Mutual and Higher Inductive Types in Homotopy Type Theory

Paolo Capriotti

August 12, 2014

Abstract

Inductive types can be cleanly represented internally as W-types [14] [20], that is, as initial algebras of containers [1]. In this paper, we give a similar presentation that extends the notion of W-type to more general forms of induction, including mutually defined data types and higher inductive types.

1 Introduction

This paper gives a unified presentation of the syntax and semantics of higher inductive types [12] [16] in Homotopy Type Theory, combining ideas from the existing purely semantic approach [13] with the dialgebra construction for induction induction [2].

The basic idea is to regard constructors as containers (also known as familially representable functors, or polynomial functors) on some category $\mathcal{C}$, with a forgetful functor U to the category of types Type.

The corresponding category of algebras then another category over Type. This constitutes the basic step of an iterative process, which begins with Type and terminates on some category, the initial object of which is the desired inductive type.

For example, let us consider the classic inductive definition of binary trees A, with constructors:

$$l : A$$
$$n : A \to A \to A.$$

We begin with $C_0 = \mathsf{Type}$, then consider the functor

$$F_0 : \mathcal{C}_0 \to \mathsf{Type}$$
$$F_0(X) :\equiv 1,$$

expressing the parameters of the constructor l. We then take C_1 to be the category of algebras of F_0, i.e. the category of pointed types. Now we move to the constructor n. The associated functor is:

$$F_1 : \mathcal{C}_1 \to \mathsf{Type}$$
$$F_1(X, l) :\equiv X \times X,$$

correspondingly, we get a category of dialgebras C_2.

Note that since F_1 factors through Type, we could have merged these two steps into one, by taking the coproduct of F_0 and the functor factorising F_1. For non-mutual, non-higher inductive definitions, it is the case most of the times that point constructors can be merged into a single one. However, we will see how allowing constructors to depend on previous ones turns out to be a crucial feature in the general case.

We could stop here, take the initial object of C_2, and observe that its universal property is equivalent to the usual eliminator of the corresponding inductive type. However, to illustrate how higher induction fits into this scheme, we add a new path constructor:

$$q : (x, y : A) \to n(x, y) \equiv n(y, x).$$

Path constructors require more data. We still need a functor

$$F_2 : \mathcal{C}_2 \to \mathsf{Type}$$
$$F_2(X, l, n) :\equiv X \times X,$$

determining the parameters of the constructor q; furthermore, we have to specify two natural transformations:

$$l, r : F_2 \to U_2,$$

where $U_2 : \mathcal{C}_2 \to \mathsf{Type}$ is the forgetful functor. The two natural transformations l and r determine a functor

$$\mathsf{Eq}\,(l, r) : \int_{\mathcal{C}_2} F_2 \to \mathsf{Type}$$
$$\mathsf{Eq}\,(l, r)\,(X, l, n, (x, y)) :\equiv (l(x) = r(x)),$$

and the category C_3 is obtained as the category of *dependent* algebras:

$$C_3 :\equiv (X : \mathcal{C}_2) \times ((x : F_2(X)) \to \mathsf{Eq}\,(l, r)\,(X, x)).$$

As before, the initial object of C_3 is the required higher inductive type, which in this case is the type of non-planar binary trees.

The astute reader will have noticed at this point that we have so far been quite imprecise about what exactly we mean by "category". In fact, the usual

definition [15, chapter 9] does not work here, since Type itself is not a category in this sense.

The correct notion to use here is probably that of an $(\infty, 1)$-category, for which there currently exists no internalisation. We can get very close, however, by using what we might call *semi-Segal types*, i.e. semi-simplicial types such that all Segal maps are equivalences. Unfortunately, semi-simplicial types are also problematic to internalise, and although techniques to work with them do exist [17] [4], they are rather convoluted and require the development to be carried out externally.

Therefore, we will take a shortcut here, and restrict ourselves to sets. In particular, we replace Type with Set everywhere, and work with actual (1-)categories.

As a consequence, all our inductive definitions are going to be 0-truncated. That doesn't mean that higher constructors are useless, as they can still be used to construct quotiens, but it does severely limit the expressiveness of the framework. In particular, we cannot use it to construct higher types like spheres.

Nevertheless, we can still obtain new interesting constructions, which had not been previously formally developed, like the syntax of type theory with definitional equality.

2 Shapes

Before defining our syntax for constructors, we need to specify the *shape* of the induction. The shape determines which of the common induction patterns we are going to use, and provides the choice of possible constructors that can be expressed.

Examples of shapes include: *simple induction*, where a single type is defined, *indexed induction*, which defines a type family over a fixed type, *mutual induction*, where several types are defined simultaneously, or any arbitrary combination of these. In particular, *induction induction* [2] can be regarded as a form of mutual induction where some of the types being defined are also indexed over some of the others.

We do not address general *induction recursion* [6] [7], but it is worth noting that the special case of *small* induction recursion (i.e. when the codomain of the recursive definition is not larger than the type being defined) can be obtained as a special case of indexed induction [9].

Definition 2.1. The 2-category of *sorts* is the full sub-2-category of Cat generated by Set and 1.

Sorts classify the artefacts of an inductive definition. The sort Set corresponds to the types being defined, whereas 1 corresponds to indices.

Definition 2.2. A *shape* is a finite 2-diagram in the 2-category of sorts, i.e. a strict 2-functor from a finite strict 2-category to the 2-category of sorts.

A shape determines the relationship between types and indices. For example, to define an indexed type $A : \mathbb{N} \to \mathsf{Set}$ by induction, we can use the shape

$$1 \xrightarrow{\mathbb{N}} \mathsf{Set}.$$

For a mutual induction with no indices involved, we can use a discrete diagram with several copies of Set as the shape.

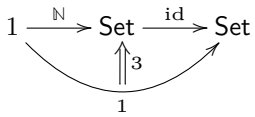
For an induction induction with types $A : \mathsf{Set}$ and $B : A \to \mathsf{Set}$, we would take

$$\mathsf{Set} \xrightarrow{\mathrm{id}} \mathsf{Set}.$$

However, our definition of shape is much more general than this. We can have arbitrarily complicated indexing structures, like:

$$A : \mathbb{N} \to \mathsf{Set}$$
$$B : A(3) \to \mathsf{Set},$$

which is represented by the shape:



Definition 2.3. Let S be a shape. The *base category* of S is the oplax limit of S.

The base category is where the construction of the inductive type begins. All the categories that we construct will be equipped with a forgetful functor to the base category.

In the case of simple induction, the base category is Set itself. In general, it is the category of types with indices arranged in exactly the configuration dictated by the shape. The result of an inductive definition is going to correspond to an object in this category.

3 Containers

In this section, we generalise the notion of container [1] [8] to functors from an arbitrary category to Set. These are sometimes called *familially representable* functors [5].

Definition 3.1. A *container* is given by

- A set A. The elements of A are called *shapes*.
- A family $B : A \to \mathsf{Set}$ with base A. For any $a : A$, the elements of $B(a)$ are called *positions*.

Definition 3.2. Let F be a container given by shapes A and positions B. The *extension* of F is the functor (also denoted F) given on objects by:

$$\begin{aligned} &F : \mathsf{Set} \to \mathsf{Set} \\ &F(X) :\equiv \sum(a : A).\ B(a) \to X \end{aligned} \tag{1}$$

Definition 3.3. Let F_1, F_2 be containers, with shapes A_1, A_2, and positions B_1, B_2. A morphism $F_1 \to F_2$ is given by

- A function $f : A_1 \to A_2$.
- A function $g : \prod(a : A_1).\ B_2(f(a)) \to B_1(a)$.

A morphism of containers $(f, g) : F_1 \to F_2$ determines a natural transformation between the two extensions as follows:

$$\begin{aligned} &\alpha : (X : \mathsf{Set}) \to F_1(X) \to F_2(X) \\ &\alpha_X \langle a, u \rangle :\equiv \langle f(a), u \circ g_a \rangle \end{aligned} \tag{2}$$

4 Constructors

A constructor is given by two pieces of data:

- the *parameters*;
- the *target*.

For example, consider the constructor q from the introduction:

$$q : (x, y : A) \to n(x, y) = n(y, x).$$

The parameters of q are given by the functor $F(X, l, n) :\equiv X \times X$. The target is given by the functor $G(X, l, n, (x, y)) :\equiv n(x, y) = n(y, x)$.

Both the parameters and target of a constructors are subject to certain restrictions.

Intuitively, the functor F defining the parameters of a constructor is required to be "strictly positive", which we will approximate by requiring that F be a container.

Functors for targets are even more restricted. Intuitively, only functors returning one of the types being defined, or some equality of those types, make sense as targets.

Parameters of a constructors may also include *indices*. For instance, when defining vectors (i.e. length-indexed lists) as an indexed inductive type $V : \mathbb{N} \to \mathsf{Set}$, one typically has a constructor of the form:

$$\mathsf{nil} : V(0)$$

and the 0 argument of V has to be provided as part of the specification of the constructor parameters.

For the rest of this section, let $S : J \to \mathsf{Cat}$ be a shape, and $\mathcal{C}_0$ its base category. Let $\mathcal{C}$ be any category equipped with a (forgetful) functor $U : \mathcal{C} \to \mathcal{C}_0$. For any $j : J$, we denote by J/j the slice 2-category over j, and by $J//j$ its sub-2-category generated by all the objects except the identity morphism on j.

Definition 4.1. Let $j : J$ be a 0-cell in the shape diagram. A *j-cone* is an oplax cone for S over J/j such that its restriction to $J//j$ is the cone induced by the forgetful functor U.

For any j, j-cones form a category, morphisms being modifications that restrict to the identity on $J//j$. The category of j-cones can be regarded as a subcategory of the category of functors $\mathcal{C} \to \mathcal{C}_0$.

The forgetful functor U trivially induces a j-cone, which we will refer to as the *trivial* j-cone, and denote U^j.

Given a j-cone F, we denote by $\bar{F}$ its component on the identity of j. Note that F is completely specified by the single functor $\bar{F}$, together with a finite number of natural transformations.

If $S_j = \mathsf{Set}$, we can form the category category $\int_{\mathcal{C}} \bar{F}$, which is itself equipped with a forgetful functor to $\mathcal{C}_0$. We can therefore talk about j-cones on $\int_{\mathcal{C}} \bar{F}$ as well. There is a canonical such j-cone, denoted T^F, with:

$$\begin{aligned} T^F_j &: \int_{\mathcal{C}} \bar{F} \to \mathcal{C}_0 \\ T^F_j(X, x) &:\equiv 1 \end{aligned}.$$

The functor T^F is characterised by the property:

$$\mathsf{Lan}(F) = T^F,$$

where the left Kan extension is taken over the forgetful functor $\int_{\mathcal{C}} \bar{F} \to \mathcal{C}$.

Lemma 4.2. *Suppose $S_j = \mathsf{Set}$, let F be a j-cone on $\mathcal{C}$, G a j-cone on $\int_{\mathcal{C}} \bar{F}$, and $l, r : T^F \to G$ j-cone morphisms. The functor*

$$\mathsf{Eq}\,(l,r) : \int_{\mathcal{C}} \bar{F} \to \mathcal{C}_0$$
$$\mathsf{Eq}\,(l,r)\,(X,x) :\equiv (l_j(X,x) = r_j(X,x))$$

is a j-cone.

We are now ready to define constructors.

Definition 4.3. For any $j : J$, a *j-parameter* is a j-cone whose j-th component is a container.

Definition 4.4. Let F be a j-parameter. A *target* for F is a functor $\int_{\mathcal{C}} \bar{F} \to \mathcal{C}_0$ defined inductively as follows:

- the trivial j-cone U^j is a target;
- for any target G, and j-cone morphisms $l, r : T^F \to G$, the j-cone $\mathsf{Eq}\,(l,r)$ is a target.

Definition 4.5. A *constructor* is given by:

- a node j, with $S_j = \mathsf{Set}$;
- a j-parameter F;
- a target for F.

We say that a constructor (j, F, G) is a *point constructor* if G is the trivial j-cone; otherwise, we say that it is a *higher constructor*.

5 Algebras

As in section 4, let $S : J \to \mathsf{Cat}$ be a shape, $\mathcal{C}_0$ its base category, and $\mathcal{C}$ any category with a forgetful functor $U : \mathcal{C} \to \mathcal{C}_0$.

Definition 5.1. Let $c = (j, F, G)$ be a constructor on $\mathcal{C}$. An *algebra* for c is given by:

- an object X of $\mathcal{C}$,
- a j-cone morphism $f : T^F \circ \widehat{X} \to G \circ \widehat{X}$,

where $\widehat{X}$ denotes the functor

$$\widehat{X} : \bar{F}(X) \to \int_{\mathcal{C}} \bar{F}$$
$$\widehat{X}(x) :\equiv (X, x),$$

and $\bar{F}(X)$ is regarded as a discrete category.

Algebras of a constructor form a category $\mathsf{Alg}_{\mathcal{C}} c$. It is easy to see that, when the target G is the trivial j-cone, algebras of c correspond exactly to the dialgebras used in [2].

More generally, the target G is allowed to depend on x. Therefore, an algebra over X is given by a dependent function:

$$(x : \bar{F}(X)) \to G(X, x),$$

compatible with the j-cone structures of F and G.

Definition 5.2. An *inductive specification* is a finite sequence of constructors $(c_0, \dots, c_n)$, where c_0 is a constructor on $\mathcal{C}_0$, and c_{i+1} is a constructor over the category of algebras of c_i. An algebra of the inductive specification is just an algebra of c_n. An *inductive type* is an initial algebra of an inductive specification.

Under no assumptions on the ambient type theory, we cannot say much about categories of algebras of inductive specifications. However, we can show that the inductive framework just introduced always produces valid inductive types, given the existence of simple W-types and quotients:

Proposition 5.3. *If every container has an initial algebra and coequalisers exist in* Set*, then every inductive specification has an initial algebra.*

In fact, the converse also holds, since both initial algebras of containers and coequalisers can be realised with simple inductive specifications. The hypotheses of proposition 5.3 are verified in a large class of models of Homotopy Type Theory, including simplicial sets, or more generally simplicial presheaves over elegant Reedy categories [13] [18].

The following result follows from proposition 5.3, and shows that common examples can indeed be encoded within the framework of inductive specifications (see section 6).

Proposition 5.4. *Let* $\mathcal{C}$ *be the category of algebras of an inductive specification, and* j *be any node with* $S_j = \mathsf{Set}$*. The trivial* j*-cone is a* j*-parameter.*

6 Examples

6.1 Non-planar trees

We will revisit the example of the introduction, and show how it can be obtained as an initial algebra of an inductive specification.

We are defining a single type, so the shape diagram S in this case is particularly simple: only one node $*$ of sort Set, with no cells. The corresponding base category $\mathcal{C}_0$ is just Set itself. Therefore, $*$-cones are just functors to Set.

The l constructor does not have any parameters. Therefore, we define:

$$F_0 : \mathcal{C}_0 \to \mathsf{Set} F_0(X) :\equiv 1.$$

The functor F_0 is clearly a container, so it defines a $*$-parameter. The corresponding target is just the identity. We then get a category of algebras, whose objects are given by:

$$\mathcal{C}_1 \equiv (X : \mathsf{Set}) \times X,$$

since an $(*, F_0, \mathrm{id})$-algebra is none other than a pointed set.

The parameter for the n constructor is given by:

$$F_1 : \mathcal{C}_1 \to \mathsf{Set} F_1(X, l) :\equiv X \times X.$$

Proposition 5.4 implies that F_1 is a container, since it can be obtained by composing the container $X \mapsto X \times X$ on Set with the trivial $*$-cone.

Again, the target is the trivial $*$-cone, and the corresponding category of algebras is given by:

$$\mathcal{C}_2 : (X : \mathsf{Set}) \times (l : X) \times (X \to X \to X).$$

We now get to the path constructor q. The parameter is given by:

$$\begin{aligned} &F_2 : \mathcal{C}_2 \to \mathsf{Set} \\ &F_2(X, l, n) :\equiv X \times X, \end{aligned}$$

which is a container by proposition 5.4 again. To define the target, we need two natural transformations:

$$l, r : T^{F_2} \to U : \int_{C_2} F_2 \to \mathsf{Set}.$$

In the introduction, we gave an equivalent presentation of l and r as transformations $F_2 \to U_2$. The definitions of l is:

$$l_{(X, l, n, (x, y))}(*) :\equiv n(x, y),$$

and similarly for r (with x and y swapped). Of course, in pratice one might prefer to express l and r as container morphisms, which can be easily done

by unfolding the representation of U_2 as a container. This has the advantage that there is no need to prove naturality, and allows the definition to be easily generalised to an untruncated setting (see section 7.3).

Now we take $(*, F_2, \mathsf{Eq}\,(l, r))$ as our constructor, and the initial object of the corresponding category of algebras is the resulting inductive type of non-planar trees.

6.2 Contexts and types

We can also obtain the main example of [2] in our system: we want to simultaneously define a type $\mathsf{Con} : \mathsf{Set}$ and a family $\mathsf{Ty} : C \to \mathsf{Set}$, equipped with the following constructors:

$$\begin{aligned}
&\mathsf{nil} : \mathsf{Con}\\
&\mathsf{cons} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}(\Gamma) \to \mathsf{Con}\\
&\mathsf{base} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}(\Gamma)\\
&\mathsf{pi} : (\Gamma : \mathsf{Con})(A : \mathsf{Ty}(\Gamma)) \to \mathsf{Ty}(\mathsf{cons}(\Gamma, A)) \to \mathsf{Ty}(\Gamma).
\end{aligned}$$

Now the shape diagram is slightly more complicated. Namely, it has two nodes a and b, with an arrow $a \to b$. Both nodes are mapped to Set, and the arrow to the identity functor. The diagram therefore looks like this:

$$\mathsf{Set} \xrightarrow{\mathrm{id}} \mathsf{Set}.$$

The base category is simply the category of *families* of sets, which we represent using indexing (see 7.1 for a discussion about indices versus functions):

$$\mathcal{C}_0 \equiv (C : \mathsf{Set}) \times (C \to \mathsf{Set}).$$

The constructors nil and cons can be defined similarly to the point constructors of the previous example, as they are given simply by a container as the a-parameter and the trivial a-cone as the target. Note that since a is the initial node in the diagram, an a-cone is just a functor to Set. Once these two constructors are defined, we obtain a category of algebras:

$$\mathcal{C}_2 :\equiv (C : \mathsf{Set}) \times (T : C \to \mathsf{Set}) \times (n : C) \times (c : (\Gamma : C) \to T(\Gamma) \to C).$$

The base container is already more interesting. Its parameter is now a b-cone, which is given by a functor:

$$\begin{aligned}
&F_2 : \mathcal{C}_2 \to \mathsf{Set}\\
&F_2(C, T, n, c) :\equiv C,
\end{aligned}$$

together with a natural transformation

$$\alpha : F_2 \to U^a$$
$$\alpha_{(C,T,n,c)}(\Gamma) :\equiv \Gamma,$$

specifying the value of the index. The target is again trivial (the trivial b-cone in this case).

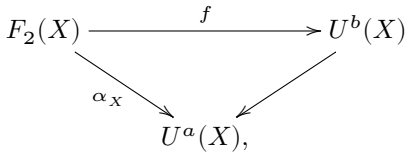
Now the category of algebras is composed of objects X of $\mathcal{C}_2$, together with b-cone morphisms:

$$T^{F_2} \circ \widehat{X} \to U^b \circ \widehat{X}.$$

Since the target is trivial, this can be simplified to a b-cone morphism:

$$F_2 \circ X \to U^b \circ X,$$

where X is regarded as a functor $1 \to \mathcal{C}_2$. By expanding the definition, this amounts to a function $f : F_2(X) \to U^b(X)$ such that the diagram:



commutes. Therefore, we get:

$$\begin{aligned}\mathcal{C}_3 &\equiv ((C,T,n,c) : \mathcal{C}_2) \times ((x : F_2(C,T,n,c)) \to T(\alpha(x)))\\ &\equiv (C : \mathsf{Set}) \times (T : C \to \mathsf{Set}) \times (n : C)\\ &\times (c : (\Gamma : C) \to T(\Gamma) \to C) \times ((\Gamma : C) \to T(\Gamma)),\end{aligned}$$

as expected.

The last constructor pi is defined similarly. Note that the parameter of pi depends on the previous constructor cons. This is where our iterative approach shows its strength: by being able to define each constructor over the algebras of the previous one, we can encode such dependencies without any difficulty.

The end result is the initial object of the category

$$\begin{aligned}\mathcal{C}_4 \equiv\ & (C : \mathsf{Set}) \times (T : C \to \mathsf{Set}) \times (n : C)\\ &\times (c : (\Gamma : C) \to T(\Gamma) \to C) \times (b : (\Gamma : C) \to T(\Gamma))\\ &\times (p : (\Gamma : C)(A : T(\Gamma)) \to T(c(\Gamma, A)) \to T(\Gamma)).\end{aligned}$$

6.3 The circle and the torus

Although our system is limited to 0-truncated types, we are still able to express the inductive specifications of higher types. Of course, the results are going to

be 0-truncated versions of the desired higher inductive types, which are not very useful, but can serve to illustrate how our definition of higher constructor would work in the untruncated setting (see section 7.3).

We begin with the circle $S^1 : \mathsf{Set}$, defined inductively by the two constructors:

$$\begin{aligned} &b : S^1 \\ &p : b = b. \end{aligned}$$

The definition of the inductive specification for S^1 proceeds similarly to the non-planar tree example (section 6.1). The resulting category of algebras is:

$$\mathcal{C}_2 \equiv (X : \mathsf{Set}) \times (b : X) \times (b = b).$$

Of course, C_2 is isomorphic to the category of pointed sets, since the third component is always trivial (X is a set!), but we will ignore this for the sake of the example.

To get to the torus, we first need to add another constructor q, essentially identical to p, producing the category:

$$\mathcal{C}_3 \equiv (X : \mathsf{Set}) \times (b : X) \times (b = b)^2.$$

Finally, we add the last constructor, expressing the commutativity of p and q:

$$\alpha : p \cdot q = q \cdot p, \tag{3}$$

where $\cdot$ denotes path concatenation (i.e. transitivity of equality).

The parameter F_3 is trivial:

$$\begin{aligned} &F_3 : \mathcal{C}_3 \to \mathsf{Set} \\ &F_3(X) :\equiv 1, \end{aligned}$$

hence we can identify $\int_{\mathcal{C}_3} F$ with $\mathcal{C}_3$.

The target, being a higher constructor of level 2, is more involved. We first define natural transformations

$$\begin{aligned} &l, r : 1 \to U \\ &l_{(X,b,p,q)}(*) :\equiv b \\ &r_{(X,b,p,q)}(*) :\equiv b, \end{aligned}$$

and consider the functor $\mathsf{Eq}\,(l, r)$:

$$\begin{aligned} &\mathsf{Eq}\,(l, r) : \mathcal{C}_3 \to \mathsf{Set} \\ &\mathsf{Eq}\,(l, r)\,(X, b, p, q) :\equiv (b = b). \end{aligned}$$

Now we can define natural transformations $l', r' : 1 \to \mathsf{Eq}\,(l, r)$, corresponding to the two sides of the equality of the constructor α in (3). By Yoneda, such a natural transformation is equivalently expressed as an element of $\mathsf{Eq}\,(l, r)\,(0_{\mathcal{C}_2}) \equiv (b_0 = b_0)$, where $0_{\mathcal{C}_2} \equiv (X_0, b_0, p_0, q_0)$ is the initial object of $\mathcal{C}_3$.

We then take l' to be the natural transformation corresponding to $p_0 \cdot q_0$ and r' the one corresponding to $q_0 \cdot p_0$. It is easy to see that they behave as expected, producing the target $G_3 \equiv \mathsf{Eq}\,(l', r')$:

$$G_3 : \mathcal{C}_3 \to \mathsf{Set}$$
$$G_3(X, b, p, q) :\equiv (p \cdot q = q \cdot p),$$

and resulting in the category of algebras:

$$\mathcal{C}_4 :\equiv (X : \mathsf{Set}) \times (b : X) \times (p, q : b = b) \times (p \cdot q = q \cdot p).$$

7 Limitations and further work

The induction framework presented in this paper is in some ways much more general than existing treatments, but is fundamentally lacking in many aspects:

1. it does not take indexing into account;
2. it only allows covariant dependencies between the types being defined;
3. it does not allow arbitrary inductive-recursive definitions (not even small) with values in one of the types being defined;
4. it only considers 0-truncated types.

We will describe those points in the details in the following, and suggest possible extensions to the framework to address them.

7.1 Taking indexing into account

Definition 2.2 allows to specify dependencies between types, but it does not say whether dependencies are to be intended as indices or simply as inductive-recursive functions.

When we just work internally (therefore modulo equivalence), the distinction does not matter. However, if we are interested in computational properties of the resulting inductive type, we might notice a difference in reduction behaviour.

For example, consider a shape of the form $\mathrm{id} : \mathsf{Set} \to \mathsf{Set}$. The resulting initial algebra can be either regarded as a set together with a family:

$$A : \mathsf{Set}$$
$$B : A \to \mathsf{Set},$$

or as two sets and a function:

$$\begin{aligned} &X : \mathsf{Set} \\ &Y : \mathsf{Set} \\ &f : Y \to X, \end{aligned}$$

Now, suppose the inductive definition contains constructors (using a syntax corresponding to the first interpretation):

$$\begin{aligned} &a_0 : A \\ &b_0 : B(a_0), \end{aligned}$$

in the second interpretation, they would look like:

$$\begin{aligned} &x_0 : X \\ &y_0 : Y \\ &p_0 : f(y_0) = x_0, \end{aligned}$$

and there is no reason why the equality represented by p_0 should be judgemental.

Furthermore, when taking computational properties into account, inductive types are required to be more than just initial algebras. Namely, the usual elimination property is equivalent to requiring that every *algebra fibration* has an algebra section.

Categories of algebras therefore need to be equipped with some fibrational structure in order to make this definition precise, and then we could show that this condition is equivalent to initiality, as it is already known for simple induction [3], induction induction [2], and certain special cases of higher inductive types [19].

One possible solution is to augment shapes with a distinguished acyclic subdiagram, signalling which type dependencies should be realised via indexing. Consequently, oplax cones will have to be Reedy-fibrant with respect to the acyclic portion of the diagram (similarly to the gluing construction in [17]).

7.2 Generalising dependencies between nodes

There is no reason why the dependencies appearing in a shape diagram have to be expressed as functors.

For example, if $F : \mathsf{Set}^{\mathrm{op}} \to \mathsf{Set}$ is a contravariant functor, definition 2.2 does not allow an inductive specification resulting in a pair of types of the form:

$$\begin{aligned} &A : \mathsf{Set} \\ &B : F(A) \to \mathsf{Set}, \end{aligned}$$

which is, for example, perfectly valid in Agda.

Another example which is not covered is provided by certain (small) inductive-recursive definitions with values in one of types being defined. For instance, the following "shape":

$$A : \mathsf{Set}$$
$$B : \mathsf{Set}$$
$$f : A \to B \to A,$$

cannot be realised within the current system.

One idea to generalise the framework to incorporate such definitions is to replace Cat with the generalised multicategory of categories and multivariate functors (as described for example in [11, example 4.2.9]), and shapes with "multidiagrams".

This has the pleasant side-effect of removing the need for the special sort 1, since indexing dependencies could be expressed directly with a nullary functor.

7.3 Generalising to untruncated types

By far the most problematic limitation of the current approach is that it is limited to 0-truncated types. This has the disturbing consequence that path constructors higher than level 1, although possible, are completely inconsequential.

The reason for this limitation, as explained in the introduction, is that the notion of category that we use is not powerful enough to accurately describe the categories of algebras that we work with, nor their functors and natural transformations. In particular, Type itself is not a category in the proper sense.

However, since we make heavy use of containers and container morphisms, which can be defined only in terms of objects, it might seem that it should be possible to get away with a more naive definition of "category", for example, one without the restriction that the morphisms form a set.

Unfortunately, this approach is bound to fail, since the number of coherence properties that we need to impose on morphisms turns out to be equal to the number of constructors. Therefore, even if a simple definition of category including objects, morphisms, composition and associativity would be enough to express inductive specifications with 2 constructors, at step 3 we would be able to define algebras, but not their morphisms.

It is probably impossible to work with an approximated notion of category that does not take all the coherence properties into account. The most promising approach seems to be using semi-Segal types [10] to formalise the tower of coherence conditions on morphisms.

To carry out this plan in full generality, we need to employ untruncated semi-simplicial types, which usually means stepping out of the type theory and work externally in a model of Reedy-fibrant presheaves.

Alternatively, one could just settle on a finite truncation index, and use a finite axiomatisation of semi-Segal types by exploiting the fact that equality over a certain level is trivial.

For example, by setting the truncation level to 1, one would only require three coherence conditions (corresponding to composition, associators and pentagon identities). Level 1 would be already enough to express many interesting higher inductive types, like the circle, the torus, and the Rezk completion.

References

[1] Michael Abott, Thorsten Altenkirch, and Neil Ghani. Containers - Constructing Strictly Positive Types. *Theoretical Computer Science*, 342:3–27, September 2005.

[2] Thorsten Altenkirch, Peter Morris, Fredrik Nordvall Forsberg, and Anton Setzer. A Categorical Semantics for Inductive-Inductive Definitions. In *CALCO*, pages 70–84, 2011.

[3] Steve Awodey, Nicola Gambino, and Kristina Sojakova. Inductive types in homotopy type theory. *ArXiv e-prints*, January 2012.

[4] Paolo Capriotti and Nicolai Kraus. Eliminating Higher Truncations via Constancy.

[5] Aurelio Carboni and Peter Johnstone. Connected Limits, Familial Representability and Artin Glueing. *Mathematical Structures in Computer Science*, 5(4):441–459, 1995.

[6] Peter Dybjer and Anton Setzer. A Finite Axiomatization of Inductive-Recursive Definitions. volume 1581 of *Lecture Notes in Computer Science*, pages 129–146. Springer, 1999.

[7] Peter Dybjer and Anton Setzer. Induction-recursion and initial algebras. *Ann. Pure Appl. Logic*, 124(1-3):1–47, 2003.

[8] Nicola Gambino and Martin Hyland. Wellfounded Trees and Dependent Polynomial Functors. volume 3085 of *Lecture Notes in Computer Science*, pages 210–225. Springer, 2003.

[9] Peter Hancock, Conor McBride, Neil Ghani, Lorenzo Malatesta, and Thorsten Altenkirch. Small induction recursion. In *Typed Lambda Calculi and Applications, 11th International Conference, TLCA 2013*, pages 156–172, 2013.

[10] Yonatan Harpaz. Quasi-unital ∞-categories. *ArXiv e-prints*, September 2012.

[11] Tom Leinster. *Higher Operads, Higher Categories*. August 2004.

[12] Peter Lumsdaine. Higher Inductive Types: a tour of the menagerie. *Homotopy Type Theory Blog*, April 2011.

[13] Peter Lumsdaine and Michael Shulman. Semantics of Higher Inductive Types.

[14] Ieke Moerdijk and Erik Palmgren. Wellfounded Trees in Categories, 1999.

[15] The Univalent Foundations Program. *Homotopy type theory: Univalent foundations of mathematics*. first edition, 2013.

[16] Michael Shulman. Homotopy Type Theory VI. *The n-Category Café*, April 2011.

[17] Michael Shulman. Univalence for inverse diagrams and homotopy canonicity. *ArXiv e-prints*, March 2012.

[18] Michael Shulman. The univalence axiom for elegant Reedy presheaves. *ArXiv e-prints*, July 2013.

[19] Kristina Sojakova. Higher Inductive Types as Homotopy-Initial Algebras. *ArXiv e-prints*, February 2014.

[20] Benno van den Berg and Ieke Moerdijk. W-types in Homotopy Type Theory. *ArXiv e-prints*, July 2013.