The Three-HITs Theorem

Andrej Bauer[*1] **and Niels van der Weide**[†2]

1 **University of Ljubljana, Slovenia**
`Andrej.Bauer@andrej.com`
2 **iCIS, Radboud University, Nijmegen, The Netherlands**
`nweide@cs.ru.nl`

Abstract

We show that all higher inductive types can be constructed from coequalizers, path coequalizers and homotopy colimits. The proof is inspired by Adámek's theorem which constructs inductive types as a colimit of a functor. This way one can reason about all higher inductive types by instead studying a small number of examples.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases homotopy type theory, higher inductive types, category theory

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Higher inductive types (HITs) generalize normal inductive types by allowing constructors for both points and paths, rather than just for the points. While an inductive type is freely generated from a signature, a higher inductive type is freely generated from a signature together with constructors for equations. Numerous examples and definitions of such types have already been given in the literature [3, 6, 8, 24, 25], but a definition with a good metatheory is still lacking. As a step towards that goal, we simplify the definition given in [8] bringing it closer to the intuition and the intended meaning.

Philosophically, one sees an inductive type as a type 'which is built step by step'. Starting with the nonrecursive constructors, new terms are made by applying recursive constructors to previously built terms. This is explained by the fact that inductive types are initial algebras for a functor [16, 17] and that such algebras are obtained by fixed point iteration [2]. Following our intuition on higher inductive types, one would expect that they can be constructed in a similar fashion. However, since equations are also allowed in the specification, identifications need to be made during the construction.

The goal of this paper is to formally justify this idea by showing that the higher inductive types defined in [8] can be generated from three specific higher inductive types, namely coequalizers, path coequalizers, and homotopy colimit. More concretely, we will prove the following theorem.

▶ **Theorem 1** (Three-HITs Theorem). *In Martin-Löf type theory extended with a coequalizers, path coequalizers and homotopy colimits, we can interpret each higher inductive type from [8].*

[*] The author acknowledges the financial support from the Slovenian Research Agency (research core funding No. P1-0294) at Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia. This material is based upon work supported by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF under Award No. FA9550-14-1-0096.
[†] This work was partially supported by EU COST Action Eutypes (CA15123).

This means that for each HIT we can define a type with the same introduction, elimination and computation rules.

Also, this generalizes the results in [15, 19, 20] where the result is showed for truncations. With this result the metatheory of higher inductive types could be simplified significantly, because rather than a general class, one only needs to check metatheoretical properties for three HITs.

In Section 2 we shall recall some of the required material for this paper. More concretely, we give the syntax of higher inductive types, and using that we define required types. Next we define the approximating sequence of a higher inductive type in Section 3, and in Section 4 we show that the colimit of this sequence satisfies the rules of the given HIT. For this we need some lemmata which are proved in Section 5. These are formalized in Coq [10] using the library from [9].

2 A definition of HITs

In this section we provide the definitions of higher inductive types that we shall consider. We recall the scheme for specifying higher inductive types from [8].

▶ **Definition 2.** A *polynomial (expression)* in a type variable X is a built inductively from the variable X, type constants C, binary products $\times$, and binary sums $+$.

For example, if N is the type of natural numbers, the expression $(X + \mathsf{N}) \times X$ is a polynomial expression in X. We write $P[X]$ to indicate a polynomial in variable X. Given a type A, we may substutute A for X to obtain a type $P[A]$, as follows:

$$X[A] :\equiv A, \qquad (P_1 \times P_2)[A] :\equiv P_1[A] \times P_2[A],$$
$$C[A] :\equiv C, \qquad (P_1 + P_2)[A] :\equiv P_1[A] + P_2[A].$$

In a similar fashion a map $f : A \to B$ is transformed to a map $P[f] : P[A] \to P[B]$:

$$X[f] :\equiv f, \qquad (P_1 \times P_2)[f]\,(x, y) :\equiv (P_1[f]\,x, P_2[f]\,y),$$
$$C[f] :\equiv \mathsf{id}_C, \qquad (P_1 + P_2)[f](\mathsf{inl}\,x) :\equiv \mathsf{inl}\,(P_1[f]\,x),$$
$$(P_1 + P_2)[f](\mathsf{inr}\,y) :\equiv \mathsf{inr}\,(P_2[f]\,y).$$

Somewhat less obviously, a polynomial acts on a type family $B : A \to \textsc{Type}$ to give its lifting $\overline{P}[B] : P[A] \to \textsc{Type}$,

$$\overline{X}[B]\,u :\equiv B\,u, \qquad \overline{P_1 \times P_2}[B]\,u :\equiv (\overline{P_1}[B]\,u) \times (\overline{P_2}[B]\,u),$$
$$\overline{C}[B]\,u :\equiv C, \qquad \overline{P_1 + P_2}[B]\,(\mathsf{inl}\,u) :\equiv \overline{P_1}[B]\,u,$$
$$\overline{P_1 + P_2}[B]\,(\mathsf{inr}\,u) :\equiv \overline{P_2}[B]\,u,$$

while a map $f : \prod_{x\,A} B_1\,x \to B_2\,x$ lifts to $\overline{P}[f] : \prod_{u:P[A]} \overline{P}[B_1]\,u \to \overline{P}[B_2]\,u$:

$$\overline{X}[f]\,u :\equiv f\,u \qquad \overline{P_1 \times P_2}[f]\,(u_1, u_2) :\equiv (\overline{P_1}[f]\,u_1, \overline{P_2}[f]\,u_2)$$
$$\overline{C}[f]\,u :\equiv u \qquad \overline{P_1 + P_2}[f]\,(\mathsf{inl}\,u) :\equiv \overline{P_1}[f]\,u,$$
$$\overline{P_1 + P_2}[f]\,(\mathsf{inr}\,u) :\equiv \overline{P_2}[f]\,u.$$

There is of course a strong analogy between the action of $P[X]$ and a polynomial functor, but we hesitate to call $P[X]$ a functor as there is no category to speak of, and in any case the manipulations we have just described are syntactic.

Next we define the notion of a constructor term.

▶ **Definition 3.** Given a function $c : A\,T \to T$ with A a polynomial and T a type, we say t is a *constructor term* over c if we can find polynomials F and G such that $x : F\,T \vdash t : G\,T$ can be derived using the following rules.

$$\frac{t : B \qquad T \text{ does not occur in } B}{x : F\,T \vdash t : B} \qquad \frac{}{x : F\,T \vdash x : F\,T} \qquad \frac{x : F\,T \vdash r : A\,T}{x : F\,T \vdash c\,r : T}$$

$$\frac{j \in \{1,2\} \qquad x : F\,T \vdash r : G_1\,T \times G_2\,T}{x : F\,T \vdash \pi_j\,r : G_j\,T} \qquad \frac{j \in \{1,2\} \qquad x : F\,T \vdash r_j : G_j}{x : F\,T \vdash (r_1, r_2) : G_1\,T \times G_2\,T}$$

$$\frac{j \in \{1,2\} \qquad x : F\,T \vdash r : G_j\,T}{x : F\,T \vdash \mathsf{in}_j\,r : G_1\,T + G_2\,T}$$

Using constructor terms we give the following scheme of higher inductive types.

▶ **Definition 4.** A *higher inductive type* is defined according to the following scheme

```
Inductive $H$ :=
| $c : A\,H \to H$
| $p_i : \prod(x : B_i\,H), t_i = r_i \qquad (i = 1, \ldots, m)$
```

where A and each B_i are polynomials, and each t_i and r_i are constructor terms over c of type H with $x : B_i\,H$ as variable.

Before we can give the rules for higher inductive types, we need to define the lift of a constructor term.

▶ **Definition 5.** Given is a constructor $c : A\,H \to H$, a type family $Y : H \to \textsc{Type}$, and a term $f : \prod(x : A\,H), \bar{A}\,Y\,x \to Y(c\,x)$. For a constructor term $x : F\,H \vdash r : G\,H$ we define the *lift* $\widehat{r}$ *of* r with type $x : F\,H, h_x : \bar{F}\,Y\,x \vdash \widehat{r} : \bar{G}\,Y\,r$ by induction in r as follows.

$$\widehat{t} := t \qquad \widehat{x} := h_x \qquad \widehat{c_i\,r} := f_i\,r\,\widehat{r}$$

$$\widehat{\pi_j\,r} := \pi_j\,\widehat{r} \qquad \widehat{(r_1, r_2)} := (\widehat{r_1}, \widehat{r_2}) \qquad \widehat{\mathsf{in}_j\,r} := \widehat{r}$$

With all these notions we can give the introduction, elimination and computation rules of higher inductive types. The introduction rules for H as given in Definition 4 are

$$c : A\,H \to H,$$

$$p_i : \prod(x : B_i\,H), t_i = r_i.$$

We also have an elimination rule for which we use the lifting of constructor terms.

$$\frac{\begin{array}{c}\vdash Y : H \to \textsc{Type} \\ \vdash f : \prod(x : A\,H), \bar{A}\,Y\,x \to Y\,(c\,x) \qquad \vdash q_i : \prod(x : B_i\,H)(h_x : \bar{B}_i\,Y\,x), \widehat{t_i} =_{p_j\,x} \widehat{r_j}\end{array}}{\vdash H\mathrm{rec}(f, q_1, \ldots, q_n) : \prod(x : H), Y\,x}$$

Let us abbreviate $H\mathrm{rec}(f, q_1, \ldots, q_n)$ by $H\mathrm{rec}$. The type H also has computation rules for each point $t : A\,H$

$$H\mathrm{rec}\,(c\,t) \equiv f\,t\,(\bar{A}\,H\mathrm{rec}\ t),$$

and for each $a : B_i\,H$

$$\mathrm{apD}\ H\mathrm{rec}\,(p_i\ a) \equiv q_i\,a\,(\bar{B}_i\,H\mathrm{rec}\ a).$$

Note that these equalities are definitional rather than propositional.

Let us now give some examples of higher inductive types which will be crucial in this paper. The first one identifies points in some type, and we call it the *coequalizer*.

```
Inductive coeq (A, B : Type) (f, g : A → B) :=
| inC : B → coeq A B f g
| glueC : ∏(a : A), inC (f a) = inC (g a)
```

The elimination rule for this type is as follows

$$\frac{\vdash Y : \mathsf{coeq}\ A\ B\ f\ g \to \textsc{Type} \qquad \vdash i_Y : \prod(b : B), Y\ (\mathsf{inC}\ b) \qquad \vdash g_Y : \prod(a : A), \mathsf{glueC}_*(i_Y\ (f\ a)) = i_Y\ (g\ a)}{\vdash \mathsf{coeqrec}(i_Y, g_Y) : \prod(x : \mathsf{coeq}\ A\ B\ f\ g), Y\ x}$$

Note the similarities with the definition of the coequalizer in category theory [22].

Next we define a type which identifies two paths in some type which we shall call the *path coequalizer.*

```
Inductive pcoeq (A, B : Type) (p : A → ∑(b₁, b₂ : B), (b₁ = b₂) × (b₁ = b₂)) :=
| inP : B → pcoeq A B f g
| glueP : ∏(a : A), ap inP (π₁(π₃(p a))) = ap inP (π₂(π₃(p a)))
```

We will abbreviate $p_i\ a = \pi_i(\pi_3(p\,a))$ for $i = 1, 2$. Note that this type has 2-paths, so it does not follow the given syntax. Furthermore, the identified paths used in the parameter are obliged to have the same endpoints. The introduction and computation rules are as expected, but we will give the elimination rule.

For the elimination rule we first need a lemma.

▶ **Lemma 6.** *Given is* $Y : \mathsf{pcoeq} \to \textsc{Type}$ *and* $i_Y : \prod(b : B), Y(\mathsf{inP}\ b)$. *Then we have a term*

$$\mathrm{coh} : \prod(a : A), (\pi_1\ a)_*^{\lambda b, Y(\mathsf{inP} b)}(i_Y\ b_1) = (\pi_2\ a)_*^{\lambda b, Y(\mathsf{inP} b)}(i_Y\ b_1)$$

Proof. For this we need to prove that for all type families $Y : A \to \textsc{Type}$, maps $f : A \to B$, points $z : Y(f\ a)$, and paths $p : a = b$ we have

$$\mathrm{tc} : p_*^{\lambda x, Y(f\ x)}\ z = (\mathrm{ap}\ \ f\ p)_*^{Y}\ z.$$

This can be proven with path induction. Now we have the following equalities

$$\begin{aligned}(\pi_1\ a)_*^{\lambda b, Y(\mathsf{inP} b)}(i_Y\ b_1) &= (\mathrm{ap}\ \ \mathsf{inP}\ p_1)_*^{Y}\ (i_Y\ b_1)\\ &= (\mathrm{ap}\ \ \mathsf{inP}\ p_2)_*^{Y}\ (i_Y\ b_1)\\ &= (\pi_2\ a)_*^{\lambda b, Y(\mathsf{inP} b)}(i_Y\ b_1)\end{aligned}$$

The first and the last follow from tc, and the second from glueP. ◀

Now we give the elimination rule of pcoeq for which we use coh.

$$\frac{\vdash Y : \mathsf{pcoeq}\ A\ B\ f\ g \to \textsc{Type} \qquad \vdash i_Y : \prod(b : B), Y\ (\mathsf{inP}\ b) \qquad \vdash g_Y : \prod(a : A), \mathrm{coh}^{-1} \bullet \mathrm{apD}\ \ i_Y\ (p_1\ a) = \mathrm{apD}\ \ i_Y\ (p_2\ a)}{\vdash \mathsf{pcoeqrec}(i_Y, g_Y) : \prod(x : \mathsf{pcoeq}\ A\ B\ f\ g), Y\ x}$$

The last HIT we need is the homotopy colimit, and for that we again are inspired by the definition from category theory.

```
Inductive hocolim (F : ℕ → Type) (f : ∏(n : ℕ, F n → F(n + 1))) :=
| inc : ∏(n : ℕ), F n → hocolim F f
| com : ∏(n : ℕ)(x : F n), inc n x = inc (n + 1) (f n x)
```

Now the elimination rule is as follows.

$$\frac{\begin{array}{c}\vdash Y : \mathsf{hocolim}\ F\ f \to \textsc{Type} \\ \vdash i_Y : \prod(n : \mathbb{N})(x : F\ n), Y\ (\mathsf{inc}\ n\ x) \\ \vdash c_Y : \prod(n : \mathbb{N})(x : F\ n), \mathsf{com}_*(i_Y\ n\ x) = i_Y\ (n+1)\ (f\ n\ x)\end{array}}{\vdash \mathsf{hocolimrec}(i_Y, c_Y) : \prod(x : \mathsf{hocolim}\ F\ f), Y\ x}$$

Lastly, we need function extensionality for the proof. Let us first introduce some notation.

▶ **Definition 7.** Let A, B be types, let $f, g : A \to B$ be maps, and let $x : A$ be a point of A. Then for all $e : f = g$ we define a path $e \,\square\, x : f\ x = g\ x$ using path induction which sends refl to refl.

▶ **Definition 8.** Given are types A, B and two maps $f, g : A \to B$. Then we define the type FunExt, of which the inhabitants represent proofs of function extensionality, as follows

$$\mathsf{FunExt} = \left(\prod(x : A), f\ x = g\ x\right) \to f = g.$$

We will assume that we have an inhabitant $\mathsf{FE} : \mathsf{FunExt}$ such that for $p : \prod(x : A), f x = g x$ and $x : A$ we have

$$(\mathsf{FE}\ p) \,\square\, x = p\ x.$$

In homotopy type theory function extensionality can be proved by assuming univalence [21] or by assuming the existence of an interval object [23].

3 The Approximator

Let us assume that some higher inductive type H is given. In order to construct H as a colimit, we first need to give the approximations in the colimit, and for that we define the *approximator*.

Before giving the definition, let us acquire inspiration by looking in more detail at how an inductive type is constructed. Such a type T is defined by a constructor $c : A\,T \to T$ with A polynomial. Equivalently, we can define the type T with a signature which is a polynomial functor A, and then by Adámek's theorem T is the following colimit.

$$\begin{tikzcd} \mathbf{0} \arrow[r] & A\,\mathbf{0} \arrow[r] & A(A\,\mathbf{0}) \arrow[r] & \ldots \end{tikzcd}$$

To understand what this does, let us assume that $A\,X = 1 + X$, so that $T = \mathbb{N}$. This means we have two inclusions $1 \longrightarrow 1 + X$ and $X \longrightarrow 1 + X$, and we call them 0_C and S_C respectively. At every step we formally add for each $x : X$ a successor $S_C\ x$, and we add 0_C. Repeatedly applying this construction to the empty type $\mathbf{0}$ gives the natural numbers $\mathbb{N}$.

One would like to construct higher inductive types in a similar fashion. The first difference is that instead of starting with $\mathbf{0}$, we start with all the nonrecursive constructors. Also, in this construction, the functor A is applied at every step. Instead we will add terms built from the recursive constructors using previously constructed terms as arguments. Since extra equalities might be present in the higher inductive type, we need to make identifications during the construction. Rather than just adding points at every step, we also need to glue the specified paths.

To understand more precisely what should be done, let us consider an example.

```
Inductive $\mathbb{N}_2$ :=
| $0 : \mathbb{N}_2$
| $S : \mathbb{N}_2 \to \mathbb{N}_2$
| $p : S(S\,0) = 0$
```

We build a sequence F of approximations. The first approximation just has a constructor 0, and after that we add a constructor for $S\,0$ to obtain the second approximation. Continuing this way, we ultimately arrive at the third approximation, which we call $F'\,3$, where we found inhabitants 0, $S\,0$, and $S(S\,0)$. Now we can make the first identification, and to do so, we take the following homotopy pushout

$$\begin{tikzcd}
\mathbf{1}+\mathbf{1} \arrow[r, "\delta"] \arrow[d, "0+S(S\,0)"'] & I^1 \arrow[d] \\
F'\,3 \arrow[r] & F\,3
\end{tikzcd}$$

to obtain the actual third approximation $F\,3$, and we continue our construction with that one. Note that to glue during the nth step, we need refer to elements from the $(n-2)$th step. So, in order to do the identification of the nth step, inhabitants from a previous approximation need to be used.

Due to the usage of constructor terms, one always has to go back a fixed number of steps. By extending the syntax, it might be needed to go back an arbitrary amount of steps. This happens in the following example.

```
Inductive $H$ :=
| $c : H$
| $f : H \to H$
| $p : \prod(n : \mathbb{N}), f^n\,0 = 0$
```

This example might seem like it is permitted by the syntax, but actually it is not. The function f^n is defined as a polymorphic map, and using it requires the type H as an argument. In constructor terms one cannot use the type, and hence this definition is not allowed An extension with such types will make the construction more complicated, and thus they will not be considered in this paper.

Let us make this idea formal, and for that we start with a higher inductive type given as follows.

```
Inductive $H$ :=
| $c : A_{\text{nonrec}} \to H$
| $c_{\text{rec}} : A_{\text{rec}}\,H \to H$
| $p_i : \prod(x : B_i\,H), t_i = r_i \qquad (i = 1, \ldots, m)$
```

Note that the nonrecursive and recursive point constructors are separated in this definition. The first approximation will be given using the nonrecursive constructors.

```
Inductive $H_{\text{nonrec}}$ :=
| $c'_{\text{nonrec}} : A_{\text{nonrec}} \to H_{\text{nonrec}}$
```

Next we need to generate the other approximations, and that will be done in two steps. First, we note that types can be extended with a recursive constructor.

```
Inductive $H_{\text{rec}}$ $(P : \text{Type})$:=
| $c'_{\text{rec}} : A_{\text{rec}}\,P \to H_{\text{rec}}\,P$
```

To do the identifications, we need to be able to interpret the constructor terms. For that we use that each constructor terms only uses each constructor a finite amount of times, and thus there is a maximum number $n \geq 1$ of times a constructor is used. In order to define the approximator, we start with a type $H^n_{\mathbf{Con}}$ in which all the constructor terms can be interpreted.

Inductive $H^n_{\mathbf{Con}}\ (P : \text{TYPE}) :=$
$|\ \mathbf{term} : P + H_{\text{rec}}\ P + \ldots + H^n_{\text{rec}}\ P \to H^n_{\mathbf{Con}}\ P$

Note that this type is actually a labeled sum. This is because $H_{\text{rec}}\ P$ is isomorphic to $A_{\text{rec}}\ P$, and thus this $H^n_{\mathbf{Con}}\ P$ is actually $P + A_{\text{rec}}\ P + \ldots + A^n_{\text{rec}}\ P$. Hence, we do not need inductive types in general for the construction, but we just need to have sums and products. If n is clear from the context, then we shall not write it down. From Lemma 11 we can conclude that we can interpret the constructor terms in $H_{\mathbf{Con}}\ P$

Now we have sufficient to define the approximator. Given is the following data

- Types P, Q, R;
- A map $j_Q : H_{\mathbf{Con}}\ Q \to P$;
- An inclusion $j_R : H_{\mathbf{Con}}\ R \to Q$;
- Paths $p_Q : \prod(x : Q), j_Q(\overline{t_i}\ x) = j_Q(\overline{r_i}\ x)$;
- Paths $p_R : \prod(x : R), j_R(\overline{t_i}\ x) = j_R(\overline{r_i}\ x)$.

The type $H_{\mathbf{Approx}}$ depends on the given data, but for clarity we will suppress it from the notation. We start with $H_{\mathbf{Con}}\ P$.

First, we need to guarantee that the added paths have the right endpoints. Note that we have two maps $\overline{t_i}, \overline{r_i} : B_i\ P \to H_{\mathbf{Con}}\ P$, and that gives the first coequalizer.

$$\begin{tikzcd} B_i\ P \arrow[r, shift left, "\overline{t_i}"] \arrow[r, shift right, "\overline{r_i}"'] & H_{\mathbf{Con}}\ P \arrow[r, "\mathsf{inC}_1"] & C_1 \end{tikzcd}$$

However, this is not exactly the type we want due to coherency issues. For example, in P we have terms using c_{rec} at most k times, so in $H_{\mathbf{Approx}}\ P$ we have duplicates of terms using c_{rec} less than k times.

Remember that we we have $j_Q : H_{\mathbf{Con}}\ Q \to P$. Now we define the type C_2 which is the coequalizer of the following two arrows.

$$\begin{tikzcd} H_{\mathbf{Con}}\ Q \arrow[r, shift left, "\mathsf{in}_1 \circ j_Q"] \arrow[r, shift right, "H_{\mathbf{Con}}\ (j_Q \circ \mathsf{in}_1)"'] & H_{\mathbf{Con}}\ P \arrow[r, "\mathsf{inC}_1"] & C_1 \end{tikzcd}$$

To explain what this does, let us assume for a moment that we have one construct c with arity 1. For $c'\ x$ with $x : Q$, the upper map is the inclusion of $H_{\mathbf{Con}}\ Q$ on the first coordinate of $H_{\mathbf{Con}}\ P$. The lower map sends $c'\ x$ to $c'\ (j_Q\ (\mathsf{in}_1\ x))$, so it replaces x by its image in P.

Lastly, we define $H_{\mathbf{Approx}}$, and for this we make one more identification. Instead of identifying points, we will identify paths in this step. Since also paths are added during the construction, there might also be duplicated paths. The resulting type should be free, so these paths should be identified. We define a map $q : B_i\ R \to \sum(x, y : P), (x = y) \times (x = y)$ sending $x : B_i\ R$ to the pair

$$(\overline{t_i}\ x, \overline{r_i}\ x, \mathrm{ap}\ (j_Q \circ \mathsf{in}_1)\ (p_R\ x), p_Q\ (B_i\ j_R\ x))$$

Concluding the construction, we define $H_{\mathbf{Approx}}$ as $\mathsf{pcoeq}\ (B_i\ R)\ C_2\ q$.

Now we can define the approximating sequence.

▶ **Definition 9.** In the setting as described, we simultaneously define a sequence of approximations $F : \mathbb{N} \to \textsc{Type}$ to H and maps $f : \prod(n : \mathbb{N}), F\,n \to F(n+1)$ as follows.

- We define $F\,0 = H_{\text{nonrec}}$.
- We define $F\,1 = H_{\mathbf{Approx}}$ taking $P = F\,0$ and $Q, R = \mathbf{0}$ and the maps are defined by $\mathbf{0}$rec.
- We define $F\,2 = H_{\mathbf{Approx}}$ with $P = F\,1, Q = F\,0$ and $R = \mathbf{0}$. We have a map $Q \to P$ defined to be the composition of all inC with in_1.
- We define $F(n+3) = H_{\mathbf{Approx}}$ with $P = (F(n+2)$, $Q = F(n+1)$, and $R = F\,n$. The maps $Q \to P$ and $R \to Q$ are defined to be the composition of inC with in_1 as in the construction. The paths p_Q and p_R are given by the path glueC from the first coequalizer.

For the maps $f\,n : F\,n \to F(n+1)$, note that we always have the following sequence of maps

$$F\,n \longrightarrow H_{\mathbf{Con}}\,(F\,n) \longrightarrow C_1 \longrightarrow \ldots \longrightarrow C_4 \longrightarrow H_{\mathbf{Approx}}.$$

Taking P to be $F\,n$, then we have $F(n+1) = H_{\mathbf{Approx}}(F\,n)$, and thus the composition gives the map $F\,n \to F(n+1)$.

4 The Rules

Now we have defined an object $\mathsf{hocolim}\ F\ f$, which is supposed to interpret the higher inductive type. In order to finish the proof of Theorem 1, we need to show that it satisfies the rules. This means that we have to make functions which interpret the introduction rules, and an eliminator such that the computation rules are satisfied. We will do this step by step, and refer to lemmata in Section 5 when needed.

4.1 Introduction Rules

In order to show that this is the desired type, we first show that it has the correct introduction rules. These come in three flavors: the nonrecursive and the recursive points, and the paths.

Let us start by defining a map $A_{\text{nonrec}} \to \mathsf{hocolim}\,F\,f$ which gives the introduction rule for the nonrecursive point constructor. Since $F\,0$ is defined by H_{nonrec}, which has a constructor $c'_{\text{nonrec}} : A_{\text{nonrec}} \to H_{\text{nonrec}}$, this can be defined by the following composition.

$$A_{\text{nonrec}} \xrightarrow{c'_{\text{nonrec}}} F\,0 \xrightarrow{\mathsf{inc}\ 0} \mathsf{hocolim}\ F\ f$$

Next we show that we also have the recursive point constructor meaning that we have a map $A_{\text{rec}}\,(\mathsf{hocolim}\ F\ f) \to \mathsf{hocolim}\ F\ f$. This is slightly more complicated, and for that we first need a lemma which says that colimits over $\mathbb{N}$ commute with polynomials.

▶ **Lemma 10.** *The types $A\,(\mathsf{hocolim}\ F\ f)$ and $\mathsf{hocolim}\ (A \circ F)\ (A\ f)$ are isomorphic for all polynomials A.*

Proof. If A is constant then it follows from Lemma 13. For the identity it is trivial, and for sums and products, it follows from Lemmata 14 and 15 respectively. ◀

Now we will construct the map $A_{\text{rec}}\,(\mathsf{hocolim}\ F\ f) \to \mathsf{hocolim}\ F\ f$, and by Lemma 10 it suffices to make a map $\mathsf{hocolim}\ (A_{\text{rec}} \circ F)\ (A_{\text{rec}}\ f) \to \mathsf{hocolim}\ F\ f$. For this we use the recursion rule of $\mathsf{hocolim}$, and we start with the following string of maps

$$A_{\text{rec}}(F\,n) \xrightarrow{c'_{\text{rec}}} H_{\text{rec}}(F\,n) \xrightarrow{\mathbf{term} \circ \mathsf{in}_2} H_{\mathbf{Con}}(F\,n) \xrightarrow{\mathsf{inC}} H_{\mathbf{Approx}}(F\,n) = F(n+1)$$

Composing this map with inc, gives maps $A_{\text{rec}}(F\ n) \to \mathsf{hocolim}\ F\ f$ for all $n : \mathbb{N}$.

Next we need to show the commutativity of the following triangle.

$$\begin{tikzcd}
A_{\text{rec}}(F\ n) \arrow[rr, "A_{\text{rec}}(f\ n)"] \arrow[dr] & & A_{\text{rec}}(F(n+1)) \arrow[dl] \\
& \mathsf{hocolim}\ F\ f &
\end{tikzcd}$$

Before we continue, let us recall that inductive types are functors. Suppose, we have an inductive type T with a parameter P, then a function $f : P \to Q$ gives a function $T\ f : T\ P \to T\ Q$. Let us start with the following rectangle

$$\begin{tikzcd}
A_{\text{rec}}(F\ n) \arrow[r, "c'_{\text{rec}}"] \arrow[d, "A_{\text{rec}}(f\ n)"'] & H_{\text{rec}}(F\ n) \arrow[r, "\mathbf{term} \circ \mathsf{in}_2"] \arrow[d, "H_{\text{rec}}(f\ n)"'] & H_{\mathbf{Con}}(F\ n) \arrow[d, "H_{\mathbf{Con}}(f\ n)"'] \\
A_{\text{rec}}(F(n+1)) \arrow[r, "c'_{\text{rec}}"'] & H_{\text{rec}}(F(n+1)) \arrow[r, "\mathbf{term} \circ \mathsf{in}_2"'] & H_{\mathbf{Con}}(F(n+1))
\end{tikzcd}$$

The left square commutes, because by definition of $H_{\text{rec}}\ f$ we have

$$H_{\text{rec}}\ (f\ n)\ (c'_{\text{rec}} x) = c'_{\text{rec}}(A_{\text{rec}}\ (f\ n)\ x).$$

For a similar reason, the right square commutes as well. Hence, it suffices to show that the following diagram commutes.

$$\begin{tikzcd}
H_{\mathbf{Con}}(F\ n) \arrow[r, "\mathsf{inC}"] \arrow[d, "H_{\mathbf{Con}}\ (f\ n)"'] & F(n+1) \arrow[d, "f(n+1)"] \\
H_{\mathbf{Con}}(F(n+1)) \arrow[r, "\mathsf{inC}"'] & H_{\mathbf{Approx}}\ (F(n+1))
\end{tikzcd}$$

Note that $H_{\mathbf{Approx}}(F(n+1)) = F(n+2)$ and $H_{\mathbf{Approx}}(F\,n) = F(n+1)$. Hence, this diagram commutes, because of the coherency added in C_2.

Next we need to define the introduction rules for the paths. Let us start by extending the maps $t_i, r_i : B_i\ (\mathsf{hocolim}\ F\ f) \to \mathsf{hocolim}\ F\ f$. Since homotopy colimits commute with polynomials, it suffices to make a map $\mathsf{hocolim}\ (B_i\ F)\ (B_i\ f) \to \mathsf{hocolim}\ F\ f$, and for this we need to start by defining maps $B_i(F\ n) \to \mathsf{hocolim}\ F\ f$. Note that by construction we always have maps $\overline{t_i}, \overline{r_i} : B_i(F\ n) \to H_{\mathbf{Con}}(F\ n)$, and thus we obtain maps $B_i(F\ n) \to F(n+1)$ by composing these with inC.

Now we need to check the commutativity of the following diagram

$$\begin{tikzcd}
B_i(F\ n) \arrow[r, "\overline{t_i}"] \arrow[d, "B_i(f\ n)"'] & H_{\mathbf{Con}}(F\ n) \arrow[r, "\mathsf{inC}"] \arrow[d, "H_{\mathbf{Con}}(f\ n)"] & C_2(F\ n) \arrow[r, "\mathsf{inC}"] & F(n+1) \arrow[d, "f\ n"] \\
B_i(F(n+1)) \arrow[r, "\overline{t_i}"'] & H_{\mathbf{Con}}(F(n+1)) \arrow[r, "\mathsf{inC}"'] & C_2(F(n+1)) \arrow[r, "\mathsf{inC}"'] & F(n+2)
\end{tikzcd}$$

The right rectangle commutes, because the maps inC are built by composing constructors of the coequalizer, and thus it commutes by definition. Now it computes overall due to Lemma 12 which says that $\mathsf{inC}(\overline{t_i}(B_i(f\ n)\ x))$ and $\mathsf{inC}(H_{\mathbf{Con}}(f\ n)(\overline{t_i}\ x))$ are equal for all $x : B_i(F\ n)$.

In order to show that we indeed have paths p_i, we use the eliminator of the homotopy colimit. We need to show that for all $x : \mathsf{hocolim}\ F\ f$ we have $t_i\ x = r_i\ x$. First, we need to give for all $x : F\,n$ an inhabitant of $t_i\ (\mathsf{inc}\,n\,x) = r_i\ (\mathsf{inc}\,n\,x)$. By the first identification we have a path $\mathsf{glueC}\,x$ between $\overline{t_i}\,x$ and $\overline{r_i}\,x$, and thus the right path is $\mathrm{ap}\ (\mathsf{inc}\,n)\ (\mathsf{glueC}\,x)$.

To finish the proof, we need to show that

$$(\mathsf{com}\,n\,x)_*(\mathrm{ap}\ (\mathsf{inc}\,n)\ (\mathsf{glueC}\,x)) = \mathrm{ap}\ (\mathsf{inc}\ (n+1))\ (\mathsf{glueC}\ ((B_i\ (f\ n))\ x)).$$

This follows from the following computation.

$$\begin{aligned}&\mathrm{ap}\ (\mathsf{inc}\ (n+1))\ (\mathsf{glueC}\ ((B_i\ (f\ n))\ x))\\ &\quad= \mathrm{ap}\ (\mathsf{inc}\ (n+1))\ (\mathrm{ap}\ (f\ n)\ \mathsf{glueC}\,x))\\ &\quad= \mathrm{ap}\ (\mathsf{inc}\ (n+1)\circ (f\ n))\ \mathsf{glueC}\,x))\\ &\quad= \mathsf{com}_*\ (\mathrm{ap}\ (\mathsf{inc}\,n)\ (\mathsf{glueC}\,x))\end{aligned}$$

The first step follows from the third added coherency, the second from Lemma 2.2.2 in [25], and the last step from Lemma 16.

4.2 Elimination Rule

For the next step we define the right eliminator for $\mathsf{hocolim}\,F\,f$, so suppose that we are given the following data

$$Y : \mathsf{hocolim}\ F\ f \to \textsc{Type},$$

$$c_{Y,\mathrm{nonrec}} : \prod(a : A_{\mathrm{nonrec}}), Y(c_{\mathrm{nonrec}}\ a),$$

$$c_{Y,\mathrm{rec}} : \prod(x : A_{\mathrm{rec}}\ \mathsf{hocolim}\ F\ f), \overline{A_{\mathrm{rec}}}\ Y\ x \to Y(c_{\mathrm{rec}}\ x),$$

$$q_{Y,i} : \prod(x : B_i\ H)(h_x : \overline{B_i}\ Y\ x), \widehat{t_i} =_{p_j\ x} \widehat{r_j}.$$

In order to make a map $h : \prod(x : \mathsf{hocolim}\ F\ f), Y\ x$, we use the induction principle of $\mathsf{hocolim}\,F\,f$, and for that we first need to make maps $h\,n : \prod(x : F\ n), Y(\mathsf{inc}\,n\,x)$ for $n : \mathbb{N}$.

We start by making a map $h\,0 : \prod(x : F\ 0), Y(\mathsf{inc}\,0\,x)$. Recall that $F\,0$ was defined to be H_{nonrec} which only has a constructor $c'_{\mathrm{nonrec}} : A_{\mathrm{nonrec}} \to H_{\mathrm{nonrec}}$. So, let us assume that we have $a : A_{\mathrm{nonrec}}$. Since $c_{\mathrm{nonrec}}\ a = \mathsf{inc}\,0\ (c'_{\mathrm{nonrec}}\ a)$, it suffices to find an inhabitant of $Y(c_{\mathrm{nonrec}}\ a)$, and for that we take $c_{Y,\mathrm{nonrec}}\ a$. Hence, we get a map $h\,0$.

Now suppose that we have a map $h\,n : \prod(x : F\ n), Y(\mathsf{inc}\,n\,x)$, and our goal is to make a map $h(n+1) : \prod(x : F(n+1)), Y(\mathsf{inc}\ (n+1)\ x)$. In order to do so, we will first look at how to extend $h\,n$ to a map $\prod(x : H_{\mathbf{Con}}(F\ n)), Y(\mathsf{inc}\ (n+1)\ (\mathsf{inC}\ x))$.

Let us do this in the general case. Suppose, we have a map $g : P \to \mathsf{hocolim}\ F\ f$, and that we already constructed $\varphi : \prod(x : P), Y(g\ x)$. Our goal is to extend the map into $\varphi' : \prod(x : H_{\mathrm{rec}}P), Y(c_{Y,\mathrm{rec}}\ x)$, and for that we use H_{rec}-induction. Note that for each $c'_{\mathrm{rec}}\ x$ with $x : A_{\mathrm{rec}}\ P$ we have the type $Y(c_{\mathrm{rec}}\ (A_{\mathrm{rec}}\ g\ x))$, and thus we have a type family on H_{rec}. Now let $x : A_{\mathrm{rec}}\ P$ and $y : \overline{A_{\mathrm{rec}}}\ Y\ x$ be given. Then we need to give an element of the type $Y(c_{\mathrm{rec}}\ (A_{\mathrm{rec}}\ g\ x))$ for which we take $c_{Y,\mathrm{rec}}\ (A_{\mathrm{rec}}\ g\ x)\ y$. We will often leave the argument $A_{\mathrm{rec}}\,g\,x$ implicit.

Now we can also extend the map to $H^n_{\mathbf{Con}}\ P$, because we can define this map on each component. On the component P it is just φ, and on the other components we define it via extension. We call this map h_1.

Next we need to extend the map to $H_{\mathbf{Approx}}$. This is done by the universal mapping property of the coequalizer, and that requires some steps. First, to give an paths between the images of $t_i\,x$ and $r_i\,x$. Note that the images of these are $\widehat{t_i}\,x$ and $\widehat{r_i}\,x$ respectively, and then $q_{Y,i}$ gives the desired paths. This gives an extension h_2 to C_1.

Next we check the coherency conditions, and we start with the first. We need to check that $\mathsf{in}_1(\mathsf{inC}\,x)$ and $H_{\mathbf{Con}}\,\mathsf{inC}\,x$ get mapped to the same element for $x : H_{\mathbf{Con}}(F\,n)$ by h_2. To do so, we use a case distinction on $H_{\mathbf{Con}}(F\,n)$, and for the case $x \equiv \mathsf{in}_1\,y$ with $y : F\,n$, we have the following equations

$$h_2\;(\mathsf{inC}(\mathsf{in}_1(\mathsf{inC}\;(\mathsf{in}_1\;y)))) \equiv h_1\;(\mathsf{in}_1(\mathsf{inC}(\mathsf{in}_1\;y))),$$

$$\begin{aligned} h_2\;(\mathsf{inC}(H_{\mathbf{Con}}\;(\mathsf{inC}\circ\mathsf{in}_1)\;(\mathsf{in}_1\;y))) &\equiv h_1\;(H_{\mathbf{Con}}\;(\mathsf{inC}\circ\mathsf{in}_1)\;(\mathsf{in}_1\;y)) \\ &\equiv h_1\;(\mathsf{in}_1(\mathsf{inC}(\mathsf{in}_1\;y))) \end{aligned}$$

Since they are definitionally equal, we can just use the path refl.

For the other cases, we take a look at $c'_{\text{rec}}\;y$, and for y we assume that $h_1(\mathsf{in}_1(\mathsf{inC}\,y)) = h_1(H_{\mathbf{Con}}\;(\mathsf{inC}\circ\mathsf{in}_1)\;y)$. Note that at each level h is defined using the eliminator of the coequalizer. This means that $h\,n\;(\mathsf{inC}\;(c'_{\text{rec}}\;y))$ and $c_{Y,\text{rec}}(h\,n\;(\mathsf{inC}\,y))$ are equal by definition, and thus we can make the following computations.

$$\begin{aligned} h_1\;(\mathsf{in}_1(\mathsf{inC}\;(c'_{\text{rec}}\;y))) &\equiv h\,n\;(\mathsf{inC}\;(c'_{\text{rec}}\;y)) \\ &= c_{Y,\text{rec}}(h\,n\;(\mathsf{inC}\;y)) \end{aligned}$$

$$\begin{aligned} h_1\;(H_{\mathbf{Con}}\;(\mathsf{inC}\circ\mathsf{in}_1)\;(c'_{\text{rec}}\;y)) &\equiv h_1\;(c'_{\text{rec}}\;(H_{\mathbf{Con}}\;(\mathsf{inC}\circ\mathsf{in}_1)\;y)) \\ &\equiv c_{Y,\text{rec}}\;(h_1\;(H_{\mathbf{Con}}\;(\mathsf{inC}\circ\mathsf{in}_1)\;y)) \\ &= c_{Y,\text{rec}}\;(h_1(\mathsf{in}_1(\mathsf{inC}\;y))) \\ &\equiv c_{Y,\text{rec}}\;(h\,n\;(\mathsf{inC}\;y)). \end{aligned}$$

For the second coherency we need to check that the paths $\mathsf{ap}\;h_2\;(\mathsf{ap}\;(f\;n)\;(p_i\;x))$ and $\mathsf{ap}\;h_2\;(p_i\;(f\;n\,x))$ are equal. Here we use several Lemmata, namely Lemma 2.2.2 from [25] and Lemma 16. For the first path we can show that it is equal to $q_{Y,i}\;(h\,n\,x)$.

$$\begin{aligned} \mathsf{ap}\;\;h_2\;(\mathsf{ap}\;(f\;n)\;(p_i\;x)) &\equiv \mathsf{ap}\;\;h_2\;(\mathsf{ap}\;(\mathsf{inC}\circ\mathsf{in}_1)\;(p_i\;x)) \\ &= \mathsf{ap}\;(h_2\circ\mathsf{inC})\;(\mathsf{ap}\;\;\mathsf{in}_1\;(p_i\;x)) \\ &= \mathsf{ap}\;\;h_1\;(\mathsf{ap}\;\;\mathsf{in}_1\;(p_i\;x)) \\ &= \mathsf{ap}\;(h_1\circ\mathsf{in}_1)\;\;(p_i\;x) \\ &= \mathsf{ap}\;(h\;n)\;\;(p_i\;x) \\ &\equiv q_{Y,i}\;(h\;n\;x) \end{aligned}$$

Since $(h_2\circ\mathsf{inC})\;x \equiv h_1\;x$ and $(h_1\circ\mathsf{in}_1)\;x \equiv h\,n\,x$, we can use the path refl, and thus the transport is the identity. For the other path we can do the same.

$$\begin{aligned} \mathsf{ap}\;\;h_2\;(p_i\;(f\;n\,x)) &\equiv q_{Y,i}\;(h_2\;(f\;n\,x)) \\ &\equiv q_{Y,i}\;(h_2\;(\mathsf{inC}\;\mathsf{in}_1\;x)) \\ &\equiv q_{Y,i}\;(h_1\;(\mathsf{in}_1\;x)) \\ &\equiv q_{Y,i}\;(h\;n\;x) \end{aligned}$$

To finish the proof, we need to give the image for the path com. More concretely, we need to show that $\mathsf{inc}\,n\,x$ and $\mathsf{inc}\;(n+1)\;(f\;x)$ are mapped to the same element. The map $f : F\,n \to F(n+1)$ is defined by the elimination rule of the coequalizer as follows

$$F\;n \xrightarrow{\mathsf{in}_1} H_{\mathbf{Con}}\;(F\;n) \xrightarrow{\mathsf{inC}} \ldots \xrightarrow{\mathsf{inC}} F(n+1)$$

Since the map to Y was defined via the universal property of the coequalizer, it will commute automatically. The commutativity is given by the computation rule of the coequalizer.

All in all, we have acquired a map $\prod(x : \mathsf{hocolim}\ F\ f), Y\ x$, and this way we defined the right eliminator for $\mathsf{hocolim}\ F\ f$. We shall call the eliminator $H\mathrm{ind}$.

4.3 Computation Rules

Lastly, we show that this eliminator also satisfies the computation rules. First, we prove that for each $t : A_{\mathrm{rec}}(\mathsf{hocolim}\ F\ f)$ that $H\mathrm{ind}(c_{\mathrm{rec}}\ t) \equiv f_i\ t\ (\overline{A_{\mathrm{rec}}}\ H\mathrm{ind}\ t)$. Again we use that colimits commute with polynomials.

Let $n : \mathbb{N}$ and $x : A_{\mathrm{rec}}(F\ n)$. Now we can perform the computations in an intermediate stage of the construction, and using the computation rules we get

$$\begin{aligned} H\mathrm{ind}(c_{\mathrm{rec}}\ (\mathsf{inc}\ n\ x)) &\equiv H\mathrm{ind}(\mathsf{inc}\ (n+1)\ (c'_{\mathrm{rec}} x)) \\ &\equiv c_{Y,\mathrm{rec}}\ (A_{\mathrm{rec}}\ (\mathsf{inc}\ n)\ x)\ (\overline{A}\ H\mathrm{ind}\ x) \end{aligned}$$

Hence, we can always take refl to be the path. This will also give an image for com, and thus the computation rules for the points are satisfied.

Note that the this computation rule is a propositional equality. This is logical, because it is proven all $x : A_{\mathrm{rec}}(F\ n)$. However, for terms built from c_{nonrec} and c_{rec}, we have a definitional equality. This is because $c_{\mathrm{nonrec}}\ a$ for $a : A_{\mathrm{nonrec}}$ is defined to be $\mathsf{inc}\ 0\ (c'_{\mathrm{nonrec}}\ a)$. All the closed terms are thus inhabitants of some $F\ n$, and since at every step the equalities are definitional, we can conclude that for closed terms the equality is definitional.

Now we show the computation rules for the paths, and in that case we have a parameter $a : B_i\ (\mathsf{hocolim}\ F\ f)$. By using that polynomials commute with homotopy colimits, we can again assume that we have $n : \mathbb{N}$ and $x : B_i(F\ n)$.

$$\begin{aligned} \mathrm{apD}\ \ H\mathrm{ind}\ (p_i\ (\mathsf{inc}\ n\ x)) &\equiv \mathrm{apD}\ \ H\mathrm{ind}\ (\mathrm{ap}\ (\mathsf{inc}\ (n+1))\ \mathsf{glueC}\ x) \\ &= \mathrm{apD}\ (H\mathrm{ind} \circ \mathsf{inc}(n+1))\ (\mathsf{glueC}\ x) \\ &= \mathrm{apD}\ (h\ n)\ (\mathsf{glueC}\ x) \\ &\equiv q_i\ (B_i\ H\mathrm{rec}\ \ x) \end{aligned}$$

5 Lemmata

▶ **Lemma 11.** *Suppose, we have a constructor term t such that $x : F\ T \vdash t : G\ T$ which uses at most n constructors, and that we have a map $c'_{\mathrm{nonrec}} : A_{\mathrm{nonrec}} \to P$. Then t induces a map $\overline{t} : F\ P \to H^n_{\mathbf{Con}}\ (G\ P)$ by replacing the constructors c_{nonrec} and c_{rec} by c'_{nonrec} and c'_{rec} respectively.*

Proof. We use induction on the form of the constructor term.

- $t = a$ with $a : B$ and B does not use T. Then we define $\overline{t}\ y = \mathsf{in}_1\ a$.
- $t = x$ with $x : F\ T$. Then we define $\overline{t}\ y = \mathsf{in}_1\ y$.
- $t = c_{\mathrm{nonrec}}\ a$ with $a : A_{\mathrm{nonrec}}$. Then we define $\overline{t}\ y = \mathsf{in}_1\ (c'_{\mathrm{nonrec}}\ a)$.
- $t = c_{\mathrm{rec}}\ r$ with $r : A_{\mathrm{rec}}\ T$ where r uses at most $n-1$ constructors. By induction we have a map $\overline{r} : F\ P \to A_{\mathrm{rec}}\ H^{n-1}_{\mathbf{Con}}$. Then we define $\overline{t}\ y = c'_{\mathrm{rec}}(\overline{r}\ x)$.
- For the rules for the projection, pairing and injection it is trivial. ◀

▶ **Lemma 12.** *Suppose, we have a constructor term t such that $x : F\ T \vdash t : G\ T$ which uses at most n constructors, and that we have a map $c'_{\mathrm{nonrec}} : A_{\mathrm{nonrec}} \to Q$. Furthermore, we assume that we have a map $f : Q \to P$ Then for all $x : F\ P$ the terms $\overline{t}(F\ f\ x)$ and $(H_{\mathbf{Con}} \circ G)\ f\ (\overline{t}\ x)$ are equal.*

Proof. Again we use induction on the constructor term.

- $t = a$ with $a : B$ and B does not use T. Then we have $G\,P = B$, and thus $\overline{t}(F\,f\,x) = a$. On the other hand, we have,
$$(H_{\mathbf{Con}} \circ G)\;f\;(\overline{t}\,x) = G\;f\;a = a.$$
- $t = x$ with $x : F\,T$. Then $G\,P = F\,P$, and thus $\overline{t}(F\,f\,x) = \mathsf{in}_1(F\,f\,x)$. Also,
$$(H_{\mathbf{Con}} \circ G)\;f\;(\overline{t}\,x) = (H_{\mathbf{Con}} \circ G)\;f\;(\mathsf{in}_1\;x) = \mathsf{in}_1(G\;f\;x) = \mathsf{in}_1(F\;f\;x).$$
- $t = c_{\mathrm{nonrec}}\,a$ with $a : A_{\mathrm{nonrec}}$. Then $G\,P = P$, and thus $\overline{t}(F\,f\,x) = \mathsf{in}_1(f(c'_{\mathrm{nonrec}}\,a))$. Also,
$$(H_{\mathbf{Con}} \circ G)\;f\;(\overline{t}\,x) = H_{\mathbf{Con}}\;f\;(\mathsf{in}_1(c'_{\mathrm{nonrec}}\,a)) = \mathsf{in}_1(f(c'_{\mathrm{nonrec}}\,a))$$
- $t = c_{\mathrm{rec}}\,r$ with $r : A_{\mathrm{rec}}\,T$ where r uses at most $n-1$ constructors. The induction hypothesis in this case is that $H_{\mathbf{Con}}\;f\;(\overline{r}\,x) = \overline{r}(F\,f\,x)$. Then again $G\,P = P$, and thus $\overline{t}(F\,f\,x) = c'_{\mathrm{rec}}\,(\overline{r}(F\,f\,x))$. Furthermore,
$$(H_{\mathbf{Con}} \circ G)\;f\;(\overline{t}\,x) = H_{\mathbf{Con}}\;f\;(c'_{\mathrm{rec}}\,(\overline{r}\,x)) = c'_{\mathrm{rec}}\,(H_{\mathbf{Con}}\;f\;(\overline{r}\,x)) = c'_{\mathrm{rec}}\,(\overline{r}(F\;f\;x))$$
- For the rules for the projection, pairing and injection it is trivial. ◀

▶ **Lemma 13.** *The type* $\mathsf{hocolim}\;(\lambda n.A)\;(\lambda n.\,\mathrm{Id})$ *is isomorphic to* A.

Proof. We define the map $f : A \to \mathsf{hocolim}\;(\lambda n.A)\;(\lambda n.\,\mathrm{Id})$ to be $\mathsf{inc}\;0$. Next we define $g : \mathsf{hocolim}\;(\lambda n.A)\;(\lambda n.\,\mathrm{Id}) \to A$ by $\mathsf{hocolim}$ recursion. We send $\mathsf{inc}\,n\,a$ to a, and then the required diagrams commute by reflexivity. The proof that these maps are mutual inverses is straightforward but tedious, so we refer the reader to the accompanying Coq code [10]. ◀

▶ **Lemma 14.** *Colimits commute with coproducts, so* $\mathsf{hocolim}\;(G_1 + G_2)\;(g_1 + g_2)$ *and* $\mathsf{hocolim}\;G_1\,g_1 + \mathsf{hocolim}\;G_2\,g_2$ *are isomorphic.*

Proof. This one is not difficult either, so we will be brief and refer the reader to [10] for details. We start by making a map
$$\mathsf{hocolim}\;(G_1 + G_2)\;(g_1 + g_2) \to \mathsf{hocolim}\;G_1\,g_1 + \mathsf{hocolim}\;G_2\,g_2.$$
This map is defined by recursion over the colimit, so take any $n : \mathbb{N}$ and $x : G_i\,n$. Then the image of $\mathsf{inc}\,n\;(\mathsf{in}_i\;x)$ is defined to be $\mathsf{in}_i(\mathsf{inc}\,n\,x)$, and the image of $\mathsf{com}\,n\;(\mathsf{in}_i\;x)$ is defined by $\mathrm{ap}\;\mathsf{in}_i\;(\mathsf{com}\;G_i\;g_i\;n\;x)$.

For the map in the other direction, we need to make
$$\mathsf{hocolim}\;G_i\,g_i \to \mathsf{hocolim}\;(G_1 + G_2)\;(g_1 + g_2).$$
We send $\mathsf{inc}\,n\,x$ to $\mathsf{inc}\,n\;(\mathsf{in}_i\;x)$ and $\mathsf{com}\,n\,x$ to $\mathsf{com}\,n\;(\mathsf{in}_i\;x)$. ◀

▶ **Lemma 15.** *Colimits commute with products.*

Proof. This follows from a more general result which says that colimits commute with sigma types which is proven in [12, 13] under 'commutation with sigmas'. ◀

▶ **Lemma 16.** *Given are types* A, B *and functions* $f, g : A \to B$ *such that we have an inhabitant* $e : f = g$. *Furthermore, assume that we have a path* $p : x = y$ *where* x *and* y *are terms of type* A. *Then for all we have a path*
$$e_*\;(\mathrm{ap}\;\;f\;p) = \mathrm{ap}\;\;g\;p.$$

All in all, we have an inhabitant of the type

$$\prod(f, g : A \to B)(e : f = g)(x, y : A)(p : x = y), e_* \,(\mathrm{ap}\ f\ p) = \mathrm{ap}\ g\, p$$

where A, B : Type.

Proof. Since this statement is universally quantified over p, we can apply path induction. Assuming p to be refl, the map e_* is the identity, and in that case it holds. Hence, it follows by path induction. ◀

▶ **Lemma 17.** *Again we are given types A, B, functions $f, g : A \to B$, an inhabitant $e : f = g$, and a path $p : f\ x = f\ y$ with $x, y : A$. Then we have a path*

$$e_*\, p = (e \,\Box\, x)^{-1} \bullet p \bullet (e \,\Box\, x)$$

Formally, this means that for A, B : Type we have an inhabitant of the type

$$\prod(f, g : A \to B)(e : f = g)(x, y : A)(p : f\ x = f\ y), e_*\, p = (e \,\Box\, x)^{-1} \bullet p \bullet (e \,\Box\, y).$$

Proof. Again we use path induction on e, so we assume that $e = \mathsf{refl}$. Then $e \,\Box\, x = \mathsf{refl}$ and $e \,\Box\, y = \mathsf{refl}$, so the right side is equal to p. Also, in this case e_* is the identity map, so the left side also is p. ◀

6 Conclusion and Further Work

MOVED FROM BEGINNING: A possible extension would be to use arbitrary containers as in [1], but we shall refrain to do so. With that extension the given proof requires the axiom of choice which generally does not hold in type theory.

Higher inductive types can thus be constructed if we have the interval, homotopy pushout and homotopy colimits. Since the construction is done in homotopy type theory without assuming extra axioms, the proof is constructive. Notice that the only inductive types we needed were the sum and product types rather than the general class of W-types or those described inductive schemes. Hence, we can also deduce the existence of all inductive types from this theorem if we just have sums, products, and the three HITs.

There is still some work which remains to be done. First of all, we expect that this theorem will have applications in the semantics and the metatheory of higher inductive types. There are already several interpretations of homotopy type theory in abstract homotopy theory [5, 7, 11, 14, 18]. Rather than using the full-blown definition one could instead just look at a small number of cases, and that could simplify checking whether all higher inductive types exist in some model. Also, the theorem could also be used for generic programming with higher inductive types [4]. If we know how a function is defined for the interval, the homotopy pushout, and the homotopy colimit, then we know how it is defined for all higher inductive types.

Furthermore, there might be some interesting theoretical possibilities for the Three-HITs Theorem. We expect that it can be generalized if a syntax of higher inductive types with a higher dimensional paths is given. If constructor terms are used, then most of the proof might even be reusable.

The work of Sojakova [24] relates higher inductive types to homotopy initial algebra. Since initial algebra are made as a colimit, it could be that this construction and homotopy initial algebras are related in some way. Also, the work by Altenkirch et al. [3] allows more general constructors which is not allowed in the syntax used in this paper. If the proof can be extended somehow to also allows these constructors, then that would pave the way towards constructive semantics of QIITs.

References

1 Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Containers: Constructing Strictly Positive Types. *Theoretical Computer Science*, 342(1):3–27, 2005.

2 Jiří Adámek. Free Algebras and Automata Realizations in the Language of Categories. *Commentationes Mathematicae Universitatis Carolinae*, 15(4):589–602, 1974.

3 Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, and Fredrik Nordvall Forsberg. Quotient Inductive-Inductive Types. *arXiv preprint arXiv:1612.02346*, 2016.

4 Thorsten Altenkirch and Conor McBride. Generic Programming within Dependently Typed Programming. In *Generic Programming*, pages 1–20. Springer, 2003.

5 Peter Arndt and Krzysztof Kapulkin. Homotopy-Theoretic Models of Type Theory. In *International Conference on Typed Lambda Calculi and Applications*, pages 45–60. Springer, 2011.

6 Steve Awodey, Nicola Gambino, and Kristina Sojakova. Inductive Types in Homotopy Type Theory. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, pages 95–104. IEEE Computer Society, 2012.

7 Steve Awodey and Michael A Warren. Homotopy Theoretic Models of Identity Types. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 146, pages 45–55. Cambridge Univ Press, 2009.

8 Henning Basold, Herman Geuvers, and Niels van der Weide. Higher Inductive Types in Programming. *Journal of Universal Computer Science*, 23(1):63–88, jan 2017. `http://www.jucs.org/jucs_23_1/higher_inductive_types_in`.

9 Andrej Bauer, Jason Gross, Peter LeFanu Lumsdaine, Michael Shulman, Matthieu Sozeau, and Bas Spitters. The HoTT Library: A Formalization of Homotopy Type Theory in Coq. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, pages 164–172, 2017.

10 Andrej Bauer and Niels van der Weide. Accompanying Lemmas of Three-HITs Theorem in Coq. `https://github.com/nmvdw/Three-HITs`, 2016.

11 Marc Bezem, Thierry Coquand, and Simon Huber. A Model of Type Theory in Cubical Sets. In *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26, pages 107–128, 2014.

12 Simon Boulier. Colimites et Structure de Modele en Théorie des Types Homotopique. `http://perso.eleves.ens-rennes.fr/~sboul434/documents/rapport_de_stage_M2_Nantes.pdf`, 2015.

13 Simon Boulier. Colimits in HoTT. `https://homotopytypetheory.org/2016/01/08/colimits-in-hott/`, 2016.

14 Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. *arXiv preprint arXiv:1611.02108*, 2016.

15 Floris van Doorn. Constructing the Propositional Truncation using Non-Recursive HITs. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, pages 122–129. ACM, 2016.

16 Peter Dybjer. Inductive Families. *Formal aspects of computing*, 6(4):440–465, 1994.

17 Peter Dybjer and Anton Setzer. Induction–Recursion and Initial Algebras. *Annals of Pure and Applied Logic*, 124(1-3):1–47, 2003.

18 Chris Kapulkin and Peter LeFanu Lumsdaine. The Simplicial Model of Univalent Foundations (after Voevodsky). *arXiv preprint arXiv:1211.2851*, 2012.

19 Nicolai Kraus. The General Universal Property of the Propositional Truncation. *arXiv preprint arXiv:1411.2682*, 2014.

20 Nicolai Kraus. Constructions with Non-Recursive Higher Inductive Types. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 595–604. ACM, 2016.

21 Dan Licata. Another Proof That Univalence Implies Function Extensionality. `https://homotopytypetheory.org/2014/02/17/another-proof-that-univalence-implies-function-extensionality/`, 2014.

22 Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5. Springer Science & Business Media, 2013.

23 Mike Shulman. An Interval Type Implies Function Extensionality. `https://homotopytypetheory.org/2011/04/04/an-interval-type-implies-function-extensionality/`, 2011.

24 Kristina Sojakova. Higher Inductive Types as Homotopy-Initial Algebras. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 31–42, 2015.

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