

Constructing Inductive-Inductive Types in Cubical Type Theory

Jasper Hugunin

University of Washington, Seattle WA, USA

jasper@hugunin.net ✉ <https://orcid.org/0000-0002-1133-5354>

Abstract. Inductive-inductive types are a joint generalization of mutual inductive types and indexed inductive types. In extensional type theory, inductive-inductive types can be constructed from inductive types, and this construction has been conjectured to work in intensional type theory as well. In this paper, we show that the existing construction requires Uniqueness of Identity Proofs, and present a new construction (which we conjecture generalizes) of one particular inductive-inductive type in cubical type theory, which is compatible with homotopy type theory.

1 Introduction

Inductive-inductive types allow for the mutual inductive definition of a type and a family over that type. As an example, we can simultaneously define contexts and types defined in a context, with dependently typed context extension:

$$\begin{array}{ll} \text{Ctx} : \text{Type}, & \text{Ty} : \text{Ctx} \rightarrow \text{Type}, \\ \epsilon : \text{Ctx}, & \text{U} : (\Gamma : \text{Ctx}) \rightarrow \text{Ty } \Gamma, \\ \text{ext} : (\Gamma : \text{Ctx}) \rightarrow \text{Ty } \Gamma \rightarrow \text{Ctx}, & \text{El} : (\Gamma : \text{Ctx}) \rightarrow \text{Ty } (\text{ext } \Gamma (\text{U } \Gamma)). \end{array}$$

Such definitions have been used for example by Danielsson [9] and Chapman [5] to define intrinsically typed syntax of a dependent type theory, and Agda supports such definitions natively.

These types have been studied extensively in Nordvall Forsberg [15]. There, in §5.3, inductive-inductive types with simple elimination rules (defined in op. cit. §3.2.5) are constructed from indexed inductive types in extensional type theory, and in §5.4 this is conjectured to work in intensional type theory as well.

In this paper, we first show that this construction does not work in intensional type theory without assuming Uniqueness of Identity Proofs (UIP), which is incompatible with the Univalence axiom of Homotopy Type Theory [18]. We then give an alternate construction in cubical type theory [6], which is compatible with Univalence. Specifically, this paper makes the following contributions:¹

1. In §2, we show that, in intensional type theory, if the types constructed by Nordvall Forsberg satisfy the simple elimination rules, then UIP holds (formalized in both Coq and Agda).
2. In §3, we give the construction of a particular inductive-inductive type with simple elimination rules in cubical type theory (formalized in cubical Agda).

¹ The formalization can be found at <https://github.com/jashug/ConstructingII>.

1.1 Syntax and Conventions

We mostly mimic Agda syntax. The double bar symbol $=$ is used for definitions directly and by pattern matching, and for equality of terms up to conversion. We write $(a : A) \rightarrow B$ for the dependent product type, and $A \rightarrow B$ for the non-dependent version. Functions are given by pattern matching $f x = y$ or by lambda expressions $f = \lambda x.y$. Similarly $(a : A) \times B$ is the dependent pair type, and $A \times B$ the non-dependent version. Pairs are (a, b) , and projections are $p.1$ and $p.2$. The unit type is \top , with unique inhabitant \star . Identity types are $x \equiv_X y$ for the type of identifications of x with y in type X , and we write `refl` for a proof of reflexivity. We do not assume that axiom K holds for identity types. We write `Type` for a universe of types (where Agda uses `Set`). In section 3 we work in cubical type theory, which will be explained there.

1.2 Running Example of an Inductive-Inductive Definition

For the purposes of this paper, we will focus on one relatively simple inductive-inductive definition (with only 5 clauses), parametrized by a type X , which is given in Figure 1. We will use this definition to prove that Nordvall Forsberg’s construction implies UIP in §2 and as a running example to demonstrate our construction in cubical type theory in §3.

Our example starts with the simplest inductive-inductive sorts, taking $A : \text{Type}$ and $B : A \rightarrow \text{Type}$, and then populates A and B with simple constructors which suffice for our proof of UIP. We have `inj`, which is supposed to give exactly one element of each $B a$, while `ext` lets us mix B s back into the A s (mirroring the type of context extension), and η gives us something to start with: one element of A for each element of X (following the use of η in [15, Example 3.3]). The proof of UIP in §2 proceeds by considering the type $B (\text{ext } (\eta x) (\text{inj } (\eta x)))$ for some $x : X$, and noticing that, while the simple elimination rules tell us that there should only be one element of this type (given by `inj`), in Nordvall Forsberg’s construction there are actually as many as there are proofs of $x \equiv_X x$.

Our goal in this paper is to construct $(A, B, \eta, \text{ext}, \text{inj})$ of the types given in Figure 1 such that the simple elimination rules hold without using UIP. But first, we will show why Nordvall Forsberg’s approach is not sufficient.

2 Deriving UIP

Uniqueness of Identity proofs (UIP) for a type X is the principle that, for all $x : X, y : X, p : x \equiv_X y, q : x \equiv_X y$, the type $p \equiv_{x \equiv_X y} q$ is inhabited. Equivalently, for all $x : X, p : x \equiv_X x$, the type $p \equiv_{x \equiv_X x} \text{refl}$ is inhabited. It expresses that there is at most one proof of any equality. UIP is independent of standard intensional type theory [13], and is inconsistent with Homotopy Type Theory [18].

Nordvall Forsberg’s construction of inductive-inductive types is described in [15, §5.3]. In this section, we show that if the simple elimination rules hold for

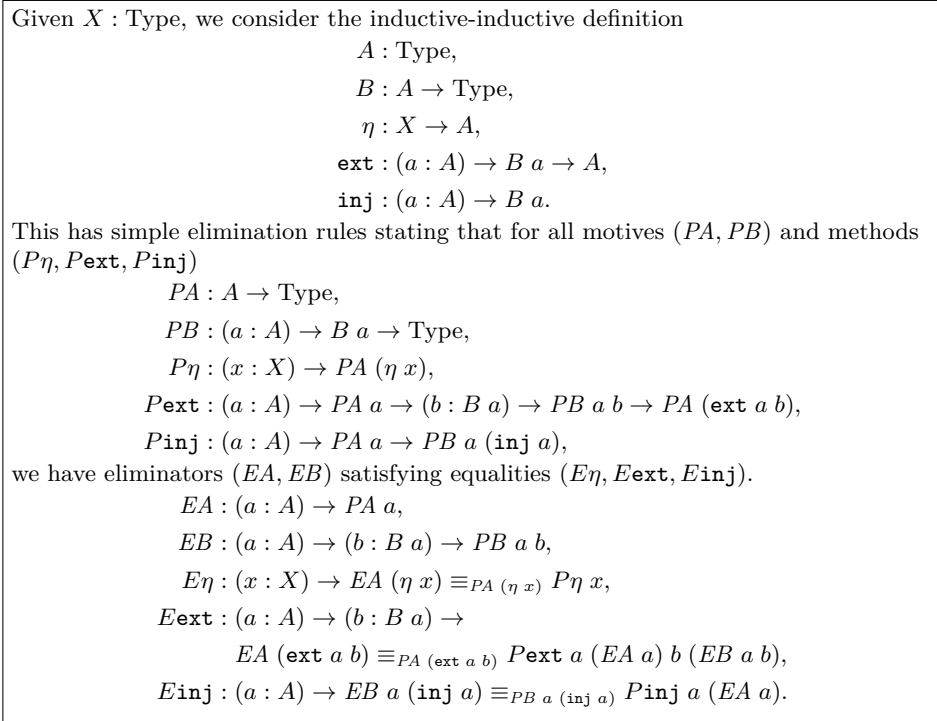


Fig. 1: Running Example

this construction of the inductive-inductive type in Figure 1, then UIP holds for the type X (Theorem 1). This argument has been formalized in both Coq version 8.8.0 [8] (see `UIP_from_Forsberg_II.v`) and Agda using the `--without-K` flag (see `UIP_from_Forsberg_II.agda`)

To recap, Nordvall Forsberg [15, §5.3] constructs an inductive-inductive type by first defining an approximation (the *pre-syntax*) which drops the A index from B leaving a mutual inductive definition. Concretely, we have A_{pre} and B_{pre} defined as in Figure 2. Then a mutual indexed inductive definition is used to define the index relationship between A_{pre} and B_{pre} ; these are the goodness predicates A_{good} and B_{good} . Finally, the inductive object $(A, B, \eta, \mathbf{ext}, \mathbf{inj})$ is defined by pairing the pre-syntax with goodness proofs (see Figure 3).

In extensional type theory, Nordvall Forsberg proved that $A_{\text{good}} a$ is a mere proposition (all inhabitants are equal) [15, Lemma 5.37(ii)]. In intensional type theory as well, if function extensionality and UIP hold then A_{good} is a mere proposition. This uniqueness of goodness proofs justifies having the definition of B ignore the goodness proof a_{good} , since a_{good} can have at most one value.

In the next two subsections, we prove that:

1. If $A_{\text{good}} a$ is a mere proposition then UIP holds for the type X (Lemma 2).
2. If the simple elimination rules from Figure 1 hold for the $(A, B, \eta, \mathbf{inj}, \mathbf{ext})$ constructed above then $A_{\text{good}} a$ is a mere proposition (Lemma 5).

Dropping the inductive index from B leaves a mutual inductive definition.

$$\begin{aligned}
&A_{\text{pre}} : \text{Type}, \\
&B_{\text{pre}} : \text{Type}, \\
&\eta_{\text{pre}} : X \rightarrow A_{\text{pre}}, \\
&\mathbf{ext}_{\text{pre}} : A_{\text{pre}} \rightarrow B_{\text{pre}} \rightarrow A_{\text{pre}}, \\
&\mathbf{inj}_{\text{pre}} : A_{\text{pre}} \rightarrow B_{\text{pre}}.
\end{aligned}$$

Fig. 2: Pre-syntax for the running example

A mutual indexed inductive definition is used to define the index relationship between A_{pre} and B_{pre} :

$$\begin{aligned}
&A_{\text{good}} : A_{\text{pre}} \rightarrow \text{Type}, \\
&B_{\text{good}} : A_{\text{pre}} \rightarrow B_{\text{pre}} \rightarrow \text{Type}, \\
&\eta_{\text{good}} : (x : X) \rightarrow A_{\text{good}} (\eta_{\text{pre}} x), \\
&\mathbf{ext}_{\text{good}} : (a_{\text{pre}} : A_{\text{pre}}) \rightarrow A_{\text{good}} a_{\text{pre}} \rightarrow \\
&\quad (b_{\text{pre}} : B_{\text{pre}}) \rightarrow B_{\text{good}} a_{\text{pre}} b_{\text{pre}} \rightarrow A_{\text{good}} (\mathbf{ext}_{\text{pre}} a_{\text{pre}} b_{\text{pre}}), \\
&\mathbf{inj}_{\text{good}} : (a_{\text{pre}} : A_{\text{pre}}) \rightarrow A_{\text{good}} a_{\text{pre}} \rightarrow B_{\text{good}} a_{\text{pre}} (\mathbf{inj}_{\text{pre}} a_{\text{pre}}).
\end{aligned}$$

The inductive-inductive object is defined as

$$\begin{aligned}
&A = (a_{\text{pre}} : A_{\text{pre}}) \times A_{\text{good}} a_{\text{pre}}, \\
&B (a_{\text{pre}}, a_{\text{good}}) = (b_{\text{pre}} : B_{\text{pre}}) \times B_{\text{good}} a_{\text{pre}} b_{\text{pre}}, \\
&\eta x = \eta_{\text{pre}} x, \eta_{\text{good}} x, \\
&\mathbf{ext} (a_{\text{pre}}, a_{\text{good}}) (b_{\text{pre}}, b_{\text{good}}) = \mathbf{ext}_{\text{pre}} a_{\text{pre}} b_{\text{pre}}, \mathbf{ext}_{\text{good}} a_{\text{pre}} a_{\text{good}} b_{\text{pre}} b_{\text{good}}, \\
&\mathbf{inj} (a_{\text{pre}}, a_{\text{good}}) = \mathbf{inj}_{\text{pre}} a_{\text{pre}}, \mathbf{inj}_{\text{good}} a_{\text{pre}} a_{\text{good}}.
\end{aligned}$$

Here, the sorts A and B are defined as pairs of the pre-syntax with a goodness proof, and operations are performed component-wise on both the pre-syntax and the goodness proof (using sort and operation in their algebraic sense).

Fig. 3: Construction given by Nordvall Forsberg

Combining these results, we conclude that Nordvall Forsberg's construction satisfies the simple elimination rules in intensional type theory only if UIP holds (Theorem 1).

2.1 Unique Goodness implies UIP

We define notation $(x == y)$ to mean the term

$$\mathbf{ext}_{\text{pre}} (\eta_{\text{pre}} x) (\mathbf{inj}_{\text{pre}} (\eta_{\text{pre}} y)) : A_{\text{pre}}.$$

We first prove that there are at least as many proofs of $A_{\text{good}} (x == y)$ as there are of $x \equiv_X y$.

Lemma 1 ($x \equiv_X y$ is a retract of A_{good}). *For all $x : X$ and $y : X$, there are functions*

$$f : x \equiv_X y \rightarrow A_{\text{good}} (x == y), \quad g : A_{\text{good}} (x == y) \rightarrow x \equiv_X y,$$

such that for all $e : x \equiv_X y$, $g (f e) \equiv e$.

Proof. To define f , we let $f \text{ refl} =$

$$\text{ext}_{\text{good}} (\eta_{\text{pre}} x) (\eta_{\text{good}} x) (\text{inj}_{\text{pre}} (\eta_{\text{pre}} x)) (\text{inj}_{\text{good}} (\eta_{\text{pre}} x) (\eta_{\text{good}} x)).$$

To define g , pattern matching on a_{good} has only one possibility: $a_{\text{good}} =$

$$\text{ext}_{\text{good}} (\eta_{\text{pre}} x) (\eta_{\text{good}} x) (\text{inj}_{\text{pre}} (\eta_{\text{pre}} x)) (\text{inj}_{\text{good}} (\eta_{\text{pre}} x) (\eta_{\text{good}} x)),$$

forcing y to be x , and in this case $x \equiv_X y$ holds by reflexivity. Then when $e = \text{refl}$, $f e$ returns a proof in the format matched by g , so $g (f \text{ refl}) \equiv \text{refl}$, and thus $g (f e) \equiv e$.

Lemma 2 (Unique goodness implies UIP). *If $A_{\text{good}} t$ is a mere proposition for all $t : A_{\text{pre}}$, then UIP holds for the type X .*

Proof. Assume goodness proofs are unique, and take $x : X$, $y : X$, with $p : x \equiv y$, $q : x \equiv y$. We want to show that $p \equiv q$. Using the f and g from Lemma 1,

$$\begin{aligned} p &\equiv g (f p) && \text{by Lemma 1} \\ &\equiv g (f q) && \text{by uniqueness in } A_{\text{good}} (x == y), f p \equiv f q \\ &\equiv q && \text{by Lemma 1.} \end{aligned}$$

2.2 Simple Elimination Rules imply Unique Goodness

Now we prove that there are at least as many proofs of $B (t_{\text{pre}}, t_{\text{good}})$ as there are of $A_{\text{good}} t_{\text{pre}}$.

Lemma 3 (A_{good} is a retract of B). *For all $t_{\text{pre}} : A_{\text{pre}}$ and $t_{\text{good}} : A_{\text{good}} t_{\text{pre}}$, there are functions*

$$f : A_{\text{good}} t_{\text{pre}} \rightarrow B (t_{\text{pre}}, t_{\text{good}}), \quad g : B (t_{\text{pre}}, t_{\text{good}}) \rightarrow A_{\text{good}} t_{\text{pre}}$$

such that for all $a_{\text{good}} : A_{\text{good}} t_{\text{pre}}$, $g (f a_{\text{good}}) \equiv a_{\text{good}}$.

Proof. We define $f a_{\text{good}} = \text{inj}_{\text{pre}} t_{\text{pre}}, \text{inj}_{\text{good}} t_{\text{pre}} a_{\text{good}}$. By induction on B_{good} , we define a function

$$g' : (a_{\text{pre}} : A_{\text{pre}}) \rightarrow (b_{\text{pre}} : B_{\text{pre}}) \rightarrow B_{\text{good}} a_{\text{pre}} b_{\text{pre}} \rightarrow A_{\text{good}} a_{\text{pre}}$$

taking

$$g' a_{\text{pre}} (\text{inj}_{\text{pre}} a_{\text{pre}}) (\text{inj}_{\text{good}} a_{\text{pre}} a_{\text{good}}) = a_{\text{good}}.$$

Then we can define $g (b_{\text{pre}}, b_{\text{good}}) = g' t_{\text{pre}} b_{\text{pre}} b_{\text{good}}$. Then $g (f a_{\text{good}}) \equiv a_{\text{good}}$ holds by reflexivity.

Lemma 4 (B a is contractible). *Assuming the simple elimination rules from Figure 1 hold for the $(A, B, \eta, \mathit{inj}, \mathit{ext})$ constructed above, for all $a : A$ and $b : B\ a$, $\mathit{inj}\ a \equiv_{B\ a} b$*

Proof. Referring to the simple elimination rules given in Figure 1, we pattern match on B by giving motives (PA, PB) and methods $(P\eta, P\mathit{ext}, P\mathit{inj})$, and then using the resulting EB .

We set $PA\ a = \top$, and take $PB\ a\ b = \mathit{inj}\ a \equiv_{B\ a} b$. Then we have $P\eta\ x = \star$, and $P\mathit{ext}\ a\ \star\ b\ H = \star$, and we take $P\mathit{inj}\ a\ \star = \mathit{refl} : \mathit{inj}\ a \equiv_{B\ a} \mathit{inj}\ a$. The conclusion follows by $EB : (a : A) \rightarrow (b : B\ a) \rightarrow \mathit{inj}\ a \equiv_{B\ a} b$.

Lemma 5 (Simple elimination rules imply unique goodness). *If the simple eliminators hold for the $(A, B, \eta, \mathit{inj}, \mathit{ext})$ constructed above, then for all $t : A_{\mathit{pre}}$, $A_{\mathit{good}}\ t$ is a mere proposition.*

Proof. Assume that the simple elimination rules hold, and take $t : A_{\mathit{pre}}$, and a_1 and a_2 in $A_{\mathit{good}}\ t$. We use the definition of f and g from Lemma 3 with $t_{\mathit{pre}} = t$ and $t_{\mathit{good}} = a_1$.

By Lemma 4, we know that

$$\mathit{inj}\ (t, a_1) \equiv_{B\ (t, a_1)} f\ a_2.$$

Applying g to both sides, and recognizing that $g\ (\mathit{inj}\ (t, a_1))$ computes to a_1 , while $g\ (f\ a_2)$ computes to a_2 we find that

$$a_1 = g\ (\mathit{inj}\ (t, a_1)) \equiv_{A_{\mathit{good}}\ t} g\ (f\ a_2) = a_2.$$

2.3 Simple Elimination Rules for Nordvall Forsberg’s Construction only if UIP

Theorem 1. *If the simple elimination rules hold for Nordvall Forsberg’s construction, then UIP holds for the type X .*

Proof. Compose the results of Lemma 2 and Lemma 5.

Therefore Nordvall Forsberg’s approach to constructing inductive-inductive types requires UIP. Since UIP is inconsistent with the Univalence axiom at the center of Homotopy Type Theory (HoTT) [18], we have an incentive to come up with a different construction which is consistent with HoTT.

3 Constructing an Inductive-Inductive Type in Cubical Type Theory

Cubical type theory [6] is a recently developed type theory which gives a constructive interpretation of the Univalence axiom of Homotopy Type Theory. It has an implementation as a mode for Agda [19], which we use to formalize the construction given in this section of the running example from Figure 1.

The most important difference between cubical type theory and standard intensional type theory as implemented by Coq or vanilla Agda is that the identity type $x \equiv_X y$ is represented (loosely speaking) by the type of functions p from an interval type \mathbb{I} with two endpoints i_0 and i_1 to X such that $p\ i_0$ reduces to x and $p\ i_1$ reduces to y . This allows, for example, a simple proof of function extensionality: if we have $A : \text{Type}$, $B : A \rightarrow \text{Type}$, f and g functions of type $(a : A) \rightarrow B\ a$, and $h : (a : A) \rightarrow f\ a \equiv g\ a$, then we have $(\lambda i.\lambda a.h\ a\ i) : f \equiv g$. Taking $\text{cong}\ f = \lambda p.\lambda i.f\ (p\ i) : x \equiv y \rightarrow f\ x \equiv f\ y$ and \circ for function composition, we also have nice properties such as $(\text{cong}\ f) \circ (\text{cong}\ g) = \text{cong}\ (f \circ g)$.

In this section, we construct the running example from Figure 1, along with the simple elimination rules, in cubical type theory. Our construction proceeds in several steps:

- In §3.1, we approximate by dropping the indices, leaving a standard mutual inductive definition called the *pre-syntax*. This is the same as the pre-syntax given in Figure 2.
- In §3.2, we define *goodness algebras*, collections of predicates over the pre-syntax which define the index relationship (analogously to A_{good} and B_{good} from §2). We also show that a goodness algebra exists, and call it \mathbb{O} .
- In §3.3, we define a predicate *nice* on goodness algebras, such that if we have a nice goodness algebra, then we can construct the simple elimination rules. Being nice is similar to having proofs of goodness be unique as in §2.
- In §3.4, we use pattern matching over the pre-syntax to define a function S from goodness algebras to goodness algebras.
- In §3.5, we define the limit of the sequence

$$\mathbb{O}, S\ \mathbb{O}, S\ (S\ \mathbb{O}), \dots, S^n\ \mathbb{O}, \dots$$

and show that it is nice. This is the only section that utilizes the differences between cubical type theory and standard intensional type theory.

Given the nice goodness algebra in §3.5 we can then construct the simple elimination rules by §3.3. This construction has been formalized in Agda² using the `--cubical` flag which implies `--without-K` (see `RunningExample.agda`).

The intuition for our construction is that the Nordvall Forsberg’s approach of pairing an approximation with goodness predicates can be repeated, and each time the approximation gets better. Using HoTT terminology, we showed in §2 that one iteration suffices only if X has homotopy level 0 (is a homotopy set, satisfies UIP). In general, $n + 1$ iterations are sufficient if only if X has homotopy level n . The successor goodness algebra defined in §3.4 is a slightly simplified version of Nordvall Forsberg’s construction, and taking the limit (in §3.5) gives a construction which works for arbitrary homotopy levels.

3.1 Pre-syntax

The pre-syntax is the same as that used in §2, defined as a mutually inductive type in Figure 2. The constructors of the pre-syntax have the same types as the

² Agda version 2.6.0 commit `bd338484d`

constructors of the full inductive-inductive definition (given in Figure 1), except we replace $B a$ with B_{pre} everywhere, ignoring the dependence of B on A .

Consider this as the closest approximation of the target inductive-inductive type by a standard inductive type; the dependence of B on A is the only new element that inductive-inductive definitions add. Of course, this is only an approximation. We can form elements of the pre-syntax, such as

$$\mathbf{ext}_{\text{pre}} (\eta_{\text{pre}} x) (\mathbf{inj}_{\text{pre}} (\eta_{\text{pre}} y))$$

for $x \neq y$ that should be excluded from the inductive-inductive formulation, since $\mathbf{inj} (\eta y) : B (\eta y)$ while $\mathbf{ext} (\eta x) : B (\eta x) \rightarrow A$.

We will use definitions by induction and by pattern-matching on the pre-syntax in sections §3.3 and §3.4 respectively.

3.2 Goodness Algebras

As we saw in §3.1, the pre-syntax is too lenient, and contains terms we want to exclude from the inductive-inductive object. In this section, we define a notion of sub-algebra of the pre-syntax, which we will call a *goodness algebra*, and explain how to combine a goodness algebra with the pre-syntax to get an inductive-inductive object $(A, B, \eta, \mathbf{ext}, \mathbf{inj})$. We also define a goodness algebra \mathbb{O} .

In Figure 4, for each clause of the inductive-inductive specification, we define 3 things:

1. For each sort X a type $\text{Ix } X$ giving the data X depends on, and for each operation F constructing an element of sort X , a family $\text{Arg } F : Y \rightarrow \text{Ix } X \rightarrow \text{Type}$ where Y collects the arguments of the operation in the pre-syntax, where $\text{Arg } F y \phi$ gives the data needed to justify that pre-syntax constructed by F_{pre} from y has index ϕ . In later sections we will also write $\text{Ix } X \delta^G$ and $\text{Arg } F \delta^G$ to specify which goodness algebra we are working in.
2. The type of the corresponding component in the goodness algebra. For sorts, this is a predicate relating Ix and the pre-syntax, while for the operations, this is a function witnessing that each element of Arg gives a goodness proof relating the index ϕ to the pre-syntax.
3. A way to combine the goodness algebra with the pre-syntax to form an inductive-inductive object. For sorts, we pair the pre-syntax with a goodness proof, while for operations we apply the operation given by the goodness algebra, mimicking the construction in Figure 3.

Comparing this definition to the construction in §2, the mutual inductive definition of A_{good} and B_{good} (in Figure 3) has types equivalent to the result of dropping the dependence of $\delta^G.B$ on $\delta^G.A$ (defined in Figure 4), going from

$$\delta^G.B : (a : A_{\text{pre}}) \times \delta^G.A \star a \rightarrow B_{\text{pre}} \rightarrow \text{Type} \text{ to } B_{\text{good}} : A_{\text{pre}} \rightarrow B_{\text{pre}} \rightarrow \text{Type}.$$

The other difference is that we replace the inductive index (call it s) in the conclusion by a fresh variable ϕ , with the condition $s = \phi$ included in Arg .

For our running example from Figure 1, a goodness algebra is the type of tuples of

$$\delta^G = (\delta^G.A, \delta^G.B, \delta^G.\eta, \delta^G.\mathbf{ext}, \delta^G.\mathbf{inj})$$

with the types defined below. Simultaneously, we define how to combine a goodness algebra δ^G with the pre-syntax to construct an inductive-inductive object $(A, B, \eta, \mathbf{ext}, \mathbf{inj})$.

$$\begin{aligned} \text{Ix } A &= \top, \\ \delta^G.A &: \text{Ix } A \rightarrow A_{\text{pre}} \rightarrow \text{Type}, \\ A &= (a : A_{\text{pre}}) \times \delta^G.A \star a \\ \\ \text{Ix } B &= A, \\ \delta^G.B &: \text{Ix } B \rightarrow B_{\text{pre}} \rightarrow \text{Type}, \\ B \phi &= (b : B_{\text{pre}}) \times \delta^G.B \phi b, \\ \\ \text{Arg } \eta \ x \ \phi &= \top \times \star \equiv_{\text{Ix } A} \phi, \\ \delta^G.\eta &: (x : X) \rightarrow (\phi : \text{Ix } A) \rightarrow \text{Arg } \eta \ x \ \phi \rightarrow \delta^G.A \phi (\eta_{\text{pre}} x), \\ \eta \ x &= \eta_{\text{pre}} x, \ \delta^G.\eta \ x \ \star (\star, \mathbf{refl}), \\ \\ \text{Arg } \mathbf{ext} \ (a, b) \ \phi &= ((a^G : \delta^G.A \star a) \times \delta^G.B (a, a^G) b) \times \star \equiv_{\text{Ix } A} \phi, \\ \delta^G.\mathbf{ext} &: (p : A_{\text{pre}} \times B_{\text{pre}}) \rightarrow (\phi : \text{Ix } A) \rightarrow \\ &\quad \text{Arg } \mathbf{ext} \ p \ \phi \rightarrow \delta^G.A \phi (\mathbf{ext}_{\text{pre}} p), \\ \mathbf{ext} \ ((a_{\text{pre}}, a_{\text{good}}), (b_{\text{pre}}, b_{\text{good}})) &= \\ &\quad \mathbf{ext}_{\text{pre}} \ a_{\text{pre}} \ b_{\text{pre}}, \ \delta^G.\mathbf{ext} \ (a_{\text{pre}}, b_{\text{pre}}) \ \star ((a_{\text{good}}, b_{\text{good}}), \mathbf{refl}), \\ \\ \text{Arg } \mathbf{inj} \ a \ \phi &= (a^G : \delta^G.A \star a) \times (a, a^G) \equiv_{\text{Ix } B} \phi, \\ \delta^G.\mathbf{inj} &: (a : A_{\text{pre}}) \rightarrow (\phi : \text{Ix } B) \rightarrow \text{Arg } \mathbf{inj} \ a \ \phi \rightarrow \delta^G.B \phi (\mathbf{inj}_{\text{pre}} a), \\ \mathbf{inj} \ (a_{\text{pre}}, a_{\text{good}}) &= \mathbf{inj}_{\text{pre}} \ a_{\text{pre}}, \ \delta^G.\mathbf{inj} \ a_{\text{pre}} \ (a_{\text{pre}}, a_{\text{good}}) \ (a_{\text{good}}, \mathbf{refl}). \end{aligned}$$

We also define the goodness algebra \mathbb{O} by

$$\begin{aligned} \mathbb{O}.A \ \phi \ a &= \top, & \mathbb{O}.B \ \phi \ b &= \top, \\ \mathbb{O}.\eta \ x \ \phi \ t &= \star, & \mathbb{O}.\mathbf{ext} \ (a, b) \ \phi \ t &= \star, & \mathbb{O}.\mathbf{inj} \ a \ \phi \ t &= \star. \end{aligned}$$

Fig. 4: Goodness algebras

3.3 Niceness

In this section, we identify a property *niceness* that is sufficient for a goodness algebra to produce an inductive-inductive object $(A, B, \eta, \mathbf{ext}, \mathbf{inj})$ which satisfies the simple elimination rules, as given in Figure 1.

To define niceness, we use the concept of equivalence, as defined in Univalent Foundations Program [18] (§4.4 Contractible fibers). Given a function

$f : A \rightarrow B$, we write $\text{isEquiv } f$ (leaving A and B implicit) to denote that f is an equivalence between A and B . We will also write $A \simeq B$ for the type of pairs of a function f with a proof that f is an equivalence.

We will say that a goodness algebra is *nice* if we have equivalence proofs $(\delta^N.\eta, \delta^N.\text{ext}, \delta^N.\text{inj})$, with types

$$\begin{aligned} \delta^N.\eta \ x \ \phi &: \text{isEquiv } (\delta^G.\eta \ x \ \phi), \\ \delta^N.\text{ext} \ (a, b) \ \phi &: \text{isEquiv } (\delta^G.\text{ext} \ (a, b) \ \phi), \\ \delta^N.\text{inj} \ a \ \phi &: \text{isEquiv } (\delta^G.\text{inj} \ a \ \phi). \end{aligned}$$

Equivalences between types are very close to equalities between types (the Univalence axiom makes this precise). If we have a *nice* goodness algebra, the combined data looks similar to a recursive definition:

$$\begin{aligned} \delta^G.A &: \top \rightarrow A_{\text{pre}} \rightarrow \text{Type}, \\ \delta^G.B &: ((a : A_{\text{pre}}) \times \delta^G.A \star a) \rightarrow B_{\text{pre}} \rightarrow \text{Type}, \\ \delta^G.A \ \phi \ (\eta_{\text{pre}} \ x) &\simeq \text{Arg } \eta \ x \ \phi, \\ \delta^G.A \ \phi \ (\text{ext}_{\text{pre}} \ a \ b) &\simeq \text{Arg } \text{ext} \ (a, b) \ \phi, \\ \delta^G.B \ \phi \ (\text{inj}_{\text{pre}} \ a) &\simeq \text{Arg } \text{inj} \ a \ \phi. \end{aligned}$$

However, the dependence of $\delta^G.B$ on $\delta^G.A$ makes this what Nordvall Forsberg calls a “recursive-recursive” definition, and so we cannot use the standard eliminator of the pre-syntax. In §3.5, we will expend much effort to construct a solution to this system. Once we have done so, the inductive-inductive object produced by the goodness algebra will satisfy the simple elimination rules, as we show in the following lemma.

Lemma 6 (Nice goodness algebras give simple elimination rules). *Given a goodness algebra δ^G with proof of niceness δ^N , the inductive-inductive object $(A, B, \eta, \text{ext}, \text{inj})$ produced from δ^G as specified in §3.2 satisfies the simple induction rules given in Figure 1.*

Proof. The proof is formalized in `RunningExample.agda`. The main idea of the proof is to induct on the pre-syntax, and exploit the equivalences provided by niceness δ^N . In the `inj` case for example, we have a proof of $\delta^G.B \ \phi \ (\text{inj}_{\text{pre}} \ a)$. But without loss of generality, we can replace that goodness proof with $\delta^G.\text{inj}$ applied to an element of $\text{Arg } \text{inj} \ a \ \phi$, which contains both a proof $a_{\text{good}} : \delta^G.A \star a$ and a proof that $(a, a_{\text{good}}) \equiv \phi$. Using J to eliminate that equality leaves a goal to which the provided simple induction step for `inj` applies. This proof does not use cubical type theory in any essential way.

3.4 Successor Goodness Algebra

We are trying to create a nice goodness algebra by taking the limit of successive approximations, so we need a step function, which we will call S , that takes a

goodness algebra δ^G and returns a new goodness algebra $S \delta^G$, which is closer in some sense to being nice. We do so by pattern matching on the pre-syntax to unroll one level of the recurrence equations niceness encodes.

We define by pattern matching

$$\begin{aligned}
 (E \delta^G).A &: (a : A_{\text{pre}}) \rightarrow (\phi : \text{Ix } A \delta^G) \rightarrow (Y : \text{Type}) \times (Y \rightarrow \delta^G.A \phi a), \\
 (E \delta^G).B &: (b : B_{\text{pre}}) \rightarrow (\phi : \text{Ix } B \delta^G) \rightarrow (Y : \text{Type}) \times (Y \rightarrow \delta^G.B \phi b), \\
 (E \delta^G).A (\eta_{\text{pre}} x) &= \lambda \phi. \text{Arg } \eta \delta^G x \phi, \delta^G.\eta x \phi, \\
 (E \delta^G).A (\text{ext}_{\text{pre}} a b) &= \lambda \phi. \text{Arg ext } \delta^G (a, b) \phi, \delta^G.\text{ext} (a, b) \phi, \\
 (E \delta^G).B (\text{inj}_{\text{pre}} a) &= \lambda \phi. \text{Arg inj } \delta^G a \phi, \delta^G.\text{inj } a \phi,
 \end{aligned}$$

which gives a new property Y which maps back to $\delta^G.B \phi b$ for each b and ϕ , and similarly for A .

Then, in Figure 5, we define the new goodness algebra $(S \delta^G)$ along with projection functions $(\delta^\pi \delta^G)$ which take Ix and Arg from $(S \delta^G)$ to δ^G .

We define the successor algebra $(S \delta^G)$ along with projection functions $(\delta^\pi \delta^G)$ by:

$$\begin{aligned}
 (\delta^\pi \delta^G).A &: \text{Ix } A (S \delta^G) \rightarrow \text{Ix } A \delta^G, \\
 (\delta^\pi \delta^G).A &= \lambda \star. \star, \\
 (S \delta^G).A \phi a &= (E \delta^G).A a ((\delta^\pi \delta^G).A \phi) .1, \\
 (\delta^\pi \delta^G).B &: \text{Ix } B (S \delta^G) \rightarrow \text{Ix } B \delta^G, \\
 (\delta^\pi \delta^G).B &= \lambda (a_{\text{pre}}, a_{\text{good}}). (a_{\text{pre}}, (E \delta^G).A a_{\text{pre}} \star .2 a_{\text{good}}), \\
 (S \delta^G).B \phi b &= (E \delta^G).B b ((\delta^\pi \delta^G).B \phi) .1, \\
 (\delta^\pi \delta^G).\eta x \phi &: \text{Arg } \eta (S \delta^G) x \phi \rightarrow \text{Arg } \eta \delta^G x ((\delta^\pi \delta^G).A \phi), \\
 (\delta^\pi \delta^G).\eta x \phi &= \lambda (\star, p). (\star, \text{cong } ((\delta^\pi \delta^G).A) p), \\
 (S \delta^G).\eta x \phi &= (\delta^\pi \delta^G).\eta x \phi, \\
 (\delta^\pi \delta^G).\text{ext} (a, b) \phi &: \text{Arg ext } (S \delta^G) (a, b) \phi \rightarrow \text{Arg ext } \delta^G (a, b) ((\delta^\pi \delta^G).A \phi), \\
 (\delta^\pi \delta^G).\text{ext} (a, b) \phi &= \lambda ((a_{\text{good}}, b_{\text{good}}), p). \text{let } a^G := (E \delta^G).A a \star .2 a_{\text{good}} \text{ in} \\
 &\quad ((a^G, (E \delta^G).B b (a, a^G) .2 b_{\text{good}}), \text{cong } ((\delta^\pi \delta^G).A) p), \\
 (S \delta^G).\text{ext} (a, b) \phi &= (\delta^\pi \delta^G).\text{ext} (a, b) \phi, \\
 (\delta^\pi \delta^G).\text{inj } a \phi &: \text{Arg inj } (S \delta^G) a \phi \rightarrow \text{Arg inj } \delta^G a ((\delta^\pi \delta^G).B \phi), \\
 (\delta^\pi \delta^G).\text{inj } a \phi &= \lambda (a_{\text{good}}, p). ((E \delta^G).A a \star .2 a_{\text{good}}, \text{cong } ((\delta^\pi \delta^G).B) p), \\
 (S \delta^G).\text{inj } a \phi &= (\delta^\pi \delta^G).\text{inj } a \phi.
 \end{aligned}$$

Fig. 5: Successor goodness algebra

The projection functions $(\delta^\pi \delta^G)$ consist of applying the map given by the second component of $(E \delta^G)$ everywhere in sight. The sorts are then defined by the first component of $(E \delta^G)$, while the operations can be defined to be the corresponding projection function itself.

Concretely, for the sort B , we define $(\delta^\pi \delta^G).B$ to map between $\text{Ix } B (S\delta^G)$ and $\text{Ix } B \delta^G$. This consists of applying the function $((E \delta^G).A a_{\text{pre}} \star .2)$ which we defined by pattern matching above to a_{good} . Then, since $(S \delta^G).B$ gets an inductive index ϕ in $(S \delta^G)$ but $((E \delta^G) b \phi .1)$ is expecting an inductive index in δ^G , we span the gap with the projection function $(\delta^\pi \delta^G).B$ just defined. The definition of A follows the same pattern, but $(\delta^\pi \delta^G).A$ is even simpler because $\text{Ix } A \delta^G = \top$ regardless of what goodness algebra we are working in.

For the operations, consider inj . Like with the sorts, we first define a projection function $(\delta^\pi \delta^G).\text{inj } a \phi$, which maps from $\text{Arg inj } (S \delta^G)$ to $\text{Arg inj } \delta^G$, and we fix up the inductive index ϕ with $(\delta^\pi \delta^G).B$. For the first component of Arg , we use the function given by the second component of $(E \delta^G).A$ to fix up a_{good} . For the second component, applying the projection $(\delta^\pi \delta^G).B$ to the equality proof works out on the left hand side because all these projection functions are doing the same thing: applying the function given by the second component of $(E \delta^G)$ everywhere. Finally, we can define $(S \delta^G).\text{inj} = (\delta^\pi \delta^G).\text{inj}$, because $(S \delta^G).\text{inj } a \phi$ is supposed to have codomain

$$(S \delta^G).B \phi (\text{inj}_{\text{pre}} a),$$

which is defined to be

$$(E \delta^G).B (\text{inj}_{\text{pre}} a) ((\delta^\pi \delta^G).B \phi) .1,$$

which reduces on $(\text{inj}_{\text{pre}} a)$ to

$$\text{Arg inj } \delta^G a ((\delta^\pi \delta^G).B \phi),$$

which is exactly the codomain of $(\delta^\pi \delta^G).\text{inj } a \phi$.

3.5 Limit of Goodness Algebras

We will now construct a nice goodness algebra by taking the limit of the sequence $S^n \mathbb{O}$ and showing that it is nice, where $S^n \mathbb{O}$ is defined by recursion on n with $S^0 \mathbb{O} = \mathbb{O}$, $S^{1+n} \mathbb{O} = S(S^n \mathbb{O})$. But first, we consider the limit of a chain of types.

Limit of Types This subsection *Limit of Types* is formalized in `Chain.agda`.

In order to take the limit of successive goodness algebras, we need to know how to work with *chains* of types. Specifically, given $(X : \mathbb{N} \rightarrow \mathbb{I} \rightarrow \text{Type})$ and $\pi : (n : \mathbb{N}) \rightarrow X (n + 1) i_0 \rightarrow X n i_1$, we consider the limit given by the type

$$\text{chain.t } X \pi = (f : (n : \mathbb{N}) \rightarrow X n i_0) \times ((n : \mathbb{N}) \rightarrow f n \equiv_{X n} \pi n (f (n + 1))).$$

If we have $x : \text{chain.t } X \pi$, then let $x.p$ denote the second projection.

This definition is designed to work well in cubical type theory, and uses the interval \mathbb{I} and native heterogeneous equality $x \equiv_X y$ where $X : \mathbb{I} \rightarrow \text{Type}$ (where we can form $p = \lambda i.w : x \equiv_X y$ when $p i_0 = x$, $p i_1 = y$, and $p i : X i$). In particular, this definition allows for dependent chains without transporting over

the base equality, which is problematic in cubical type theory because transport gets stuck on neutral types; instead given

$$\begin{aligned} A : \mathbb{N} \rightarrow \text{Type} \quad & \text{with} \quad f_A : (n : \mathbb{N}) \rightarrow A (1 + n) \rightarrow A n \quad \text{and} \\ B : (n : \mathbb{N}) \rightarrow A n \rightarrow \text{Type} \quad & \text{with} \\ f_B : (n : \mathbb{N}) \rightarrow (a : A (1 + n)) \rightarrow B (1 + n) a \rightarrow B n (f_A n a), \end{aligned}$$

we can form

$$\begin{aligned} LA &= \text{chain.t } (\lambda n. \lambda i. A n) f_A && : \text{Type}, \\ LB &= \lambda a. \text{chain.t } (\lambda n. \text{cong}(B n)(a.p n)) (\lambda n. f_B n (a.p (1 + n) i_0)) : LA \rightarrow \text{Type} \end{aligned}$$

using $\text{cong}(B n)(a.p n)$ which is particularly well behaved in cubical type theory.

This construction commutes with most type formers: dependent function types, dependent pair types, identity types, and constants. We also note a dependent version of the fact that the limit of a chain is equivalent to the limit of a shifted chain to substitute for Ahrens et al. [1, Lemma 12].

Lemma 7 (Dependent chain equivalent to shifted chain). *Given*

$$\begin{aligned} X : \mathbb{N} \rightarrow \text{Type}, \quad \pi_X : (n : \mathbb{N}) \rightarrow X (1 + n) \rightarrow X n, \\ Y_0 : (n : \mathbb{N}) \rightarrow X n \rightarrow \text{Type}, \quad Y_1 : (n : \mathbb{N}) \rightarrow X n \rightarrow \text{Type}, \\ f : (n : \mathbb{N}) \rightarrow (x : X n) \rightarrow Y_1 n x \rightarrow Y_0 n x, \\ g : (n : \mathbb{N}) \rightarrow (x : X (1 + n)) \rightarrow Y_0 (1 + n) x \rightarrow Y_1 n (\pi_X n x), \\ x : \text{chain.t } (\lambda n. \lambda i. X n) \pi_X, \end{aligned}$$

and letting the X arguments to f and g be implicit, we can define the types

$$\begin{aligned} t &= \text{chain.t } (\lambda n. \text{cong}(Y_0 n)(x.p n)) (\lambda n. \lambda y. f n (g n y)), \\ t^+ &= \text{chain.t } (\lambda n. \text{cong}(Y_1 n)(x.p n)) (\lambda n. \lambda y. g n (f (1 + n) y)). \end{aligned}$$

Applying f component-wise gives a function from t^+ to t . This function is an equivalence.

We only use Lemma 7 when $Y_1 n (\pi_X n x) = Y_0 (1 + n) x$, so we may take g to be the identity, leaving t^+ the shifted chain of t up to X arguments.

Limit of Goodness Algebras Now we use the lemmas about chains to construct a nice goodness algebra, and then conclude by constructing an inductive-inductive object $(A, B, \eta, \text{ext}, \text{inj})$ that satisfies the simple elimination rules.

Lemma 8. *A nice goodness algebra exists.*

Proof. The sorts of the limit goodness algebra are defined as a chain, and operations act pointwise on each component of the chain. To prove that the operations are equivalences, we compose a proof that Arg commutes with chains (given by combining the lemmas about chains commuting with type formers) with a proof that for each sort, the chain given by the $(E (S^n \mathbb{O}))$ is equivalent to the chain given by $(S^n \mathbb{O})$ (given by Lemma 7). Since $(E (S^n \mathbb{O}))$ is defined by pattern matching to reduce to Arg , the right and left sides of these equivalences agree, and we find that the operations are indeed nice. See the formalization for details.

Theorem 2. *There exists an inductive-inductive object $(A, B, \eta, \mathbf{ext}, \mathbf{inj})$ that satisfies the simple elimination rules as defined in Figure 1.*

Proof. A nice goodness algebra exists by Lemma 8, therefore we can construct $(A, B, \eta, \mathbf{ext}, \mathbf{inj})$ satisfying the simple elimination rules by Lemma 6.

We have therefore succeeded. In cubical type theory, the inductive-inductive definition from Figure 1 is constructible.

4 Related Work

The principle of simultaneously defining a type and a family over that type has been used many times before. Danielsson [9] used an inductive-inductive-recursive definition to define the syntax of dependent type theory, and Chapman [5] used an inductive-inductive definition for the same purpose. Conway’s surreal numbers [7] are given (up to a defined equivalence relation) by the inductive-inductive definition of number and less than, where less than is a relation indexed by two numbers [15, §7.1]. The HoTT book §11.3 gives a definition of the Cauchy reals as a higher inductive-inductive definition [18].

In his thesis and previous papers [15, 16, 17], Nordvall Forsberg studies the general theory of inductive-inductive types, axiomatizing a limited class of such definitions, and giving a set theoretic model showing that they are consistent. He also considers various extensions such as allowing a third type indexed by the first two, allowing the second type to be indexed by two elements of the first, or combining inductive-inductive definitions with inductive-recursive definitions from Dybjer and Setzer [10].

There have been several attempts to define a general class of inductive-inductive types larger than that in Nordvall Forsberg’s thesis. Kaposi and Kovács [14] gives an external syntactic description of a class which includes higher inductive-inductive types, and Altenkirch et al. [2] gives a semantic description of a class including quotient inductive-inductive types, but neither gives a type of codes that can be reasoned about internally. Working with UIP, Altenkirch et al. [4] propose a class of quotient inductive-inductive types.

Nordvall Forsberg’s thesis [15] appears to give the best previously known reduction of inductive-inductive types to regular inductive types known. As we have shown, Nordvall Forsberg’s approach can only be applied to intensional type theory if UIP holds. Furthermore, the equations for both Nordvall Forsberg’s approach and our approach only hold propositionally.

Many other structures have been reduced to plain inductive types. Our construction of inductive-inductive types can be seen as an adaptation of the technique in Ahrens et al. [1], where coinductive types are constructed from \mathbb{N} by taking a limit. Indexed inductive types (which are used in Nordvall Forsberg’s construction) are constructed from plain inductive types in Altenkirch et al. [3], with good computational properties (provided an identity type that satisfies J strictly). And small induction-recursion is reduced to plain indexed inductive types in Hancock et al. [11].

5 Conclusions and Future Work

In this paper, we have:

1. Shown that the construction of inductive-inductive types given by Nordvall Forsberg implies UIP.
2. Given an alternative construction of one particular inductive-inductive type in cubical type theory, which is compatible with Homotopy Type Theory.

We claim that the construction of our specific running example is straightforwardly generalizable to other inductive-inductive types, and have formalized the construction of a number of other examples including types with non-finitary constructors and indices to support this claim (see the GitHub repository referenced in the introduction).

Going forward, we would like to investigate

- An internal definition of inductive-inductive specifications in HoTT. Early experiments suggest that this requires surmounting difficulties related to increasingly complex coherence conditions similar to those encountered when defining semi-simplicial sets, c.f. Herbelin [12].
- Extending the proof given here to construct the general elimination rules. The general elimination rules were defined in Nordvall Forsberg [15], but that formulation they relies on either strict computation rules or extensional type theory to be well typed. Kaposi and Kovács [14] give equivalent rules which are well typed in intensional type theory.
- Identifying what needs to be added for the simple elimination rules to have the expected computational behavior. Given the similar construction method, this hopefully also allows the construction of coinductive types with nice computational behavior, c.f. Ahrens et al. [1].
- In the opposite direction from the previous point, rewriting the construction given here in Coq + Function Extensionality. While the elimination rules will have poor computational behavior, this would make using inductive-inductive types in Coq possible without requiring any change to Coq itself, while being compatible with HoTT. In particular, using cubical type theory makes the proofs in §3.5 simpler, but we speculate that axiomatic function extensionality is sufficient.

Acknowledgements I would like to thank Talia Ringer and Dan Grossman from the UW PLSE lab, for their invaluable feedback throughout the revision process. I also thank Pavel Pancheckha, John Leo, Remy Wang, and Fredrik Nordvall Forsberg for their comments.

Some of this work was completed while studying at Tokyo Institute of Technology under Professor Ryo Kashima. I would like to thank Professor Kashima, as well as my fellow lab members and mentors Asami and Maniwa for making my stay both productive and enjoyable.

Bibliography

- [1] Ahrens, B., Capriotti, P., Spadotti, R.: Non-wellfounded trees in homotopy type theory. In: TLCA (2015)
- [2] Altenkirch, T., Capriotti, P., Dijkstra, G., Kraus, N., Nordvall Forsberg, F.: Quotient inductive-inductive types. In: Baier, C., Dal Lago, U. (eds.) Foundations of Software Science and Computation Structures, FoSSaCS. pp. 293–310. Springer International Publishing (2018)
- [3] Altenkirch, T., Ghani, N., Hancock, P., McBride, C., Morris, P.: Indexed containers. *Journal of Functional Programming* 25, e5 (2015)
- [4] Altenkirch, T., Kaposi, A., Kovács, A.: Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.* 3(POPL), 2:1–2:24 (Jan 2019), <http://doi.acm.org/10.1145/3290315>
- [5] Chapman, J.: Type theory should eat itself. *Electronic Notes in Theoretical Computer Science* 228, 21 – 36 (2009), <http://www.sciencedirect.com/science/article/pii/S157106610800577X>, proceedings of the International Workshop on Logical Frameworks and Metalanguages: Theory and Practice (LFMTP 2008)
- [6] Cohen, C., Coquand, T., Huber, S., Mörtberg, A.: Cubical Type Theory: a constructive interpretation of the univalence axiom. *IfCoLog Journal of Logics and their Applications* 4(10), 3127–3169 (Nov 2017), <http://www.collegepublications.co.uk/journals/ifcolog/?00019>
- [7] Conway, J.: On Numbers and Games. Ak Peters Series, Taylor & Francis (2000), <https://books.google.com/books?id=tXiVo8qA5PQC>
- [8] Coq Development Team, T.: The Coq proof assistant, version 8.8.0 (Apr 2018), <https://doi.org/10.5281/zenodo.1219885>
- [9] Danielsson, N.A.: A formalisation of a dependently typed language as an inductive-recursive family. In: Altenkirch, T., McBride, C. (eds.) *Types for Proofs and Programs*. pp. 93–109. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
- [10] Dybjer, P., Setzer, A.: A finite axiomatization of inductive-recursive definitions. In: Girard, J.Y. (ed.) *Typed Lambda Calculi and Applications*. pp. 129–146. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
- [11] Hancock, P., McBride, C., Ghani, N., Malatesta, L., Altenkirch, T.: Small induction recursion. In: *Typed Lambda Calculi and Applications, 11th International Conference, TLCA 2013*. pp. 156–172 (2013)
- [12] Herbelin, H.: A dependently-typed construction of semi-simplicial types. *Mathematical Structures in Computer Science* 25(5), 1116–1131 (2015)
- [13] Hofmann, M., Streicher, T.: The groupoid interpretation of type theory. In: *Twenty-five years of constructive type theory (Venice, 1995)*, Oxford Logic Guides, vol. 36, pp. 83–111. Oxford Univ. Press, New York (1998)
- [14] Kaposi, A., Kovács, A.: A Syntax for Higher Inductive-Inductive Types. In: Kirchner, H. (ed.) *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018)*. Leibniz Internatio-

- nal Proceedings in Informatics (LIPIcs), vol. 108, pp. 20:1–20:18. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2018), <http://drops.dagstuhl.de/opus/volltexte/2018/9190>
- [15] Nordvall Forsberg, F.: Inductive-inductive definitions. Ph.D. thesis, Swansea University (2013)
 - [16] Nordvall Forsberg, F., Setzer, A.: Inductive-inductive definitions. In: Dawar, A., Veith, H. (eds.) CSL 2010. Lecture Notes in Computer Science, vol. 6247, pp. 454–468. Springer, Heidelberg (2010)
 - [17] Nordvall Forsberg, F., Setzer, A.: A finite axiomatisation of inductive-inductive definitions. In: Berger, U., Hannes, D., Schuster, P., Seisenberger, M. (eds.) Logic, Construction, Computation, Ontos mathematical logic, vol. 3, pp. 259–287. Ontos Verlag (2012)
 - [18] Univalent Foundations Program, T.: Homotopy Type Theory: Univalent Foundations of Mathematics. <https://homotopytypetheory.org/book>, Institute for Advanced Study (2013)
 - [19] Vezzosi, A.: Adding cubes to agda (2017), <https://hott-uf.github.io/2017/abstracts/cubicalagda.pdf>