)$.

For $\mathsf{W}$-types it gets a bit more delicate. For each $a : A$ and $f : B(a) \to X$, now it is not sufficient to directly provide some term of $C$. We need to provide a proof which, assuming that the family is inhabited for every subtree ($\prod_{b:B(a)} C(fb)$), then it is also inhabited at the supremum of $(a, f)$. This new term, $\prod_{b:B(a)} C(fb) \equiv C^{\mathsf{W}_A B}$, is what we usually call the induction hypothesis.

Based on our knowledge of the induction principle for $\mathbb{N}$, we can imagine how inductivity for its algebras would look like:

$$\mathrm{Ind}_{(X,0_X,\mathsf{succ}_X)}(x) :\equiv \prod_{(C,0_C,\mathsf{succ}_C):\sum_{C:X\to\mathcal{U}} C(0_x)\times\left(\prod_{x':X} C(x')\to C(\mathsf{succ}_X\, x')\right)} C(x)$$

For a general endofunctor $T$, we can write the inductivity property for its $T$-algebras $(X, \alpha_X)$ as:

$$\mathrm{Ind}_{(X,\alpha_X)}(x) :\equiv \prod_{(C,\alpha_C):\sum_{C:X\to\mathcal{U}} \prod_{t:T_0 X} C^T(t)\to C(\alpha_X t)} C(x)$$

More closely:

$$\alpha_C : \prod_{t:T_0 X} C^T(t) \to C(\alpha_X t)$$

Here $C^T(t)$ is a type family over $T_0 X$ that represents the induction hypothesis for $t : T_0 X$. This will depend on the particular $T$. For degenerate inductive types, that do not require an induction hypothesis, we can just choose $C^T(t) :\equiv \mathbb{1}$, i.e. they provide no information, and in such case $C^T(t) \to C(\alpha_X t) \simeq \mathbb{1} \to C(\alpha_X t) \simeq C(\alpha_X t)$. For proper inductive types, it is more complicated. The naturals (given by $T_0 X :\equiv \mathbb{1} + X$) can be encoded with $C^T :\equiv [\lambda\star.\mathbb{1}, C]$; in other words, $C^T(\mathsf{inl}\,\star) :\equiv \mathbb{1}$ and $C^T(\mathsf{inr}\, x) :\equiv C(x)$, so that:

$$\prod_{t:T_0 X} C^T(t) \to C(\alpha_X t) \simeq \tag{4.1}$$

$$\prod_{t:\mathbb{1}+X} C^T(t) \to C(\alpha_X t) \simeq \tag{4.2}$$

$$\left(\prod_{\star:\mathbb{1}} C^T(\mathsf{inl}\,\star) \to C(\alpha_X(\mathsf{inl}\,\star))\right) \times \left(\prod_{x:X} C^T(\mathsf{inr}\, x) \to C(\alpha_X(\mathsf{inr}\, x))\right) \simeq \tag{4.3}$$

$$\left(\prod_{\star:\mathbb{1}} \mathbb{1} \to C(\alpha_X(\mathsf{inl}\,\star))\right) \times \left(\prod_{x:X} C(x) \to C(\alpha_X(\mathsf{inr}\, x))\right) \simeq \tag{4.4}$$

$$C(\alpha_X(\mathsf{inl}\,\star)) \times \left(\prod_{x:X} C(x) \to C(\alpha_X(\mathsf{inr}\, x))\right) \tag{4.5}$$

which, using our reading of $\alpha_X(\mathsf{inl}\,\star)$ as 0 and $\alpha_X \circ \mathsf{inr}$ as $\mathsf{succ}$, matches our expectations. For every polynomial functor (the ones that can be encoded as $\mathsf{W}$-types), there is a mechanical way of obtaining such an induction hypothesis constructor $(-)^T : \prod_{(X,\alpha_X):T\text{-Alg}}(X \to \mathcal{U}) \to (T_0 X \to \mathcal{U})$. But for arbitrary endofunctors, it can get tricky.

In fact, if we wanted to read more in this direction, we could visualize $\prod_{t:T_0X} C^T(t) \to C(\alpha_X t)$ as the homotopy in the following diagram:

$$\begin{tikzcd}
T_0X \arrow[d, "\alpha_X"'] \arrow[dr, "C^T"] \arrow[d, phantom, shift left=4ex, "\swarrow"] & \\
X \arrow[r, "C"'] & \mathcal{U}
\end{tikzcd}$$

By squinting our eyes, we could think of:

$$\begin{tikzcd}
T_0X \arrow[r, "T_1C"] \arrow[d, "\alpha_X"'] \arrow[dr, phantom, "\swarrow"] & T_0\mathcal{U} \arrow[d, "\alpha_{\mathcal{U}}"] \\
X \arrow[r, "C"'] & \mathcal{U}
\end{tikzcd} \tag{4.6}$$

Unfortunately, this is nonsense, as $C$ and $\mathcal{U}$ are out of range for $T_1$ and $T_0$, respectively—neither of them is in $\mathcal{U}$. Nevertheless, if we persist on this thought, and mechanically apply $T_0$ and $T_1$, we can redefine $C^T :\equiv \alpha_{\mathcal{U}} \circ T_1C$ and one is left with imposing a $T$-algebra structure on $\mathcal{U}$. Surprisingly enough, this is relatively easy for all the types seen so far. The reader can check that $[\lambda \star . \mathbb{1}, \mathsf{id}_{\mathcal{U}}]$ will work for $\mathbb{N}$, $\lambda(a, f). \prod_{b:B(a)} fb$ for $\mathsf{W}_AB$, and $\lambda z.\mathbb{1}$ for $A + B$. Despite the fact that this construction is impossible, this homotopy can be read as an adaptation of the compatibility condition that we impose on algebra morphisms. In a way, then, the induction principle is comparable to the recursion principle: given a type family that behaves "like a morphism" (by having a homotopy like the one in 4.6), there is a dependent morphism from the initial algebra to this one.

In any case, pursuing this avenue does not make things any easier: if we want to work with a general endofunctor, we cannot allow $C^T$ in our definitions, as long as we cannot characterize it using $T_0$ and $T_1$. To find inspiration, we inspect the proofs of Chapter 3 to see if we can salvage the situation. Indeed, in all three of the key results (Lemma 3.3, Lemma 3.5, and Theorem 3.6) we consider an arbitrary term $(\alpha, i) : \mathsf{W}_A^{n+1}B$, and we prove a statement about $\alpha$ by induction using $i$. And, in all three cases, the statement we prove is that $f\alpha = g\alpha$ for some functions $f$ and $g$. We can define a third property—besides naturality and inductivity—that is tailored to this specific task.

## 4.3 Initiality

We need to strike a balance. This property has to be:

- Weak enough so that it can be proved for the canonical elements (those introduced by $\alpha$).
- Strong enough so that it can be used to equalize two functions, as we did for the theorems in the previous chapter.

Looking at the requirement of equalizing functions, for given $(X, \alpha_X) : T$-Alg and $x : X$, we can think of the following:

$$P_{(X,\alpha_X)}(x) :\equiv \prod_{Y:\mathcal{U}} \prod_{f,g:X\to Y} fx = gx$$

Which states that any two functions from $X$ to $Y$ have the same image at point $x$. Unfortunately that is too strong, as, for example, taking $f, g : X \to \mathbb{2}$ defined as $f :\equiv \lambda x.0_{\mathbb{2}}$ and $g :\equiv \lambda x.1_{\mathbb{2}}$ clearly shows that the subtype $\sum_{x:X} P_{(X,\alpha_X)}(x)$ will be empty.

We can refine this a bit, equalizing only morphisms and not all functions:

$$P_{(X,\alpha_X)}(x) :\equiv \prod_{(Y,\alpha_Y):T\text{-Alg}} \prod_{f,g:(X,\alpha_X)\to(Y,\alpha_Y)} fx = gx$$

where by $(X, \alpha_X) \to (Y, \alpha_Y)$, we now mean the *encoding* of morphisms inside the type theory, i.e. the type:

$$(X, \alpha_X) \to (Y, \alpha_Y) :\equiv \sum_{f:X\to Y} \alpha_Y \circ T_1 f \sim f \circ \alpha_X$$

And by abuse of notation we write $f : (X, \alpha_X) \to (Y, \alpha_Y)$ instead of $(f, p) : (X, \alpha_X) \to (Y, \alpha_Y)$. Observe that, still, the product type takes two morphisms but the statement is that the underlying functions are equal at $x$. This is by design. It is a strong enough property to prove that $f$ and $g$ are equal as functions; we need not prove them equal as morphisms inside the type theory.

With this definition we can go quite far. As always, before proving anything we set up a battery of auxiliary functions and some lemmas.

**Definition 4.5.** *Given a $T$-algebra $(X, \alpha_X)$, we define the **initiality** as a property of $(X, \alpha_X)$:*

$$\begin{aligned}\mathsf{Init} &: \prod_{(X,\alpha_X):T\text{-}Alg} X \to \mathcal{U}\\ \mathsf{Init}_{(X,\alpha_X)}(x) &:\equiv \prod_{(Y,\alpha_Y):T\text{-}Alg} \prod_{f,g:(X,\alpha_X)\to(Y,\alpha_Y)} fx = gx\end{aligned}$$

We will omit the first argument of $\mathsf{Init}$ whenever it can be properly inferred from the type of the second one.

Defining the hierarchy of types $A_n$ is a bit trickier this time. We state it like this:

**Proposition 4.2.** *We can define:*

$$\begin{aligned}A_0 &: \mathcal{U}\\ \alpha_0 &: T_0 A_0 \to A_0\\ \rho_0 &: A_0 \to A_0\\ \mathsf{rec}_0 &: \prod_{(X,\alpha_X):T\text{-}Alg} A_0 \to X\end{aligned}$$

*And for every $n : \mathbb{N}$:*

$$\begin{aligned}
A_{n+1} &: \mathcal{U} \\
\alpha_{n+1} &: T_0 A_{n+1} \to A_{n+1} \\
\pi_n &: A_{n+1} \to A_n \\
\rho_{n+1} &: A_{n+1} \to A_0 \\
\mathsf{rec}_{n+1} &: \prod_{(X,\alpha_X):T\text{-}Alg} A_{n+1} \to X \\
l_n &: \prod_{(Y,\alpha_Y):T\text{-}Alg} \prod_{f,g:(A_n,\alpha_n)\to(Y,\alpha_Y)} f \circ \pi_n \sim g \circ \pi_n
\end{aligned}$$

Most of these are part of our usual arsenal. The newcomer, $l$, is a lemma useful in constructing the $\alpha_n$ and later on in proving properties of the idempotent. Instead of proving everything in one fell swoop, we construct each family of terms at once, assuming that the previous ones have been defined up to a given $n$.

**Definition 4.6.**

$$\begin{aligned}
A_0 &:\equiv \prod_{(X,\alpha_X):T\text{-}Alg} X \\
A_{n+1} &:\equiv \sum_{a:A_n} \mathsf{Init}_{(A_n,\alpha_n)}(a)
\end{aligned}$$

**Definition 4.7.** *Projection to the immediately underlying type:*

$$\begin{aligned}
\pi &: \prod_{n:\mathbb{N}} A_{n+1} \to A_n \\
\pi_n &:\equiv \mathsf{pr}_1
\end{aligned}$$

*Projection to the bottom of the hierarchy:*

$$\begin{aligned}
\rho &: \prod_{n:\mathbb{N}} A_n \to A_0 \\
\rho_0 &:\equiv \mathsf{id}_{A_0} \\
\rho_{n+1} &:\equiv \rho_n \circ \pi_n
\end{aligned}$$

*Elimination into arbitrary $T$-algebras:*

$$\begin{aligned}
\mathsf{rec} &: \prod_{n:\mathbb{N}} \prod_{(X,\alpha_X):T\text{-}Alg} A_n \to X \\
\mathsf{rec}_n(X,\alpha_X) &:\equiv \mathsf{rev}(X,\alpha_X) \circ \rho_n
\end{aligned}$$

The following lemma will represent our term $l$:

**Lemma 4.3** (Initiality)**.** *For every $n : \mathbb{N}$, every $(Y,\alpha_Y) : T\text{-}Alg$, and every $f, g : (A_n, \alpha_n) \to (Y, \alpha_Y)$, $f \circ \pi_n \sim g \circ \pi_n$.*

*Proof.* We want to prove the statement:

$$\prod_{(Y,\alpha_Y):T\text{-Alg}} \prod_{f,g:(A_n,\alpha_n)\to(Y,\alpha_Y)} \prod_{(a,i):A_{n+1}} (f \circ \pi_n)(a,i) = (g \circ \pi_n)(a,i)$$

Which is equivalent to any of the following:

$$\prod_{(a,i):A_{n+1}} \prod_{(Y,\alpha_Y):T\text{-Alg}} \prod_{f,g:(A_n,\alpha_n)\to(Y,\alpha_Y)} (f \circ \pi_n)(a,i) = (g \circ \pi_n)(a,i)$$

$$\prod_{(a,i):A_{n+1}} \prod_{(Y,\alpha_Y):T\text{-Alg}} \prod_{f,g:(A_n,\alpha_n)\to(Y,\alpha_Y)} fa = ga$$

$$\prod_{(a,i):A_{n+1}} \mathsf{Init}_{(A_n,\alpha_n)}(a)$$

Which is witnessed by $\lambda(a,i).i$. □

In other words, all morphisms with domain $(A_n, \alpha_n)$ that factor through $\pi_n$ are homotopic.

**Lemma 4.4.** *For every $n : \mathbb{N}$, there is a function $\alpha_n : T_0 A_n \to A_n$ such that each $\pi_n : A_{n+1} \to A_n$ is a morphism.*

*Proof.* For $n \equiv 0$, we want to define $\alpha_0 : T_0 A_0 \to A_0$. Observe that, for every $(X, \alpha_X)$, $\mathsf{rev}(X, \alpha_X) : A_0 \to X$, and hence $T_1\,\mathsf{rev}(X, \alpha_X) : T_0 A_0 \to T_0 X$. Composing appropriately with $\alpha_X : T_0 X \to X$, we obtain the desired type:

$$\alpha_0 : T_0 \left( \prod_{(X,\alpha_X):T\text{-Alg}} X \right) \to \left( \prod_{(X,\alpha_X):T\text{-Alg}} X \right)$$
$$\alpha_0 :\equiv \lambda t.\lambda(X,\alpha_X).(\alpha_X \circ T_1\,\mathsf{rev}(X,\alpha_X))t$$

For $\alpha_{n+1}$, we want the following function:

$$\alpha_{n+1} : T_0 \left( \sum_{a:A_n} \prod_{(Y,\alpha_Y):T\text{-Alg}} \prod_{f,g:(A_n,\alpha_n)\to(Y,\alpha_Y)} fa = ga \right) \to \left( \sum_{a:A_n} \prod_{(Y,\alpha_Y):T\text{-Alg}} \prod_{f,g:(A_n,\alpha_n)\to(Y,\alpha_Y)} fa = ga \right)$$

Using a similar trick as with $\alpha_0$, we can use $T_1\pi_n : T_0 A_{n+1} \to T_0 A_n$, and then apply $\alpha_n : T_0 A_n \to A_n$ to obtain the first component for $\alpha_{n+1}$. Another way to arrive to this result is by considering the conditions we want. For the following proofs, we need that $\pi_n$ be a morphism. That means that the following diagram commute:

$$\begin{tikzcd}
T_0 A_{n+1} \arrow[r, "T_1\pi_n"] \arrow[d, "\alpha_{n+1}"'] & T_0 A_n \arrow[d, "\alpha_n"] \\
A_{n+1} \arrow[r, "\pi_n"'] & A_n
\end{tikzcd}$$

So, this gives us $\alpha_n \circ T_1\pi_n$ as a candidate for the first component of $\alpha_{n+1}$.

This leaves us with the task of proving, for any $t : T_0 A_{n+1}$:

$$\prod_{(Y,\alpha_Y):T\text{-Alg}} \prod_{f,g:(A_n,\alpha_n)\to(Y,\alpha_Y)} (f \circ \alpha_n \circ T_1\pi_n)t = (g \circ \alpha_n \circ T_1\pi_n)t$$

Because $f$ and $g$ are morphisms of $T$-algebras, this goal is equivalent to

$$\prod_{(Y,\alpha_Y):T\text{-Alg}} \prod_{f,g:(A_n,\alpha_n)\to(Y,\alpha_Y)} (\alpha_Y \circ T_1 f \circ T_1\pi_n)t = (\alpha_Y \circ T_1 g \circ T_1\pi_n)t$$

By functoriality of $T_1$:

$$\prod_{(Y,\alpha_Y):T\text{-Alg}} \prod_{f,g:(A_n,\alpha_n)\to(Y,\alpha_Y)} (\alpha_Y \circ T_1(f \circ \pi_n))t = (\alpha_Y \circ T_1(g \circ \pi_n))t$$

Finally, by Lemma 4.3, $f \circ \pi_n \sim g \circ \pi_n$. □

Now that we have concluded all the definitions in Proposition 4.2, we can continue so:

**Lemma 4.5.** $\mathsf{rec}_n(X, \alpha_X)$ *is a morphism of* $T$*-Alg for every* $n : \mathbb{N}$ *and* $(X, \alpha_X) :$ $T$*-Alg.*

*Proof.* By its definition, it is sufficient to show that $\mathsf{rev}(X, \alpha_X)$ and $\rho_n$ are morphisms, which at the same time is equivalent to showing that $\mathsf{rev}(X, \alpha_X)$ and $\pi_n$ are morphisms.

For $\mathsf{rev}(X, \alpha_X)$, we need to see that $\alpha_X \circ T_1\, \mathsf{rev}(X, \alpha_X) \sim \mathsf{rev}(X, \alpha_X) \circ \alpha_0$, i.e. $(\alpha_X \circ T_1\, \mathsf{rev}(X, \alpha_X))t = (\mathsf{rev}(X, \alpha_X) \circ \alpha_0)t$ for every $t : T_0A_0$.

$$\begin{aligned}
&(\mathsf{rev}(X, \alpha_X) \circ \alpha_0)t && \equiv \quad \text{(definition of } \alpha_0\text{)}\\
&(\mathsf{rev}(X, \alpha_X) \circ (\lambda t.\lambda(X, \alpha_X).(\alpha_X \circ T_1\, \mathsf{rev}(X, \alpha_X))t))t && \equiv \quad \text{(function application)}\\
&(\mathsf{rev}(X, \alpha_X)(\lambda(X, \alpha_X).(\alpha_X \circ T_1\, \mathsf{rev}(X, \alpha_X))t)) && \equiv \quad \text{(function application)}\\
&(\alpha_X \circ T_1\, \mathsf{rev}(X, \alpha_X))t
\end{aligned}$$

For $\pi_n$, we defined $\alpha_{n+1}$ to satisfy this specific property. We want to see $\alpha_n \circ T_1\pi_n \sim \pi_n \circ \alpha_{n+1}$. This is immediate: the first component of $\alpha_{n+1}t$ is defined as $(\alpha_n \circ T_1\pi_n)t$. □

Two out of the three proofs that we need follow immediately:

**Lemma 4.6.** *For every* $n : \mathbb{N}$*, every* $(X, \alpha_X), (Y, \alpha_Y) :$ $T$*-Alg, and every* $f :$ $(X, \alpha_X) \to (Y, \alpha_Y)$*,* $f \circ \mathsf{rec}_{n+1}(X, \alpha_X) \sim \mathsf{rec}_{n+1}(Y, \alpha_Y)$*.*

*Proof.* By definition, $\mathsf{rec}_{n+1}(X, \alpha_X) \equiv \mathsf{rev}(X, \alpha_X) \circ \rho_{n+1} \equiv \mathsf{rev}(X, \alpha_X) \circ \pi_0 \circ \cdots \circ \pi_n$. The result follows then from Lemma 4.3. □

Let $\varphi :\equiv \mathsf{rec}_2(A_2, \alpha_2)$.

**Lemma 4.7.** $\varphi$ *is a pre-idempotent.*

*Proof.* The objective is $I : \varphi \circ \varphi \sim \varphi$. First let us prove $I' : \varphi \circ \mathsf{rec}_1(A_2, \alpha_2) \sim \mathsf{rec}_1(A_2, \alpha_2)$. Observe that $\mathsf{rec}_1(A_2, \alpha_2)$ factors through $\pi_0$, and apply Lemma 4.3. Then, $I :\equiv I' \circ \pi_1$. □

Regrettably, the final result does not follow. To prove that $\varphi$ is a quasi-idempotent (and thus, it splits), we need to see that $\mathsf{ap}_\varphi \circ I \sim I \circ \varphi$. Once again one can observe that both functions factor through $\pi_1$ as $\mathsf{ap}_\varphi \circ I' \circ \pi_1 \sim I \circ \mathsf{rec}_1(A_2, \alpha_2) \circ \pi_1$. But the remaining parts, $\mathsf{ap}_\varphi \circ I'$ and $I \circ \mathsf{rec}_1(A_2, \alpha_2)$, are not necessarily morphisms. We do not have a canonical $T$-algebra structure in their codomains, and in fact they are dependent functions and thus we would need a different definition to accommodate them. This is unfortunate, as, a priori, it seems like initiality captures well the spirit of the induction principle, but instead of making it literally, as inductivity does, it does it through one of the characterizations of Theorem 1.4, in such a way that we need not define the action $C^T$ for each endofunctor as described in Section 4.2.

We can also strengthen initiality more by writing:

$$\mathsf{Init}_{(X,\alpha_X)}(x) :\equiv \prod_{(Y,\alpha_Y):T\text{-Alg}} \prod_{f,g:(X,\alpha_X)\to(Y,\alpha_Y)} \mathsf{isContr}(fx = gx)$$

In which we do not only ask for a proof $p : fx = gx$, but also that any other such proof $q : fx = gx$ satisfies $p = q$. This is because $\mathsf{isContr}(A)$ implies $\mathsf{isContr}(x = y)$ for any $x, y : A$, and in particular $\mathsf{isContr}(x = y)$ implies $x = y$. Thinking like this, $\mathsf{ap}_\varphi \circ I \sim I \circ \varphi$ would follow automatically as both functions have type $\mathsf{ap}_\varphi \circ I, I \circ \varphi : \varphi \circ \varphi \circ \varphi \sim \varphi \circ \varphi$, which can be proven again by initiality almost exactly as in Lemma 4.7, and hence $\mathsf{ap}_\varphi \circ I = I \circ \varphi$ by contractibility. This time, though, the property is too strong to construct each $\alpha_{n+1}$ from the previous one.

Finally, one could also restrict $Y$ to $\mathsf{Set}$, whereby $fx = gx$ is a proposition, and so as soon as it is inhabited it is contractible. Of course, this leads to a weakening of the result equivalent to that seen in Chapter 2.

# Chapter 5

# Conclusions

## 5.1 Endofunctors

Although we have presented a few advances, the question remains on whether it can be proven that every endofunctor has an initial algebra. Efforts in this direction are the clear next steps after this thesis. Attempts to adapt initiality have seemed promising but not fruitful so far.

Another path to pursue is that of the generalized inductivity for endofunctors. This way has the key drawback that, together with the action on objects ($T_0$) and on morphisms ($T_1$), we need an action on "type families" $(-)^T : \prod_{X:\mathcal{U}}(X \to \mathcal{U}) \to (T_0X \to \mathcal{U})$, which does not seem to naturally follow from the other two, and one would have to impose a few constraints on such function to reach the splitting of an idempotent.

## 5.2 Larger universes

An observation made along the way is that, when we construct the coproduct, there seems to be no universe size restriction on the types $A$ and $B$, due to impredicativity. By taking the unary version of the coproduct—i.e. that given by a single constructor $A \to X$—we obtain some sort of coercion of $A : \mathcal{U}_i$ into $\mathcal{U}$. This has a full elimination principle into $\mathcal{U}$: given a term $a : A$, there is a corresponding $|a| : \|A\|$, where $\|A\| : \mathcal{U}$, such that every $f : A \to B$ that fits in $\mathcal{U}$ factors through $|-|$. This opens up a bunch of avenues for future work. For example, this could imply that it is not possible to expand our elimination principle into higher universes, as it would allow us to recreate Girard's paradox.

To see why, assume we have a such unary coproduct $\|-\|$ for any large type, and it comes with a full elimination principle into large types as well. Then, by taking $\|\mathcal{U}\|$, we obtain two functions, $|-| : \mathcal{U} \to \|\mathcal{U}\|$, and $\mathsf{rec}(\mathcal{U}, \mathsf{id}_{\mathcal{U}}) : \|\mathcal{U}\| \to \mathcal{U}$. If we show they are quasi-inverses, then we will have an equivalence $\mathcal{U} \simeq \|\mathcal{U}\| : \mathcal{U}$, which would incur type-in-type paradoxes. On the one hand, $\mathsf{rec}(\mathcal{U}, \mathsf{id}_{\mathcal{U}}) \circ |-| = \mathsf{id}_{\mathcal{U}}$ is just the $\beta$-rule. On the other hand, $|-| \circ \mathsf{rec}(\mathcal{U}, \mathsf{id}_{\mathcal{U}}) = \mathsf{id}_{\|\mathcal{U}\|}$ is just observing that the following diagram commutes—the first two arrows compose to $\mathsf{id}_{\mathcal{U}}$, again by the $\beta$-equality:

$$\begin{tikzcd}
\mathcal{U} \arrow[r, "|-|"] \arrow[ddr, "|-|"'] & \|\mathcal{U}\| \arrow[d, "\mathsf{rec}(\mathcal{U}{,}\mathsf{id}_{\mathcal{U}})"] \\
& \mathcal{U} \arrow[d, "|-|"] \\
& \|\mathcal{U}\|
\end{tikzcd}$$

And that there is a unique vertical arrow $\|\mathcal{U}\| \to \|\mathcal{U}\|$ that makes the diagram commutative. Because $\mathsf{id}_{\|\mathcal{U}\|}$ also satisifies this property, $|-| \circ \mathsf{rec}(\mathcal{U}, \mathsf{id}_{\mathcal{U}}) = \mathsf{id}_{\|\mathcal{U}\|}$. Hence, our best hope—which we already achieved—is to eliminate into types in $\mathcal{U}$.

## 5.3 Higher inductive types

Another clear path of work is on higher inductive types. These are similar to inductive types, but they also admit constructors that generate elements in the path types over $X$, instead of just elements of $X$ itself. For example, the circle $\mathbb{S}_1$ given by a point $x : \mathbb{S}_1$ and a path $p : x = x$ is a higher inductive type. In [1], there is a realization of $\mathbb{S}^1$ using naturality, which can eliminate into 1-types (instead of just 0-types as the construction seen in Chapter 2). Nonetheless, this requires adding another property on top of naturality. If one wanted to eliminate into even higher types in such way, it would be necessary to add higher and higher coherence conditions. Of course, the technique of Chapter 3 removes the restriction on the h-level of the types that we can eliminate to, but it is only adapted to inductive types that can be built as endofunctors of the universe $\mathcal{U}$, which higher inductive types cannot, because some of the constructors have as codomain path types instead of the type itself. Nevertheless, further study is required in this direction and Shulman's idea might be flexible enough to generalize.

## 5.4 Constructing the limit without splitting idempotents

In Section 3.2, we hinted at the idea of adding an arbitrary number of layers of inductivity to the original System F type. If one assumes $\mathbb{N}$, there is a way of constructing the limit to this succession, by way of a sequence of infinite elements enconded as a function on $\mathbb{N}$ plus a coherence condition:

$$A +_\infty B :\equiv \sum_{s:\prod_{n:\mathbb{N}} A +_n B} \prod_{n:\mathbb{N}} \pi_n(s_{n+1}) = s_n$$

$$\mathsf{inl}_\infty :\equiv \lambda a.(\lambda n.\, \mathsf{inl}_n\, a, \lambda n.\mathsf{refl}_{\mathsf{inl}_n\, a})$$

$$\mathsf{inr}_\infty :\equiv \lambda b.(\lambda n.\, \mathsf{inr}_n\, b, \lambda n.\mathsf{refl}_{\mathsf{inr}_n\, b})$$

With its recursor:

$$\mathsf{rec}_\infty : \prod_{(X,\mathsf{inl}_X,\mathsf{inr}_X):\mathcal{U}_{A+B}} A +_\infty B \to X$$

$$\mathsf{rec}_\infty :\equiv \lambda(X, \mathsf{inl}_X, \mathsf{inr}_X).\lambda(s,p).s_0(X, \mathsf{inl}_X, \mathsf{inr}_X)$$

This type can be thought of as a tuple with a countable infinity of components, indexed by $\mathbb{N}$, such that the $n$th term is the $(n+1)$th after applying $\pi_n$: $(\alpha, (\alpha, i_0), ((\alpha, i_0), i_1), \dots)$. This idea is inspired by Shulman's own splitting of the idempotent in [10, Theorem 5.3]. This gives us a way to extract arbitrarily high inductivities, but, as before, it does not allow to extract one for the whole term. In fact, there is a key difference between this and Shulman's splitting: he reverses the direction of the succession. Unfortunately, because the types of the succession are not homogeneous (each term $n$ has its own type $A +_n B$), we cannot write the terms in such a way that $\pi_n(s_n) = s_{n+1}$, because that would give an infinitely descending chain within $\mathbb{N}$. Perhaps, if there were a way to glue together all the $A +_n B$ into a single type, we could build then the endomorphism and split it directly there. All in all, this would not seem to present any advantages over the two-layer inductivity method already used.

## 5.5 Encoding the categories of algebras inside the type theory

When introducing $\mathsf{Init}$ at the end of Section 4.2, we propose building the type of all morphisms within the type theory as such:

$$(X, \alpha_X) \to (Y, \alpha_Y) :\equiv \sum_{f : X \to Y} \alpha_Y \circ T_1 f \sim f \circ \alpha_X$$

This works out well in the following proofs because they do not compare the accompanying terms, only the underlying functions. Nonetheless, they eventually provide us a result of equality between *morphisms*. How can this be? The key is in the statement of the $\eta$-rule: remember that it is postulated as $f \sim \mathsf{rec}_I(I, f \circ \alpha_I)$ for any $f : I \to X$, whether $f$ is a morphism or not. This is in contrast to the approach of directly proving that the morphisms are unique, for example used in Theorem 2.6. We just need to see that the morphisms are equal within the category, not necessarily constructed within the type system. Nonetheless, for all the results in Chapter 2, it is possible to rewrite them so as to prove that the morphisms are unique within the category implemented inside the theory. This is because the underlying types are sets, which make the property of being a morphism a proposition and hence irrelevant when comparing morphisms for equality. This, though, does not expand to further chapters when we drop the h-level limitations.

If we assume the category encoded in the theory, we *could* type our eliminators as to return morphisms instead of bare functions:

$$\begin{aligned}
\mathsf{rec}_I &: \prod_{(X, \mathsf{inl}_X, \mathsf{inr}_X) : \mathcal{U}_{A+B}} (I, \mathsf{inl}_I, \mathsf{inr}_I) \to (X, \mathsf{inl}_X, \mathsf{inr}_X) \\
\mathsf{ind}_I &: \prod_{(X, \mathsf{inl}_X, \mathsf{inr}_X) : \mathcal{U}_{A+B}^{(I, \mathsf{inl}_I, \mathsf{inr}_I)}} \prod_{(I, \mathsf{inl}_I, \mathsf{inr}_I)} (X, \mathsf{inl}_X, \mathsf{inr}_X)
\end{aligned}$$

so as to "pack" the corresponding $\beta$-equalities with each eliminator. This gives us more insight as to the relationship between elimination rules and the computation and uniqueness principles. Nonetheless, this adds complexity to the calculations and it is not really necessary for the proofs, so we remain with the simpler versions.

## 5.6 Closing words

In this thesis we have filled a missing gap in the literature regarding inductive types in impredicative type theories. We have shown that there are multiple ways to do so, and examined interesting and possibly fruitful patterns along the way. There is undoubtedly further work in this area of type theory, which would potentially have theoretical applications in mathematics and logic as well as very desirable practical applications in foundations of programming languages.

# Bibliography

[1] Steve Awodey, Jonas Frey and Sam Speight. 'Impredicative Encodings of (Higher) Inductive Types'. In: *LICS '18: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science* (2018). DOI: `10.1145/3209108.3209130`.

[2] Steve Awodey, Nicola Gambino and Kristina Sojakova. 'Homotopy-Initial Algebras in Type Theory'. In: *J. ACM* 63.6 (Jan. 2017). Place: New York, NY, USA Publisher: Association for Computing Machinery. ISSN: 0004-5411. DOI: `10.1145/3006383`. URL: `https://doi.org/10.1145/3006383`.

[3] Steve Awodey, Nicola Gambino and Kristina Sojakova. 'Inductive Types in Homotopy Type Theory'. In: *2012 27th Annual IEEE Symposium on Logic in Computer Science* (2012). DOI: `doi:10.1109/lics.2012.21`.

[4] Thierry Coquand and Christine Paulin. 'Inductively defined types'. In: *COLOG-88*. Ed. by Per Martin-Löf and Grigori Mints. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 50–66. ISBN: 978-3-540-46963-6.

[5] Jean-Yves Girard. 'Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur.' PhD Thesis. Université Paris 7, 1972.

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

[7] John C. Reynolds. 'Towards a Theory of Type Structure'. In: *Programming Symposium, Proceedings Colloque Sur La Programmation*. Berlin, Heidelberg: Springer-Verlag, 1974, pp. 408–423. ISBN: 3-540-06859-7.

[8] Egbert Rijke. *Introduction to Homotopy Type Theory*. _eprint: 2212.11082. 2022.

[9] Ivar Rummelhoff. 'Polynat in PER models'. In: *Theoretical Computer Science* 316.1 (2004), pp. 215–224. ISSN: 0304-3975. DOI: `https://doi.org/10.1016/j.tcs.2004.01.031`. URL: `https://www.sciencedirect.com/science/article/pii/S0304397504000842`.

[10] Michael Shulman. 'Idempotents in intensional type theory'. In: *Logical Methods in Computer Science* Volume 12, Issue 3 (Apr. 2017). Publisher: Centre pour la Communication Scientifique Directe (CCSD). DOI: `10.2168/lmcs-12(3:9)2016`.

[11] Michael Shulman. *Impredicative Encodings, Part 3*. Nov. 2018. URL: `https://homotopytypetheory.org/2018/11/26/impredicative-encodings-part-3/` (visited on 01/07/2023).

[12] Sam Speight. 'Impredicative Encodings of Inductive Types in Homotopy Type Theory'. MA thesis. Carnegie Mellon University, July 2017. URL: `https://raw.githubusercontent.com/sspeight93/Papers/master/impredicative-encodings-of-inductive-types-in-homotopy-type-theory.pdf`.