Impredicative Encodings of (Higher) Inductive Types

Steve Awodey
Department of Philosophy
Carnegie Mellon University
Pittsburgh, USA
awodey@cmu.edu

Jonas Frey
Department of Philosophy
Carnegie Mellon University
Pittsburgh, USA
jonasf@andrew.cmu.edu

Sam Speight
Department of Computer Science
University of Oxford
United Kingdom
samuel.speight@cs.ox.ac.uk

Abstract

Postulating an impredicative universe in dependent type theory allows System F style encodings of finitary inductive types, but these fail to satisfy the relevant η-equalities and consequently do not admit dependent eliminators. To recover η and dependent elimination, we present a method to construct *refinements* of these impredicative encodings, using ideas from homotopy type theory. We then extend our method to construct impredicative encodings of some *higher* inductive types, such as 1-truncation and the unit circle S^1.

CCS Concepts • **Theory of computation → Type theory**; *Categorical semantics*;

Keywords Impredicativity, Impredicative encodings, Inductive types, Higher inductive types, System F, Martin-Löf type theory, Homotopy type theory

1 Introduction

System F, also known as the 'Girard-Reynolds polymorphic λ-calculus', goes back to [17] and [30] (see [18, Chapter 11] for a textbook account). It extends the simply typed λ-calculus with universal quantification $\forall$ over types. Under the Curry-Howard correspondence [21], it is the type-theoretic analog of second-order propositional logic.

One of the remarkable things about System F is that it allows for the *encoding* of types such as products, sums, natural numbers, and, more generally, *finitary inductive types*. These encodings are called *impredicative*, since in defining a specific type they quantify over the totality of *all* types, which in particular contains type which is being defined. For example, the type $\mathbb{N}$ of natural numbers is encoded in System F as

$$\mathbb{N}_F \equiv \forall X. (X \to X) \to X \to X.$$

A well-known defect of such encodings, however, is that they do not satisfy the appropriate 'η-rules', which are uniqueness principles stating that for every recursive definition there is only one

function realizing it. One way to think about this failure of η is that e.g. the type described by the formula $\mathbb{N}_F$ is 'too large', in that certain models [33] may contain non-standard elements which are not generated from the constructors. On the other hand, one can show using *parametricity* arguments [31] that all the 'named' elements are standard. This discrepancy has led to attempts to *refine* the models by imposing parametricity, *dinaturality* [8], or *realizability* [9] conditions.

System F-style, impredicative encodings can also be given in dependent type theory with an *impredicative universe* (such as the *calculus of constructions* [10]); a further consequence of the failure of the η-rules, however, is then that the encoded types do not admit dependent elimination rules, which are necessary for proofs by induction, and thus indispensable for the development of mathematics in type theory.

In this article we present a new technique to restore η-rules by refining impredicative encodings in a way that is related to the parametricity and dinaturality techniques mentioned above, but in contrast to them, takes place *inside the type theory*, so that the η-equalities on the refined types become *provable*, rather than just admissible w.r.t. a model. To be more specific, for our refined encodings we can prove *propositional* versions of the required η-equalities, which – as shown in [6, 7] – are sufficient to derive the existence of the corresponding dependent eliminators.

The system of type theory that we use is similar to the calculus of constructions with a hierarchy of predicative universes (like in older versions of the Coq proof assistant), but in contrast to the calculus of constructions we assume that the lowest, impredicative universe is closed under small sigmas and identity types.

Some of our arguments make use of the 'uniqueness of identity proofs' principle, but instead of postulating it globally we exploit the notion of 0-*types* from homotopy type theory to state the relevant results for that level of the definable hierarchy of h-levels. This also allows us to explore the applicability of our techniques to *higher inductive types* in Section 5.

We emphasize that our main contribution is to give impredicative encodings of inductive types in dependent type theory that satisfy the relevant dependent elimination rules (along with the other rules). This seems to solve a long-standing problem, considered in [16], of giving "2nd-order encodings" satisfying induction. We do this by "refining" the impredicative encodings inside the type theory using identity types. This is in contrast to interpreting the system into a model and having the universal properties hold there, which is the spirit of the "parametric polymorphism" approach set out in [31], where the universal quantification of System F is "cut down" on interpretation.

Overview Section 2 specifies the system of type theory in which our work takes place, recalls the definition of n-types in homotopy type theory, and introduces the (pre-)category $\mathcal{S}et$ of small 0-types. We also include a brief description of how System F can

be translated into our setting. In Section 3 we show how to refine the System F encodings of some non-recursive, inductive sets such as the binary sum $A + B$ of two sets to recover appropriate η-rules, using an argument based on a type-theoretic version of the Yoneda lemma. Section 4 gives a related technique involving impredicative encodings of initial algebras in order to achieve the same result for genuinely inductive sets such as the type $\mathbb{N}$ of natural numbers. In Section 5 we show how these techniques generalize from sets (0-types) to 1-types, giving impredicative encodings of some of the recently introduced *higher inductive types* [40, Ch. 6]. Specifically, we give encodings for the 1-truncation and the unit circle. We believe that these encodings also illuminate the ones for conventional inductive types. Indeed, our general methodology is very much informed by the HoTT point of view. Finally, Section 6 addresses issues such as limitations of our techniques, open questions, consistency and the existence of models, and future work.

2 The System of Type Theory

Although our results are not all "higher dimensional" in nature, our work is best understood in the context of *homotopy type theory (HoTT)*; we refer to [40] as our standard reference for conventions and terminology. We thus work in a system of dependent type theory with products $\prod_{x:A} B(x)$, strong sums $\sum_{x:A} B(x)$, (intensional) identity types $\mathrm{Id}_X(x, y)$, and function extensionality, as in [40]. However, we make no use of the univalence axiom. We usually write simply $x = y$ for $\mathrm{Id}_X(x, y)$, as is now common. We then distinguish notationally between *propositional* equality $x = y$ and *definitional* equality $x \equiv y$.

Universes We augment the hierarchy of *predicative* universes $\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \dots$ assumed in [40] by adding a single *impredicative* universe $\mathcal{U}$ at the bottom,

$$\mathcal{U} : \mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \dots$$

This new universe $\mathcal{U}$ is also closed under dependent sums and identity types, like the $\mathcal{U}_i$, but instead of the usual (predicative) product formation rule

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma, x : A \vdash B : \mathcal{U}_i}{\Gamma \vdash \prod_{(x:A)} B : \mathcal{U}_i}$$

it satisfies the *impredicative* product formation rule

$$\frac{\Gamma, x : A \vdash B : \mathcal{U}}{\Gamma \vdash \prod_{(x:A)} B : \mathcal{U}}$$

which is stronger since there is no size restriction on the A.

Note that in the following, when writing $\mathcal{U}$, we always mean the *impredicative* universe – in this respect we deviate from [40], in that we do *not* use $\mathcal{U}$ as a placeholder for an unspecified $\mathcal{U}_i$.

n-Types Recall from [40, 7.1] the hierarchy of n-types (Voevodsky: "h-levels"): X is called a *(-1)-type*, or *proposition*, if it satisfies $\prod_{(x, y:X)} x = y$; it is a *0-type*, or *set*, if its identity types are always propositions; and generally, it is an *(n+1)-type* if its identity types are always n-types. Formally,

$$\mathrm{isProp}(X) :\equiv \prod_{(x, y:X)} x = y$$
$$\mathrm{isSet}(X) :\equiv \prod_{(x, y:X)} \mathrm{isProp}(x = y)$$
$$\mathrm{isType}_{(n+1)}(X) :\equiv \prod_{(x, y:X)} \mathrm{isType}_n(x = y),$$

and

$$\mathrm{Prop} :\equiv \sum_{(X:\mathcal{U})} \mathrm{isProp}(X)$$
$$\mathrm{Set} :\equiv \sum_{(X:\mathcal{U})} \mathrm{isSet}(X)$$
$$\mathrm{Type}_n :\equiv \sum_{(X:\mathcal{U})} \mathrm{isType}_n(X) \,.$$

Note that the types $\mathrm{isProp}(X)$, $\mathrm{isSet}(X)$, etc., are themselves propositions [40, Theorem 7.1.10], so that Prop, Set, etc., are *subtypes* of $\mathcal{U}$, in the sense that the first projection from the respective Σ-type is an embedding.

We normally suppress the coercion $\mathrm{pr}_1 : \mathrm{Type}_n \to \mathcal{U}$ and treat expressions of type Type_n as if they were themselves types. Thus, in particular if $X : \mathcal{U} \vdash A(X)$ is a $\mathcal{U}$-indexed family of types then the expression $\prod_{(X:\mathrm{Type}_n)} A(X)$ is a shorthand for

$$\prod_{(X:\mathrm{Type}_n)} A(\mathrm{pr}_1 X),$$

which in turn is equivalent to

$$\prod_{(X:\mathcal{U})} \mathrm{isType}_n(X) \to A(X).$$

Moreover, since the *n*-types are closed under arbitrary products [40, Theorem 7.1.9] (and suppressing the unpacking and repacking of dependent pairs), the rule

$$\frac{\Gamma, x : A \vdash B : \mathrm{Type}_n}{\Gamma \vdash \prod_{(x:A)} B : \mathrm{Type}_n}$$

is admissible for all *n*. Thus, in sum, we can view the types Type_n as *impredicative subuniverses* of $\mathcal{U}$.

We exploit the convenience of having an impredicative universe that is closed under most constructions, by working mostly inside $\mathcal{U}$ – as opposed to the usual methodology of predicative systems, where a hierarchy of universes are used "parametrically". Accordingly, we adopt the convention that the terms "proposition", "set", "n-type", etc., refer only to types in $\mathcal{U}$.

The precategory $\mathcal{S}et$ of sets in $\mathcal{U}$ The subtype Set of 0-types in $\mathcal{U}$ gives rise to a precategory $\mathcal{S}et$ where

$$\mathcal{S}et_0 = \mathrm{Set} \quad \text{and} \quad \mathrm{hom}(A, B) = (A \to B)$$

for $A, B : \mathrm{Set}$ ([40, Example 9.1.5]).

As pointed out in [40, Section 10.1.1], this precategory is (small) *complete* in that it admits equalizers (defined using Σ- and identity-types) and small products (given by type-theoretic products). Since in our setting $\mathcal{U}$ is an impredicative universe, $\mathcal{S}et$ even admits 'large' products – i.e. products indexed by arbitrary types – which we make use of in what follows.

Translation of System F There is an evident syntactic translation t from System F to our system of dependent type theory. Types of the form $A \to B$ in System F are translated to $A^t \to B^t$, and types of the form $\forall X.B$ are translated to $\prod_{X:\mathcal{U}} B^t$, where A^t and B^t are the translations of the System F types A and B. The translation of terms is equally obvious.

Similarly, we can restrict the translation by replacing $\mathcal{U}$ above with any of the impredicative subuniverses Type_n of *n*-types. For example, we may define the translation t_0 with:

$$(\forall X.B)^{t_0} :\equiv \prod_{X:\mathrm{Set}} B^{t_0}.$$

In this sense, we can speak of System F encodings of inductive types in our system of dependent type theory.

Generally, if $x{:}A \vdash P(x)$ is a (family of) propositions, then as above $\sum_{x:A} P(x)$ is a subtype of A via the first projection. Our

impredicative encodings of inductive types will be subtypes of the usual System F encodings in this sense; we say that they "sharpen" or "refine" the usual encodings.

3 Basic Set Encodings

As stated in the foregoing section, the impredicative universe $\mathcal{U}$ allows us to give 'System F style' encodings of certain inductive types. In this section we explain how these encodings fall short of the usual inductive types assumed in dependent type theory. We then indicate a way to remedy these shortcomings to a certain extent.

We start with the sum of two types A and B, whose System F encoding we translate into type theory, as explained above, by replacing the quantification over types by a dependent product over $\mathcal{U}$:

$$A +^F B \; := \; \prod_{X:\mathcal{U}} (A \to X) \to (B \to X) \to X \,. \tag{3.1}$$

It is easy to see that this encoding admits injections[1]

$$
\begin{aligned}
\mathsf{inl}^F &:= \lambda A\,B\,a\,X\,f\,g.\,f\,a \; : \; \textstyle\prod_{\{A,B:\mathcal{U}\}} A \to (A +^F B) \\
\mathsf{inr}^F &:= \lambda A\,B\,b\,X\,f\,g.\,g\,b \; : \; \textstyle\prod_{\{A,B:\mathcal{U}\}} B \to (A +^F B)
\end{aligned}
\tag{3.2}
$$

and a recursor

$$
\begin{aligned}
\mathsf{rec}^F_+ &:= \lambda A\,B\,C\,f\,g\,\phi.\,\phi\,C\,f\,g \\
&: \textstyle\prod_{\{A,B,C:\mathcal{U}\}} (A \to C) \to (B \to C) \to (A +^F B) \to C
\end{aligned}
\tag{3.3}
$$

satisfying the *definitional* β-equalities

$$\mathsf{rec}^F_+ \, f\, g\, (\mathsf{inl}^F a) \equiv f\, a \quad \text{and} \quad \mathsf{rec}^F_+ \, f\, g\, (\mathsf{inr}^F b) \equiv g\, b$$

for all $A, B, C : \mathcal{U}$, $f : A \to C$, $g : B \to C$, $a : A$ and $b : B$.

However there are several problems:

(i) the recursor only allows us to define functions into types in $\mathcal{U}$,

(ii) the η-rule

$$f = \mathsf{rec}^F_+ (f \circ \mathsf{inl}^F)(f \circ \mathsf{inr}^F), \tag{3.4}$$

where $f : A +^F B \to C$, doesn't hold, even *propositionally*,

(iii) the encoding does not admit a dependent eliminator, which would have to have the type

$$
\begin{aligned}
\mathsf{ind}^F_+ \; : \; & \textstyle\prod\{A, B : \mathcal{U}\} \\
& \textstyle\prod\{C : A +^F B \to \mathcal{U}\} \\
& \textstyle\prod(f : \prod_{(a:A)} C(\mathsf{inl}^F a)) \\
& \textstyle\prod(g : \prod_{(b:B)} C(\mathsf{inr}^F b)) \\
& \textstyle\prod(x : A +^F B), \quad C\,x.
\end{aligned}
$$

and satisfy the propositional equalities

$$\mathsf{ind}^F_+ \, f\, g\, (\mathsf{inl}^F a) \; = \; f\, a \quad \text{and} \quad \mathsf{ind}^F_+ \, f\, g\, (\mathsf{inr}^F b) \; = \; g\, b$$

for all appropriately typed f, g, a, and b.

We defer discussion of issue (i) in general to section 6 below. Issues (ii) and (iii) are related by the general theory developed in [6, 7]: briefly, in the present setting the dependent elimination rule is equivalent to the η-rule. In the following we give a way to restore the propositional η-rule for sums of sets by restricting the product in (3.1) to Set and taking a suitable subtype.

---

[1]We adopt the convention that arguments in braces $\{-\}$ in types denote *implicit* arguments, meaning that we may write e.g. $\mathsf{inl}^F a$ instead of $\mathsf{inl}^F A\,B\,a$.

3.1 Refining the encoding

Restricting the dependent product in (3.1) to the subuniverse of propositions, we obtain a well-known encoding of logical disjunction:

$$A \vee B \simeq \prod_{X:\mathsf{Prop}} (A \to X) \to (B \to X) \to X \qquad \text{for } A, B : \mathsf{Prop}$$

Observe that $A \vee B$ *is* a proposition, because Prop is closed under Π-types.

In a similar vein it seems natural to define a sum operation

$$A +^* B \; := \; \prod_{X:\mathsf{Set}} (A \to X) \to (B \to X) \to X \quad \text{for } A, B \in \mathsf{Set} \tag{3.5}$$

of sets A, B by restricting the impredicative product to types $X : \mathsf{Set}$. This type also admits injections and a recursor

$$
\begin{aligned}
\mathsf{inl}^* &:= \lambda A\,B\,a\,X\,f\,g.\,f\,a \; : \; \textstyle\prod_{\{A,B:\mathsf{Set}\}} A \to (A +^* B) \\
\mathsf{inr}^* &:= \lambda A\,B\,b\,X\,f\,g.\,g\,b \; : \; \textstyle\prod_{\{A,B:\mathsf{Set}\}} B \to (A +^* B) \\
\mathsf{rec}^*_+ &:= \lambda A\,B\,C\,f\,g\,\phi.\,\phi\,C\,f\,g \\
&\quad : \textstyle\prod_{\{A,B,C:\mathsf{Set}\}} (A \to C) \to (B \to C) \to (A +^* B) \to C
\end{aligned}
\tag{3.6}
$$

analogous to those for $A +^F B$. The η-rule still fails for $A +^* B$, but as we shall see, it is now possible to carve out a *subtype* for which it is satisfied.

As a warm-up exercise, consider first the unary case. For $A : \mathsf{Set}$ there is an embedding-retraction pair

$$
\begin{array}{ccc}
A & \xrightarrow{\;\;e\;\;} & \prod_{(X:\mathsf{Set})}(A \to X) \to X \\
& {\scriptstyle =} \searrow & \downarrow {\scriptstyle r} \\
& & A,
\end{array}
\tag{3.7}
$$

where $e(a) \equiv \lambda X f.\,f(a)$ and $r(\alpha) \equiv \alpha_A(\mathsf{id}_A)$.

Now, a term $\alpha \; : \; \prod_{X:\mathsf{Set}}(A \to X) \to X$ is a *family of maps* (switching notation for emphasis),

$$\alpha_X : X^A \to X, \qquad X : \mathsf{Set}\,.$$

We can cut down the type $\prod_{X:\mathsf{Set}}(A \to X) \to X$ to (one equivalent to) the image of e in (3.7) by requiring that the family of maps α_X be *natural in X* in the sense that for all sets X, Y and all maps $f : X \to Y$, the following square commutes.

$$
\begin{array}{ccc}
X^A & \xrightarrow{\;\alpha_X\;} & X \\
{\scriptstyle f^A} \downarrow & & \downarrow {\scriptstyle f} \\
Y^A & \xrightarrow{\;\alpha_Y\;} & Y
\end{array}
\tag{3.8}
$$

Here, $f^A \equiv \lambda g.\, f \circ g$ is the action of the functor $(-)^A : \mathcal{S}et \to \mathcal{S}et$ on $f : X \to Y$.

The *sharper encoding* A^+ of A is now:

$$A^+ := \sum_{\alpha:A^*} N(\alpha) \tag{3.9}$$

where $$A^* := \prod_{X:\mathsf{Set}} (A \to X) \to X \tag{3.10}$$

and $$N(\alpha) := \prod_{X,Y:\mathsf{Set}} \prod_{f:X\to Y} \alpha_Y \circ f^A = f \circ \alpha_X \tag{3.11}$$

Note that $\mathsf{pr}_1 : A^+ \hookrightarrow A^*$ is an embedding since $N(\alpha)$ is a proposition for all α.

Theorem 3.1 (Basic Lemma). *For any set A, we have $A \simeq A^+$.*

Proof. First, we show that $e : A \hookrightarrow A^*$ factors through $A^+ \hookrightarrow A^*$. For $a : A$, the family $e(a) : A^*$ consists of the evaluations $e(a)_X : A^X \to X$, where $g \mapsto g(a)$. If $f : X \to Y$, then indeed

$$(e(a)_Y \circ f^A)(g) = e(a)_Y(f^A(g)) = e(a)_Y(f \circ g)$$
$$= f(g(a)) = f(e(a)_X(g)) = (f \circ \alpha_X)(g).$$

Now let $\alpha : A^*$ be natural in the sense expressed in (3.9), and define $a_0 :\equiv r(\alpha) \equiv \alpha_A(1_A) : A$. We claim that $\alpha = e(a_0)$, which suffices since A^* is a set. Indeed, take any X and $g : X \to A$, then we have

$$e(a_0)_X(g) = g(a_0) = g(\alpha_A(1_A)) = \alpha_X(g^A(1_A)) = \alpha_X(g),$$

using the naturality of α in the third step.   □

Remark 3.2. The categorically minded reader will recognize that the previous theorem is an instance of the Yoneda lemma. Indeed, (3.9) is the type of all natural transformations from the (covariant) representable functor $(-)^A$ to the identity functor $I : \mathcal{S}et \to \mathcal{S}et$, so by Yoneda we indeed have

$$\mathrm{Nat}\Big((-)^A, I\Big) \simeq I(A) \simeq A.$$

But since we do not require this level of generality here, we will not develop the required details.   ◇

Taking inspiration from the previous theorem, we return to the preliminary encoding $A +^* B$ from (3.5), and in order to recover the η-rule define a subtype $A + B \hookrightarrow A +^* B$ by imposing a suitable naturality condition.

We start with the observation that if we already had such a type $A + B$, then $(A + B)^*$ would be equivalent to $(A +^* B)$:

$$(A + B)^* \equiv \prod_{X:\mathrm{Set}} ((A + B) \to X) \to X$$
$$\simeq \prod_{X:\mathrm{Set}} (A \to X) \times (B \to X) \to X \qquad (3.12)$$
$$\simeq \prod_{X:\mathrm{Set}} (A \to X) \to (B \to X) \to X \equiv A +^* B$$

Now by transporting the naturality condition of the lemma along the equivalence, we can *define* $A + B$ as a subtype of $A +^* B$, where the defining condition can again be read as a naturality property, but now one that does not assume the existence of $A+B$. Specifically, we define

$$A + B :\equiv \sum_{\alpha:A+^*B} N(\alpha) \qquad \text{where}$$
$$N(\alpha) :\equiv \prod_{X,Y:\mathrm{Set}} \prod_{f:X\to Y} \prod_{\substack{h:A\to X \\ k:B\to X}} f(\alpha_X hk) = \alpha_Y(f\circ h)(f\circ k).$$
$$(3.13)$$

If we substitute $(\mathrm{inl}^* a)$ or $(\mathrm{inr}^* b)$ for α in $N(\alpha)$, the two sides of the identity type become definitionally equal, whence we can refine the injections defined in (3.2) to get the following.

$$\mathrm{inl} : \prod_{\{A,B:\mathrm{Set}\}} A \to A + B \qquad \mathrm{inr} : \prod_{\{A,B:\mathrm{Set}\}} B \to A + B$$
$$\mathrm{inl}\, a :\equiv (\mathrm{inl}^* a, \lambda XYfhk.\,\mathrm{refl}) \quad \mathrm{inr}\, b :\equiv (\mathrm{inr}^* b, \lambda XYfhk.\,\mathrm{refl})$$

The recursor (3.3) gets replaced by

$$\mathrm{rec}_+ : \prod_{\{A,B,C:\mathrm{Set}\}} (A \to C) \to (B \to C) \to A + B \to C$$
$$\mathrm{rec}_+ :\equiv \lambda ABCfg\,\xi.\,(\mathrm{pr}_1\,\xi)_C\,f\,g$$

With these definitions we can now prove the following.

Theorem 3.3. *For all sets A and B, the encoding (3.13) of the sum $A + B$, along with the structure* inl*,* inr*, and* rec_+ *just defined, satisfy*

(i) *the* definitional *β-rules*

$$\mathrm{rec}_+ f\, g\, (\mathrm{inl}\, a) \equiv f\, a \quad \text{and} \quad \mathrm{rec}_+ f\, g\, (\mathrm{inr}\, b) \equiv g\, b$$

for all C : Set*, $f : A \to C$, $g : B \to C$, $a : A$, $b : B$,*

(ii) *the* propositional *η-rule*

$$\mathrm{rec}_+(f \circ \mathrm{inl})(f \circ \mathrm{inr}) = f$$

for all C : Set *and $f : A + B \to C$.*

Proof. The β rules follow mechanically by unfolding definitions. For the η rule, we first prove a special case, namely

$$\mathrm{rec}_+\,\mathrm{inl}\,\mathrm{inr} = \mathrm{id}_{A+B} : A + B \to A + B. \qquad (3.14)$$

By function extensionality and Σ-induction it is sufficient to show that $\alpha_{A+B}\,\mathrm{inl}\,\mathrm{inr} = (\alpha, p)$ for all $\alpha : A +^* B$ and $p : N(\alpha)$. Since $A+B \hookrightarrow A+^* B$ is an embedding, this reduces to $\mathrm{pr}_1(\alpha_{A+B}\,\mathrm{inl}\,\mathrm{inr}) = \alpha$, and again by function extensionality this follows from

$$\mathrm{pr}_1(\alpha_{A+B}\,\mathrm{inl}\,\mathrm{inr})_X\,f\,g = \alpha_X\,f\,g$$

for $X :$ Set, $f : A \to X$ and $g : B \to X$. This is shown by

$$\mathrm{pr}_1(\alpha_{A+B}\,\mathrm{inl}\,\mathrm{inr})_X\,f\,g \equiv \mathrm{rec}_+ fg(\alpha_{A+B}\,\mathrm{inl}\,\mathrm{inr})$$
$$= \alpha_X(\mathrm{rec}_+ fg \circ \mathrm{inl})(\mathrm{rec}_+ fg \circ \mathrm{inr})$$
$$= \alpha_X fg,$$

where the second equality is given by $p(\mathrm{rec}_+ fg)\,\mathrm{inl}\,\mathrm{inr}$ and the third one follows from β and function extensionality.

For the general case, let again $\alpha : A +^* B$ and $p : N(\alpha)$. We have:

$$\mathrm{rec}_+(f \circ \mathrm{inl})(f \circ \mathrm{inr})(\alpha, p) \equiv \alpha_C(f \circ \mathrm{inl})(f \circ \mathrm{inr})$$
$$= f(\alpha_C(\mathrm{inl})(\mathrm{inr})) \qquad (\text{by } p)$$
$$\equiv f(\mathrm{rec}_+\,\mathrm{inl}\,\mathrm{inr}\,(\alpha, p))$$
$$= f(\alpha, p) \qquad (\text{by } (3.14)),$$

which proves the claim.   □

We emphasize that it is crucial to the proof that the encoding (3.13) of $A + B$ is itself a set, so that $A + B$ is in the range of the variable $X :$ Set – and of course, so that the sum of two sets is again a 0-type. This is ensured by the fact that the n-types are closed under Σ-, Π-, and identity-types, and of course, the impredicativity of $\mathcal{U}$. In more detail, in (3.5), the X ranges over sets, and thus the type $(A \to X) \to (B \to X) \to X$ is a 0-types. But then by impredicativity of Set, the entire type $A +^* B$ is a set. In $N(\alpha)$, the identity type is a proposition, since it is over the type of functions $(A \to X) \to (B \to X) \to Y$, which is a 0-type. Thus $N(\alpha)$ is itself a proposition, whence $A + B$ is a set.

Finally, as mentioned in Section 3, the *induction principle* below follows from recursion together with the uniqueness of the recursor (the η-rule) [6, 7]:

$$\mathrm{ind}_+ : \prod\{A, B : \mathrm{Set}\}$$
$$\prod\{C : A + B \to \mathrm{Set}\}$$
$$\prod(f : \prod_{(a:A)} C(\mathrm{inl}\, a))$$
$$\prod(g : \prod_{(b:B)} C(\mathrm{inr}\, b))$$
$$\prod(x : A + B), \quad C\, x,$$

with propositional equalities

$$\mathrm{ind}_+ f\, g\, (\mathrm{inl}\, a) = f\, a \quad \text{and} \quad \mathrm{ind}_+ f\, g\, (\mathrm{inr}\, b) = g\, b$$

for all appropriately typed f, g, a, and b.

Having done the unary and binary case of sums of sets, we might as well do the nullary one, too. The System F style encoding of the empty type $\mathbf{0}$ in $\mathcal{U}$, given by

$$\mathbf{0} \;:\equiv\; \textstyle\prod_{(X:\mathcal{U})} X \,,$$

admits a recursor

$$\mathrm{rec}_{\mathbf{0}} \;:\equiv\; \lambda X c.\, c_X \;:\; \textstyle\prod_{\{X:\mathcal{U}\}} \mathbf{0} \to X \,,$$

and it turns out that in this case we don't need any refinement, since we can already derive that $\mathbf{0}$ is a proposition and that $\mathrm{rec}_{\mathbf{0}}$ satisfies an η equality – indeed, we have

$$\lambda cd.\, c(c = d) : \prod_{c,d:\mathbf{0}} (c = d) \;\equiv\; \mathrm{isProp}(\mathbf{0})\,,$$

and the eliminator $\mathrm{rec}_{\mathbf{0}}$ is unique, by

$$\mathrm{funext}(\lambda c.\, c(c_X = fc)) \;:\; \mathrm{rec}_{\mathbf{0}} X = f$$

for any $X : \mathcal{U}$ and $f : \mathbf{0} \to X$.

Other non-recursive 0-types Unlike $\mathbf{0}$, the terminal set $\mathbf{1}$ does not have the System F form $\mathbf{1}_F :\equiv \prod_{X:\mathcal{U}} X \to X$ (uniqueness of the maps $X \to \mathbf{1}_F$ fails), but instead can be encoded as a (-1)-type via the familiar

$$\mathbf{1} \;\equiv\; \prod_{p:\mathrm{Prop}} p \to p \,. \tag{3.15}$$

Indeed, this is easily seen to be terminal for all $X : \mathcal{U}$. (Another possibility would be to define $\mathbf{1}$ by quantifying over Set and adding a naturality condition, but in this case it is sufficient to quantify over Prop since $\mathbf{1}$ itself is a proposition).

The method of adding a naturality condition can be used to encode the *set-truncation* $\|A\|_0$ of an arbitrary type A. Indeed, we can simply take

$$\|A\|_0 \;:\equiv\; \sum_{\alpha:A^*} \prod_{X,Y:\mathrm{Set}} \prod_{f:X\to Y} \alpha_Y \circ f^A = f \circ \alpha_X \,,$$

$$\text{where} \quad A^* \;:\equiv\; \prod_{X:\mathrm{Set}} (A \to X) \to X \,, \tag{3.16}$$

as in the Basic Lemma 3.1, since A^* is a set even when A is not one.

Observe how this generalizes the well-known [4, 19] [40, Exercise 3.15] *propositional truncation* of a type A,

$$\|A\| \;=\; \prod_{X:\mathrm{Prop}} (A \to X) \to X \,.$$

We only mention that it is also possible to give correct impredicative encodings of set-quotients [40, 6.10], as well as general coequalizers of sets, by related methods.

4 General inductive sets

While sums and truncations are viewed as inductive types in modern terminology, the classical idea of an inductive type involves generation from constants by repeated application of constructors. A well understood class of inductive types in type theory and category theory are *W-types* [26, pg. 43], which are generated from a family of constructors of specified – possibly infinite – arities.

Inductive types of this kind with only *finitely* many constructors, each of finite arity, can be encoded in System F [18, Section 11.5], and translating these encodings into type theory by quantifying over the impredicative universe Set leads again to types which admit the correct constructors and recursors but fail to satisfy the appropriate η-rules.

On the other hand, it is known from category theory that W-types can be understood as initial algebras of so called *polynomial functors* [15, 27]. In the following we show how a categorical construction of initial algebras relying on the impredicativity of Set gives rise to subtypes of the System F style encodings satisfying η. We elaborate this idea using as running example the inductive type of natural numbers, but the method is easily seen to generalize.

4.1 Initial algebras of endofunctors

As pointed out in Section 2, the precategory $\mathcal{S}et$ is complete in the very strong sense that it has all equalizers (constructed using Σ- and identity-types) as well as products of families of objects indexed by *arbitrary* types. It is an old observation by Hyland [22, Section 3.1] that this implies the existence of initial algebras for arbitrary endofunctors, at least in the related semantical setting of certain kinds of internal categories. In the following we give explicit type-theoretic descriptions of the required limits and initial algebras by unwinding the categorical definitions.

First, observe that limits over arbitrary (pre)category-indexed diagrams can be expressed using products and equalizers, as usual. Specifically, let $\mathbb{J}$ be a precategory: an arbitrary type of objects $\mathbb{J}_0$ and a family of *sets* of arrows $\hom : \mathbb{J}_0 \times \mathbb{J}_0 \to \mathrm{Set}$, equipped with the usual composition and unit structure, and satisfying the usual equations on these, which are *propositional*, because the $\hom(i,j)$ are sets for all $i,j : \mathbb{J}_0$. A $\mathbb{J}$-indexed 'diagram' is just a functor $D : \mathbb{J} \to \mathcal{S}et$, which can also be defined as usual, since the values D_i are all sets. The limit of D is the equalizer of the two maps

$$p,q \;:\; \prod_{i:\mathbb{J}_0} D_i \;\rightrightarrows\; \prod_{(i,j:\mathbb{J}_0)} \prod_{(u:\hom(i,j))} D_j$$
$$p\,\phi\,i\,j\,u \;:\equiv\; D(u)(\phi_i)$$
$$q\,\phi\,i\,j\,u \;:\equiv\; \phi_j \,,$$

which is given explicitly by the type

$$\varprojlim_i D_i \;:\equiv\; \sum_{(\phi:D^*)} \prod_{(i,j:\mathbb{J}_0)} \prod_{(u:\hom(i,j))} D_u(\phi_i) = \phi_j\,,$$

$$\text{where} \quad D^* \;:\equiv\; \prod_{i:\mathbb{J}_0} D_i \,, \tag{4.1}$$

together with projections

$$\pi_j \;:\equiv\; \lambda\xi.\,(\mathrm{pr}_1\xi)_j \;:\; \varprojlim_i D_i \to D_j \quad \text{for } j : \mathbb{J}_0.$$

Crucially for proving the η rule in Section 4.2, observe that $\varprojlim_i D_i$ is a set, since all the D_i are sets, Set is impredicative, and $\varprojlim_i D_i$ is therefore a sum of a family of propositions over a set.

Now recall that, given an endofunctor $F : \mathcal{S}et \to \mathcal{S}et$, the category F-**Alg** of F-*algebras* has as objects pairs $(X : \mathrm{Set}, \alpha : FX \to X)$, and as morphisms from (X, α) to (Y, β) the functions $f : X \to Y$ satisfying $f \circ \alpha = \beta \circ Ff$. Thus, type-theoretically, we have the precategory:

$$(F\text{-}\mathbf{Alg})_0 \;:\equiv\; \sum_{X:\mathrm{Set}} FX \to X \,,$$

$$\hom\big((X,\alpha),\,(Y,\beta)\big) \;:\equiv\; \sum_{f:X\to Y} f \circ \alpha = \beta \circ Ff \,.$$

The forgetful functor $U : F\text{-}\mathbf{Alg} \to \mathcal{S}et$ is just the first projection, that is:

$$U(X, \alpha) :\equiv X \quad \text{for} \quad (X, \alpha) : (F\text{-}\mathbf{Alg})_0 \,,$$
$$U(f, p) :\equiv f \quad \text{for} \quad (f, p) : \hom\big((X, \alpha), (Y, \beta)\big).$$

It is well-known, and easy to prove, that the precategory $F\text{-}\mathbf{Alg}$ inherits arbitrary limits from $\mathcal{S}et$, and these can be computed pointwise. Thus $F\text{-}\mathbf{Alg}$ has an initial object, which is just the limit of the identity functor. Since limits in $F\text{-}\mathbf{Alg}$ are computed pointwise, they are preserved by U, which means that the initial algebra $(I, i : FI \to I)$ has as its underlying set I the limit of the functor $U : F\text{-}\mathbf{Alg} \to \mathcal{S}et$, which using (4.1) we can write explicitly as

$$I :\equiv \sum_{\phi : U^*} \mathsf{Lim}(\phi) \tag{4.2}$$

where $\quad U^* :\equiv \prod_{A : F\text{-}\mathbf{Alg}} UA$,

and $\quad \mathsf{Lim}(\phi) :\equiv \prod_{(A, B : F\text{-}\mathbf{Alg})} \prod_{(f, p) : \hom(A, B)} f(\phi_A) = \phi_B \,.$

Summarizing the foregoing discussion, we have the following.

Theorem 4.1. *For any functor $F : \mathcal{S}et \to \mathcal{S}et$, the category $F\text{-}\mathbf{Alg}$ of F-algebras has an initial object*

$$i : FI \to I,$$

where:

(i) *the set I is given by the type* (4.2) *and is the limit of the forgetful functor $U : F\text{-}\mathbf{Alg} \to \mathcal{S}et$, and*
(ii) *the map $i : FI \to I$ is given by*

$$i(x) :\equiv (\lambda A. (\mathsf{pr}_2 A)(F(\pi_A)x), q(x)) \,,$$

where $q : \mathsf{Lim}(\lambda A. (\mathsf{pr}_2 A)(F(\pi_A))$ is constructed from functoriality of F and naturality of the limit cone. $\qquad \square$

We emphasize that the foregoing theorem is not merely semantically true in a certain model, but is provable in our system of type theory. In the following we use this construction of initial algebras to obtain an encoding of the type of natural numbers which refines the System F encoding.

4.2 Natural numbers

The inductive type $\mathbb{N}$ of natural numbers is generated by the constructors

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

From this specification we can derive the set-level System F style encoding

$$\mathbb{N}^* :\equiv \prod_{(X : \mathsf{Set})} (X \to X) \to X \to X \tag{4.3}$$

admitting constructors

$$0^* :\equiv \lambda X h x. x \,:\, \mathbb{N}^*$$
$$\mathsf{succ}^* :\equiv \lambda n X h x. \; h\,(n X h x) \,:\, \mathbb{N}^* \to \mathbb{N}^*$$

and a recursor

$$\mathsf{rec}_{\mathbb{N}}^* :\equiv \lambda X h x n. n X h x \,:\, \prod_{(X : \mathsf{Set})} (X \to X) \to X \to \mathbb{N}^* \to X.$$

These satisfy the β-rules

$$\mathsf{rec}_{\mathbb{N}}^* h x\, 0^* \equiv x \quad \text{and} \quad \mathsf{rec}_{\mathbb{N}}^* h x\, (\mathsf{succ}^* n) \equiv h(\mathsf{rec}_{\mathbb{N}}^* h x)$$

for $X : \mathsf{Set}$, $h : X \to X$ and $x : X$, but not the η-rule, which states that $\mathsf{rec}_{\mathbb{N}}^* h x$ is uniquely determined in the sense that

$$\big(f(0^*) = x \wedge f \circ \mathsf{succ}^* = h \circ f\big) \to \big(f = \mathsf{rec}_{\mathbb{N}}^* h x\big)$$

for all $f : \mathbb{N}^* \to X$ (and writing $\wedge$ in place of $\times$ for propositions).

On the other hand, $\mathbb{N}$ can be categorically characterized as the initial algebra of the functor functor $T : \mathcal{S}et \to \mathcal{S}et$ given by

$$T(X) :\equiv X + \mathbf{1} \,,$$
$$T(f) :\equiv \mathsf{rec}_+ (\mathsf{inl} \circ f)\, \mathsf{inr} \quad \text{for } f : X \to Y \,. \tag{4.4}$$

Here $\mathbf{1}$ is the unit type from (3.15).

Instantiating the type in Theorem 4.1 we get the type

$$\textstyle\sum_{(\phi : \prod_{(A : \sum_{(X : \mathsf{Set})} X + 1 \to X)} \mathsf{pr}_1 A)} \mathsf{Lim}(\phi)$$

for the underlying set of the initial algebra, and it turns out that the index type of the sum is equivalent to $\mathbb{N}^*$:

$$\prod_{A : \sum_{X : \mathsf{Set}} X + 1 \to X} \mathsf{pr}_1 A \simeq \prod_{X : \mathsf{Set}} (X + \mathbf{1} \to X) \to X$$
$$\simeq \prod_{X : \mathsf{Set}} (X \to X) \times X \to X$$
$$\simeq \mathbb{N}^*$$

We compose the predicate Lim with this equivalence to get a description of the initial algebra directly as a subtype Lim' of the System F encoding:

$$\mathbb{N} :\equiv \textstyle\sum_{(\nu : \mathbb{N}^*)} \mathsf{Lim}'(\nu) \quad \text{with}$$

$$\mathsf{Lim}'(\nu) :\equiv \prod_{\substack{X : \mathsf{Set} \\ Y : \mathsf{Set}}} \prod_{\substack{x : X \\ y : Y}} \prod_{\substack{h : X \to X \\ k : Y \to Y \\ f : X \to Y}} \begin{array}{l} (f(x) = y \wedge f \circ h = k \circ f) \\ \to f(\nu_X h x) = \nu_Y k y \end{array} \tag{4.5}$$

Observe that the triples (X, h, x) and (Y, k, y) in the definition of $\mathsf{Lim}'(\nu)$ can be coerced into T-algebras $(X, \mathsf{rec}_+ h\,(\lambda z. x))$ and $(Y, \mathsf{rec}_+ k\,(\lambda z. y))$. Leaving this coercion implicit (as we shall do from now on), $\mathsf{Lim}'(\nu)$ can be read as saying that we have

$$f(\nu_X h x) = \nu_Y k y$$

for every T-algebra morphism

$$f : (X, h, x) \to (Y, k, y).$$

It is easy to see that the predicate Lim' satisfies

$$\mathsf{Lim}'(0^*) \quad \text{and} \quad \textstyle\prod_{(\nu : \mathbb{N}^*)} \mathsf{Lim}'(\nu) \to \mathsf{Lim}'(\mathsf{succ}^* \nu),$$

whence the System F constructors 0^* and succ^* can be restricted to operations

$$0 : \mathbb{N} \quad \text{and} \quad \mathsf{succ} : \mathbb{N} \to \mathbb{N}$$

about which it is sufficient to know that they behave like 0^* and succ^* on the first components of dependent pairs. The recursor $\mathsf{rec}_{\mathbb{N}}^*$ also restricts to $\mathbb{N}$ in a straightforward manner

$$\mathsf{rec}_{\mathbb{N}} :\equiv \lambda X h x n. \mathsf{pr}_1(n) X h x$$
$$: \textstyle\prod_{\{X : \mathsf{Set}\}} (X \to X) \to X \to \mathbb{N} \to X \,,$$

and we have the following theorem.

Theorem 4.2. *The encodings of $\mathbb{N}$, 0, and succ given above satisfy*

(i) *definitional β-rules saying that*

$$\mathsf{rec}_{\mathbb{N}} h x\, 0 \equiv x \quad \text{and} \quad \mathsf{rec}_{\mathbb{N}} h x\, (\mathsf{succ}\, n) \equiv h\,(\mathsf{rec}_{\mathbb{N}} h x)$$

for all $X : \mathsf{Set}$, $x : X$, $h : X \to X$ and $n : \mathbb{N}$, and

(ii) *a propositional η-rule which states that*

$$\left(f(0) = x \wedge f \circ \mathrm{succ} = h \circ f \right) \;\rightarrow\; f = \mathrm{rec}_{\mathbb{N}}\, h\, x$$

for all X : Set, $x : X$, $h : X \to X$ and $f : \mathbb{N} \to X$.

Proof. The first claim is straightforward.

For the second claim we first show that

$$\mathrm{rec}_{\mathbb{N}}\,\mathrm{succ}\,0 = \mathrm{id}_{\mathbb{N}}. \tag{4.6}$$

By function extensionality and Σ-induction it is enough to show that $n\,\mathbb{N}\,\mathrm{succ}\,0 \;=\; (n, p)$ for all $n \in \mathbb{N}^*$ and $p \in \mathrm{Lim}'(n)$. Since $\mathbb{N}^* \hookrightarrow \mathbb{N}$ is an embedding this reduces to $\mathrm{pr}_1(n\,\mathbb{N}\mathrm{succ}\,0) = n$, and by function extensionality again it suffices to show that

$$\mathrm{pr}_1(n_{\mathbb{N}}\,\mathrm{succ}\,0)_X\, f\, x = n_X\, f\, x$$

for all X : Set, $f : X \to X$ and $x : X$. Since $\mathrm{rec}_{\mathbb{N}}\, f\, x : \mathbb{N} \to X$ is a morphism of T-algebras $(N, \mathrm{succ}, 0) \to (X, f, x)$, p gives an equality between the right hand side and $\mathrm{rec}_{\mathbb{N}}\, f\, x(n_{\mathbb{N}}\,\mathrm{succ}\,0)$ (see remark after (4.5)), which in turn is definitionally equal to the left hand side.

Now let X : Set, $x : X$, $h : X \to X$ and $f : \mathbb{N} \to X$ such that $f(0) = x$ and $f \circ \mathrm{succ} = h \circ f$. Given $n : \mathbb{N}$ we argue

$$
\begin{aligned}
f\, n &= f(\mathrm{rec}_{\mathbb{N}}\,\mathrm{succ}\,0\,n) &&\text{(by (4.6))}\\
&\equiv f(\mathrm{pr}_1(n)_{\mathbb{N}}\mathrm{succ}\,0) &&\\
&= \mathrm{pr}_1(n)_X\, h\, x &&\text{(since f is a T-algebra morphism}\\
& &&\text{from $(N, \mathrm{succ}, 0)$ to (X, h, x))}\\
&\equiv \mathrm{rec}_{\mathbb{N}}\, h\, x\, n, &&
\end{aligned}
$$

which by function extensionality proves the claim. □

Apart from the distinction of definitional and propositional equality, the preceding theorem says precisely that the F-algebra $(\mathbb{N}, \mathrm{succ}, 0)$ is initial in T-**Alg**, where for any T-algebra (C, f, c) the function underlying unique mediating morphism

$$(\mathbb{N}, \mathrm{succ}, 0) \to (C, f, c)$$

is given by $\mathrm{rec}_{\mathbb{N}}\, f\, c$. We reiterate that this assertion is to be understood as a statement *in type theory* (as in [6, 7]), in particular the uniqueness of the mediating morphism is up to propositional equality.

As before, the *induction principle* for $\mathbb{N}$ follows from recursion together with the η-rule (by *ibid.*).

4.3 Consequences of the existence of initial algebras

We note that the sets in this impredicative system are necessarily quite non-classical – as has been observed in related systems by several previous authors [17, 22, 29, 32].

Corollary 4.3. *The initial algebra theorem 4.1 implies the following facts about the category $\mathcal{S}et$:*

(i) *Every endofunctor $F : \mathcal{S}et \to \mathcal{S}et$ has a (least) fixed point,*

$$F(X) \cong X.$$

(By Lambek's lemma.)

(ii) *There is no (covariant) powerset functor $P : \mathcal{S}et \to \mathcal{S}et$. (Otherwise we would have $P(X) \cong X$ for some set X.)*

(iii) *The law of excluded middle fails for sets. (Otherwise we would have $P(X) = 2^X$.)*

Other 0-types We also remark that, using Theorem 4.1, one can encode initial algebras for other *polynomial* endofunctors on $\mathcal{S}et$ in just the same way, and thus obtain all (set-level) W-types (see [6, 7]). It follows that the internal category $\mathcal{S}et$ of sets is a (complete!) *predicative topos* in roughly the sense of [41], i.e. an LCC pretopos with W-types.

5 Some 1-Types

The foregoing development of impredicative encodings of inductive *sets* is satisfactory as such, but in the full system of HoTT one also has higher n-types, and for these there are corresponding notions of inductive type, known as *higher inductive types* (HITs). Examples include some basic spaces such as the spheres S^n, homotopy colimits, n-truncations $\|X\|_n$, and many others; see [40, ch. 6].[2]

In this section, we give an example of an impredicative encoding of a basic "1-HIT", namely the 1-sphere S^1 [24]. We give fewer details of the (more intricate) proofs, but provide enough specifics to hopefully give the reader a sense of what is involved in such higher encodings.

The 1-sphere S^1 is defined as a HIT by the constructors

$$
\begin{aligned}
&\mathrm{base} : S^1\\
&\mathrm{loop} : \mathrm{base} = \mathrm{base}\,.
\end{aligned}
\tag{5.1}
$$

Its dependent eliminator, given in [40, 6.4], can be derived from its recursor

$$\mathrm{rec}_{S^1} \;:\; \prod_{X:\mathrm{Type}_1}\,\prod_{x:X} (x = x) \to S^1 \to X\,, \tag{5.2}$$

together with the β-rules

$$
\begin{aligned}
\mathrm{rec}_{S^1}\,\mathrm{base} &\equiv x\,,\\
\mathrm{rec}_{S^1}(\mathrm{loop}) &= p\,,
\end{aligned}
\tag{5.3}
$$

for X : Type_1, $x : X$ and $p : x = x$, and a propositional η-rule stating the uniqueness of the recursor (see [35]).

We begin with the following encoding (originally proposed by Mike Shulman [34]), which is suggested by the previous 'System F style' ones.

$$S^1_F \;=\; \prod_{X:\mathcal{U}}\,\prod_{x:X} (x = x) \to X\,. \tag{5.4}$$

This has the same problem as the System F encoding of $\mathbb{N}$ (4.3), however: no uniqueness for the recursor, and so no induction principle. We will remedy this in the same way as before, now restricting the $\prod_X$ to 1-types, and then adding naturality and a higher "coherence condition", reflecting the fact that S^1 is a 1-type rather than a set.

First, to see where (5.4) came from, the *universal property of the circle* [40, Lemma 6.2.9] is given by the equivalence,

$$(S^1 \to X) \;\simeq\; \sum_{x:X} x = x\,. \tag{5.5}$$

That is to say, maps $S^1 \to X$ correspond to "loops" $x = x$ in X, with various basepoints $x : X$.

---

[2]Strictly speaking, some Set-level inductive types, such as quotients, are also HITs, in that they involve primitive identity paths. Even the propositional truncation is a HIT in this sense.

Thus by the same reasoning as in the Basic Lemma, if we had S^1, there would be an embedding,

$$S^1 \hookrightarrow \prod_{X:\mathsf{Type}_1} (S^1 \to X) \to X$$

$$\simeq \prod_{X:\mathsf{Type}_1} \left(\sum_{x:X} x = x \right) \to X \qquad \text{(by (5.5))}$$

$$\simeq \prod_{X:\mathsf{Type}_1} \prod_{x:X} (x = x) \to X \,.$$

This gives us (5.4) as a starting point, but we again need to refine the encoding to a suitable subtype.

Let us write

$$\Omega(X) \;:\equiv\; \sum_{x:X} x = x$$

for the (unbased) "loopspace functor", so that (5.4) is essentially the type of "families of maps $\Omega X \to X$". Let $\alpha : \prod_{X:\mathsf{Type}_1} \Omega X \to X$ be such a family, and consider the following naturality square, corresponding to (3.8)

$$\begin{array}{ccc} \Omega X & \xrightarrow{\;\Omega f\;} & \Omega Y \\ {\scriptstyle \alpha_X}\downarrow & & \downarrow{\scriptstyle \alpha_Y} \\ X & \xrightarrow[\;f\;]{} & Y, \end{array} \qquad (5.6)$$

where Ωf is defined in the usual way as the action of f on identity paths in X [40, 2.2]. Since we now have a 1-type Y in the target, the naturality condition

$$f \circ \alpha_X \;=\; \alpha_Y \circ \Omega f \qquad (5.7)$$

is not a proposition, but a set. We will add a "coherence condition" in order to cut it down further. Indeed, consider a naturality term

$$\vartheta : \prod_{X,Y:\mathsf{Type}_1} \prod_{f:X \to Y} f \circ \alpha_X \;=\; \alpha_Y \circ \Omega f \,.$$

Then for any $f : X \to Y$ and $g : Y \to Z$, we have identifications

$$\vartheta_f : f \circ \alpha_X \;=\; \alpha_Y \circ \Omega f \,,$$

$$\vartheta_g : g \circ \alpha_Y \;=\; \alpha_Z \circ \Omega g \,,$$

depicted

$$\begin{array}{ccccc} \Omega X & \xrightarrow{\;\Omega f\;} & \Omega Y & \xrightarrow{\;\Omega g\;} & \Omega Z \\ {\scriptstyle \alpha_X}\downarrow & {\scriptstyle \vartheta_f} & \downarrow{\scriptstyle \alpha_Y} & {\scriptstyle \vartheta_g} & \downarrow{\scriptstyle \alpha_Z} \\ X & \xrightarrow[\;f\;]{} & Y & \xrightarrow[\;g\;]{} & Z, \end{array}$$
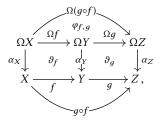
as well as one

$$\vartheta_{(g \circ f)} : (g \circ f) \circ \alpha_X \;=\; \alpha_Z \circ \Omega(g \circ f) \,.$$

Informally, the coherence condition we seek is then

$$\text{``}\ \vartheta_{(g \circ f)} = \vartheta_g \cdot \vartheta_f \ \text{''}$$

but of course, this does not type-check. Instead, we have the following "pasting scheme"

$$\begin{array}{ccccc} & & \overset{\Omega(g \circ f)}{\frown} & & \\ & & {\scriptstyle \varphi_{f,g}} & & \\ \Omega X & \xrightarrow{\;\Omega f\;} & \Omega Y & \xrightarrow{\;\Omega g\;} & \Omega Z \\ {\scriptstyle \alpha_X}\downarrow & {\scriptstyle \vartheta_f} & \downarrow{\scriptstyle \alpha_Y} & {\scriptstyle \vartheta_g} & \downarrow{\scriptstyle \alpha_Z} \\ X & \xrightarrow[\;f\;]{} & Y & \xrightarrow[\;g\;]{} & Z, \\ & & \underset{g \circ f}{\smile} & & \end{array}$$

where $\varphi_{f,g} : \Omega g \circ \Omega f \;=\; \Omega(g \circ f)$ is the provable composition law of the (pseudo-) functor Ω. This gives rise to the well-typed *coherence condition*,

$$\vartheta_{(g \circ f)} = (\alpha_Z * \varphi_{f,g}) \cdot (\vartheta_g * \Omega f) \cdot (g * \vartheta_f) \,, \qquad (5.8)$$

where $q \cdot p$ is composition of the paths p, q, and $f * p$ (resp. $p * f$) is the "whiskering" of a map f and a path p (see [40, 2.1]).

Note that (5.8) is indeed a proposition, because it is an identity between identities $(g \circ f) \circ \alpha_X \;=\; \alpha_Z \circ \Omega(g \circ f)$ in the 1-type $\Omega X \to Z$.

There is also a *unit coherence condition*, which has the simple form

$$\vartheta_{(1_X)} = \mathsf{refl}_{\alpha_X}$$

in the 1-type $\Omega X \to X$ (taking into account the fact that "Ω preserves refl").

Now for $\alpha : \prod_{X:\mathsf{Type}_1} \Omega X \to X$, let us write

$$\mathsf{Nat}(\alpha) \;:\equiv\; \prod_{X,Y:\mathsf{Type}_1} \prod_{f:X \to Y} f \circ \alpha_X = \alpha_Y \circ \Omega f$$

for the type of "naturality structures" on the family of maps α, and for $\vartheta : \mathsf{Nat}(\alpha)$, let us write

$$\mathsf{Coh}(\vartheta) \;:\equiv\; \prod_{X,Y,Z:\mathsf{Type}_1} \prod_{f:X \to Y} \prod_{g:Y \to Z} \left(\vartheta_{(1_X)} = \mathsf{refl}_{\alpha_X} \right)$$
$$\times \left(\vartheta_{(g \circ f)} = (\alpha_Z * \varphi_{f,g}) \cdot (\vartheta_g * \Omega f) \cdot (g * \vartheta_f) \right)$$

for the type of "coherence conditions" on the natural transformation ϑ.

The sharper encoding of S^1 that we seek is then

$$S^1 \;:\equiv\; \sum_{\alpha:(S^1)^*} \sum_{\vartheta:\mathsf{Nat}(\alpha)} \mathsf{Coh}(\vartheta) \qquad (5.9)$$

where, as before,

$$(S^1)^* \;:\equiv\; \prod_{X:\mathsf{Type}_1} \prod_{x:X} (x = x) \to X$$

$$\simeq \prod_{X:\mathsf{Type}_1} \left(\sum_{x:X} x = x \right) \to X$$

$$\equiv \prod_{X:\mathsf{Type}_1} \Omega X \to X \,.$$

The type (5.9) can thus be understood as consisting of those natural transformations $\Omega X \to X$ that are coherent (also called *pseudo-natural transformations*).

The constructors base and loop can then be defined:

$$\mathsf{base}^* \;:\equiv\; \lambda X x p.\ x$$

$$: \prod_{X:\mathsf{Type}_1} \prod_{x:X} (x = x) \to X$$

$$\mathsf{base} \;:\equiv\; \left(\mathsf{base}^*, \left(\lambda X Y f.\, \mathsf{refl},\ \lambda X Y Z f g.\, (\mathsf{refl}, \mathsf{refl}) \right) \right)$$

$$: \sum_{\alpha:(S^1)^*} \sum_{\vartheta:\mathsf{Nat}(\alpha)} \mathsf{Coh}(\vartheta) \qquad (5.10)$$

$$\mathsf{loop}^* \;:\equiv\; \mathsf{funext}(\lambda X\, x\, p.\ p)$$

$$: \mathsf{base}^* = \mathsf{base}^*$$

$$\mathsf{loop} \;:\equiv\; \left(\mathsf{loop}^*, (a, (b, c)) \right)$$

$$: \mathsf{base} = \mathsf{base} \,.$$

where the subterms a, b, c in loop are certain canonical higher coherences, such as $\mathrm{inv} : \vartheta \cdot \vartheta^{-1} = \mathrm{refl}$, the details of which we omit. Finally, the recursor rec_{S^1} is given by:

$$\mathrm{rec}_{S^1} :\equiv \lambda C z p \sigma. (\mathrm{pr}_1 \mathrm{pr}_1 \sigma)_C z p$$
$$: \prod_{C:\mathrm{Type}_1} \prod_{z:C} (z = z) \to S^1 \to C \qquad (5.11)$$

Theorem 5.1. *The encoding* (5.9) *of* S^1, *with the structure* base *and* loop *just given, satifies the rules for the corresponding higher inductive type, including the dependent elimination rule, with respect to (families of) 1-types.*

For the proof, as in previous cases one shows first that the encoding implies the recursion rules for S^1, including the η-rule. Then, by the results in [35, 36], it also admits dependent elimination.

Observe that our encoding can be motivated informally by a version of the Basic Lemma 3.1 for 1-types (i.e. the Yoneda lemma for bicategories) as follows, where we write I for the identity functor on the (bi)category of 1-types, and **pNat** for the groupoid of pseudo-natural transformations between pseudo-functors,

$$
\begin{aligned}
S^1 \equiv I(S^1) &\simeq \mathbf{pNat}((-)^{S^1}, I) && \text{by Yoneda} \\
&\simeq \mathbf{pNat}(\Omega, I) && \text{by (5.5)} \\
&\equiv \sum_{\alpha:(S^1)^*} \sum_{\vartheta:\mathrm{Nat}(\alpha)} \mathrm{Coh}(\vartheta).
\end{aligned}
$$

Other 1-types Similar methods can be used to give the 1-truncation $\|A\|_1$ of a general type A, in the analogous form

$$\|A\|_1 :\equiv \sum_{\alpha:A^*} \sum_{\vartheta:\mathrm{Nat}(\alpha)} \mathrm{Coh}(\vartheta) \qquad (5.12)$$

where, as in (3.16),

$$A^* :\equiv \prod_{X:\mathrm{Type}_1} (A \to X) \to X.$$

By analogy to the encoding of set quotients, we expect that one can also use the foregoing method to encode the groupoid quotient ([1]: "Rezk completion") BG of a (pre)groupoid G (cf. [35] for a type theoretic description). In principle, one should also be able to encode some "higher W-types", specified by families $x : A \vdash B(x)$ of 1-types $B(x)$, such as occur in the theory of combinatorial species [23]. But we have not investigated these possibilities.

6 Conclusion

Limitations The methods used here have certain limitations which we can now address. The first is the familiar (in type theory) issue of "large versus small elimination", i.e. elimination into types in higher universes. The encodings given here have elimination rules only with respect to types in the impredicative universe $\mathcal{U}$. That said, we made little use of the higher universes $\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \ldots$ and could simply have omitted them (with some corresponding adjustments). The resulting system would be similar to the original Calculus of Constructions [10], but with the addition of primitive identity-types. The issue of eliminating into $\mathcal{U}$ itself of course remains, as it does for the CoC.

Restricting the system to the subuniverse Set of 0-types – either by definitions as was done here, or by the addition of "extensionality" principles such as the uniqueness of identity proofs as in [2] or even the reflection rule as in [14] – seems to give a very satisfactory

type theory for sets, with quotient types, W-types, etc., within a framework with very few primitive operations.

In an intensional system admitting higher n-types, there is the interesting, and apparently new, question of whether encoded n-types can eliminate into $(n + k)$-types (note that the truncation levels are cumulative, so the converse is immediate). It seems that this is possible in some cases, but more work needs to be done to understand which ones. To give a brief indication, we have already seen the encodings $\mathbf{0} :\equiv \prod_{X:\mathcal{U}} X$ and $\mathbf{1} :\equiv \prod_{p:\mathrm{Prop}} p \to p$, but why does the first one require a product over all of $\mathcal{U}$, while Prop suffices for the second? The issue seems to be related to the preservation of certain (co)limits by the inclusions $\mathrm{Type}_n \hookrightarrow \mathrm{Type}_{n+1}$ (of course, the truncations are left adjoints to these). It seems that some encodings are "stable" in the sense that they also eliminate into higher n-types; for example, our encoding of $\mathbb{N}$ as a set should be stable, as can be seen by considering the fact that it is a W-type [7], constructed in Set, and the truncation levels are closed under W-types [12]. Similar considerations apply to other inductive 0-types encoded by the method in Section 4. Indeed, we could also encode the natural numbers $\mathbb{N}$ as a 1-type, but then show that the result is actually a 0-type and thus equivalent to the original encoding, which therefore now eliminates into all 1-types. In this way, it seems possible to establish that some n-types can indeed eliminate into $(n+k)$-types. A similar argument would seem to apply to the encoding of S^1 given in Section 5, which should therefore also be stable. An encoding that is apparently *not* stable would be a coequalizer constructed in Set; constructing it in a higher Type_n should give a homotopy colimit, which need not be a set.

The previous considerations are particularly tentative, however, because of the combinatorial difficulty of specifying the relevant coherence conditions for higher n-types. Indeed, the current methods are restricted to encodings of n-types only for very low n. More work, and perhaps some new insight, is needed to specify the coherence conditions for higher n-types, and for untruncated types, such as the 2-sphere S^2.

We also recognize that, as experience has shown, reasoning effectively about (higher) inductive types often involves large eliminations into a (univalent) universe. For conventional inductive types, the lack of such large elimination means that we can not compute initial algebras by iterating functors (since this would require elimination from $\mathbb{N}$ to $\mathcal{U}$), but fortunately we don't need this kind of iteration in the impredicative setting, since we can compute initial algebras differently, namely as in Theorem 4.1.

For higher inductive types the situation may be different, however; large elimination (plus univalence) seems to be essential in the proof that $\pi_1(S^1) = \mathbb{Z}$ (see [25]). Thus, ultimately, the utility of such encodings of HITs may be that they serve to justify adding the associated rules globally, as was done for conventional inductive types in passing from the Calculus of Constructions to the Calculus of Inductive Constructions [11, 13, 28].

Semantics A realizability model of the calculus of constructions with a proof-relevant impredicative universe is described in [38, Chapter 2]. The impredicative universe in this model is closed under small sums and identity types, and the model can be extended to include a hierarchy of predicative universes using techniques akin to those discussed in [39] (assuming sufficiently many

set-theoretic Grothendieck universes). This establishes the consistency of our system and provides a semantic framework for impredicative encodings of 0-types. Non-trivial 1-types can be modeled in a *groupoidal realizability model*, obtained by internalizing the Hofmann-Streicher groupoid model [20] in a realizability topos. A model of a type theory with an impredicative universe of types of arbitrary truncation level, based on cubical assemblies, is work in progress [3].

Related work Most of the results presented here are treated in more detail in the third-named author's M.S. thesis [37].

A formalization of the main results of this paper (including the encoding of the unit circle) has been carried out using (an impredicative branch of) the Lean proof assistant. The files are publicly available here [5].

Acknowledgments

References

[1] Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. 2015. Univalent categories and the Rezk completion. *Mathematical Structures in Computer Science* 5 (2015), 1010–1039.
[2] Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. 2007. Observational Equality, Now!. In *Proceedings of the ACM Workshop Programming Languages meets Program Verification, PLPV 2007, Freiburg, Germany, October 5, 2007*, Aaron Stump and Hongwei Xi (Eds.).
[3] Steve Awodey. 2017. Impredicative Encodings in HoTT – Toward a Realizability ∞-Topos. (2017). Public lecture at the conference "Big Proof", Newton Institute, Cambridge, available at https://www.newton.ac.uk/files/seminar/20170711090010001-1009680.pdf.
[4] Steve Awodey and Andrej Bauer. 2004. Propositions As [Types]. *J. Log. and Comput.* 14, 4 (2004), 447–471.
[5] Steve Awodey and Jonas Frey. 2018. Lean formalization of 'Impredicative encodings of (higher) inductive types'. (2018). https://github.com/awodey/Impredicative
[6] Steve Awodey, Nicola Gambino, and Kristina Sojakova. 2012. Inductive types in homotopy type theory. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*. IEEE Computer Society, 95–104.
[7] Steve Awodey, Nicola Gambino, and Kristina Sojakova. 2017. Homotopy-Initial Algebras in Type Theory. *J. ACM* 63, 6 (2017), 51:1–51:45.
[8] E.S. Bainbridge, P.J. Freyd, A. Scedrov, and P.J. Scott. 1990. Functorial polymorphism. *Theoretical Computer Science* 70, 1 (1990), 35 – 64.
[9] Aurelio Carboni, Peter Freyd, and Andre Scedrov. 1988. A categorical approach to realizability and polymorphic types. *Mathematical Foundations of Programming Language Semantics* (1988), 23–42.
[10] Thierry Coquand and Gérard Huet. 1988. The calculus of constructions. *Information and Computation* 76, 2 (1988), 95 – 120.
[11] Thierry Coquand and Christine Paulin. 1990. Inductively defined types. In *COLOG-88 (Tallinn, 1988)*. Lecture Notes in Computer Science, Vol. 416. Springer, 50–66.
[12] Nils Anders Danielsson. 2012. Positive h-levels are closed under W. (2012). https://homotopytypetheory.org/2012/09/21/positive-h-levels-are-closed-under-w
[13] Peter Dybjer. 1991. Inductive sets and families in Martin-Löf's type theory and their set-theoretic semantics. In *Logical Frameworks*, Gerard Huet and Gordon Plotkin (Eds.). Cambridge University Press, 280–30.
[14] Peter Dybjer. 1997. Representing inductively defined sets by well-orderings in Martin-L of's type theory. *Theoretical Computer Science* 176 (1997), 329–335.
[15] Nicola Gambino and Martin Hyland. 2003. Wellfounded trees and dependent polynomial functors. In *International Workshop on Types for Proofs and Programs*. Springer, 210–225.
[16] Herman Geuvers. 2001. Induction Is Not Derivable in Second Order Dependent Type Theory. In *Typed Lambda Calculi and Applications*, Samson Abramsky (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 166–181.
[17] Jean-Yves Girard. 1972. *Interprétation Fonctionnelle et Elimination des Coupures de l'Arithmétique d'Ordre Supérieur*. Ph.D. Dissertation. Paris 7, France.
[18] Jean-Yves Girard, Paul Taylor, and Yves Lafont. 1989. *Proofs and Types*. Cambridge University Press.
[19] Martin Hofmann. 1995. *Extensional concepts in intensional type theory*. Ph.D. Dissertation. University of Edinburgh. College of Science and Engineering. School of Informatics.
[20] Martin Hofmann and Thomas Streicher. 1998. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*, Giovanni Sambin and Jan M. Smith (Eds.). Oxford Logic Guides, Vol. 36. Oxford University Press, New York, 83–111.
[21] William A. Howard. 1980. The formulae-as-types notion of construction. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, J. Roger Seldin, Jonathan P.; Hindley (Ed.). Academic Press, 479–490. original paper manuscript from 1969.
[22] J.M.E. Hyland. 1988. A small complete category. *Annals of Pure and Applied Logic* 40, 2 (1988), 135 – 165.
[23] André Joyal. 1981. Une théorie combinatoire des séries formelles. *Advances in Mathematics* 42, 1 (1981), 1 – 82.
[24] Daniel R. Licata and Michael Shulman. 2013. Calculating the Fundamental Group of the Circle in Homotopy Type Theory. In *LICS 2013: Proceedings of the Twenty-Eighth Annual ACM/IEEE Symposium on Logic in Computer Science.*
[25] Daniel R. Licata and Michael Shulman. 2013. Calculating the Fundamental Group of the Circle in Homotopy Type Theory. In *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '13)*. IEEE Computer Society, Washington, DC, USA, 223–232. https://doi.org/10.1109/LICS.2013.28
[26] Per Martin-Löf. 1984. *Intuitionistic type theory*. Studies in Proof Theory, Vol. 1. Bibliopolis. iv+91 pages. Notes by Giovanni Sambin of a series of lectures given in Padua, June 1980.
[27] Ieke Moerdijk and Erik Palmgren. 2002. Type theories, toposes and constructive set theory: predicative aspects of AST. *Annals of Pure and Applied Logic* 114, 1–3 (2002), 155–201.
[28] Frank Pfenning and Christine Paulin-Mohring. 1990. Inductively defined types in the calculus of constructions. In *Mathematical Foundations of Programming Semantics, 5th International Conference, Tulane University, New Orleans, Louisiana, USA, March 29 – April 1, 1989, Proceedings*, Michael G. Main, Austin Melton, Michael W. Mislove, and David A. Schmidt (Eds.). Number 442 in Lecture Notes in Computer Science. Springer, 209–228.
[29] A. M. Pitts. 1987. Polymorphism Is Set Theoretic, Constructively. In *Category Theory and Computer Science, Proc. Edinburgh 1987 (Lecture Notes in Computer Science)*, D. H. Pitt, A. Poigné, and D. E. Rydeheard (Eds.), Vol. 283. Springer-Verlag, Berlin, 12–39.
[30] John C. Reynolds. 1974. Towards a Theory of Type Structure. In *Programming Symposium, Proceedings Colloque sur la Programmation*. Springer-Verlag, 408–423.
[31] John C. Reynolds. 1983. Types, Abstraction and Parametric Polymorphism. In *IFIP Congress*. 513–523.
[32] John C. Reynolds. 1984. Polymorphism is not set-theoretic. In *Semantics of Data Types*, Gilles Kahn, David B. MacQueen, and Gordon Plotkin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 145–156.
[33] Ivar Rummelhoff. 2004. Polynat in PER models. *Theoretical Computer Science* 316, 1-3 (2004), 215–224.
[34] Mike Shulman. 2011. Higher Inductive Types via Impredicative Polymorphism. (2011). https://homotopytypetheory.org/2011/04/25/higher-inductive-types-via-impredicative-polymorphism/
[35] Kristina Sojakova. 2015. Higher Inductive Types as Homotopy-Initial Algebras. In *Symposium on Principles of Programming Languages (POPL 2015)*, S. Rajamani and D. Walker (Eds.). 31–42.
[36] Kristina Sojakova. 2016. *Higher Inductive Types as Homotopy-Initial Algebras*. Ph.D. Dissertation. Carnegie Mellon University, USA.
[37] Sam Speight. 2017. *Impredicative Encodings of Inductive Types in Homotopy Type Theory*. Master's thesis. Carnegie Mellon University, Pittsburgh, USA. https://github.com/sspeight93/Papers/blob/master/impredicative-encodings-of-inductive-types-in-homotopy-type-theory.pdf
[38] Thomas Streicher. 1991. *Semantics of type theory*. Birkhäuser Boston.
[39] Thomas Streicher et al. 2005. Universes in toposes. *From Sets and Types to Topology and Analysis, Towards Practicable Foundations for Constructive Mathematics* 48 (2005), 78–90.
[40] The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study.
[41] B. van den Berg. 2012. Predicative toposes. *ArXiv e-prints* (July 2012). arXiv:math.CT/1207.0959