The Three-HITs Theorem*

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

1 Department of Mathematics and Physics, University of Ljubljana, Ljubljana, Slovenia
Andrej.Bauer@andrej.com

2 Department of Computer Science, Radboud University, Nijmegen, The Netherlands
nweide@cs.ru.nl

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent convallis orci arcu, eu mollis dolor. Aliquam eleifend suscipit lacinia. Maecenas quam mi, porta ut lacinia sed, convallis ac dui. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse potenti.

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) extend normal inductive types by allowing constructors for both points and paths, rather than just for the points. While inductive types are freely generated from a signature, a higher inductive type is freely generated from a signature with equations. Numerous examples and definitions of such types have been given in the literature [3, 4, 5, 15, 16], but a definition with a good metatheory remains to be given. As a step towards that goal, we simplify the definition given in [5] 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'. One starts with the nonrecursive constructors, and then at each step new terms are added by applying the recursive constructors to the previously built terms. This is explained by a theorem which says that inductive types are initial algebras for a functor [10, 11] and another one about how these algebras are obtained [2]. For higher inductive types one would expect a similar result which explains philosophically what they are. The only difference between higher inductive types and normal inductive types is that during the construction of higher inductive types several identifications are made which are described by the path constructors.

The goal of this paper is to show that the higher inductive types defined in [5] can be generated from the interval, homotopy pushouts and colimits, and that formally justifies this intuition on higher inductive types. More concretely, we will prove the following theorem.

▶ **Theorem 1** (Three-HITs Theorem). *In Martin-Löf Type Theory extended with an interval object, homotopy pushouts and colimits, we can define for each higher inductive type as in [5] a tye with the same introduction, elimination and computation rules.*

* This work was partially supported by COST Action CA15123.

Also, this generalizes the results in [9, 12, 13] 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 types.

In Section 2 we shall discuss some of the required notions for this paper. More concretely, we recall the syntax of higher inductive types, and as examples we give the interval, the homotopy pushout, and colimit. 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.

2 Preliminaries

Let us briefly recall the scheme of higher inductive types which we shall use [5]. First of all, we need *polynomial functors*.

▶ **Definition 2.** Let X be a variable. Then a *polynomial* is given by the following grammar.

$$F, G ::= B : \text{Type} \mid X \mid F \times G \mid F + G$$

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. One can prove that polynomials are functors, and that given a type family $Y : T \to \text{Type}$ and a polynomial F, we get a lift $\bar{F} : F\,T \to \text{Type}$. We will write $F\,f$ to denote the application of a polynomial F to a map f. Also, given $f : \prod(x : T), Y\,x$, one can define $\bar{F}\,f : \prod(x : H\,T), \bar{H}\,Y\,x$. Precise definitions of these are given in [5].

Next we define the notion of a constructor term.

▶ **Definition 3.** Given are functions $c : A\,T \to T$. Then 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.

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

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

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

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 → H
| p_i : ∏(x : B_i H), t_i = r_i     (i = 1, ..., 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 are a constructor $c : A\,H \to H$, a type family $Y : H \to \textsc{Type}$, and a term

$$\vdash f : \prod(x : A\,H), \bar{A}\,Y\,x \to Y(c\,x).$$

For a constructor term $F\,H \Vdash r : G\,H$ 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.

$$\widehat{t} := t \qquad \widehat{x} := h_x \qquad \widehat{c_i\,r} := f_i\,r\,\widehat{r}$$
$$\widehat{\pi_j\,r} := \pi_j\,\widehat{r} \qquad \widehat{(r_1, r_2)} := (\widehat{r_1}, \widehat{r_2}) \qquad \widehat{\mathrm{in}_j\,r} := \widehat{r}$$

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 $t : A\,H$

$$H\mathrm{rec}\,(c_i\,t) \equiv f_i\,t\,(\bar{A}\,H\mathrm{rec}\;t),$$

and for each $a : B_i\,H$

$$\mathrm{apd}\;H\mathrm{rec}\,(p_j\;a) \equiv q_j\;a\,(\bar{B}_i\,H\mathrm{rec}\;a).$$

Note that these equalities are definitional rather than propositional.

Let us finish by defining some examples of higher inductive types which will be used a lot in this paper. The first one would be the interval

```
Inductive $I^1$ :=
| $0 : I^1$
| $1 : I^1$
| $\mathbf{seg} : 0 = 1$
```

Note that for every type A with inhabitants x and y we have a path $x = y$ iff we have a map I^1 to A sending 0 and 1 to x and y respectively. Let $\mathbf{1}$ be the unit type with point $*$. Then we can define maps $\delta_0, \delta_1 : \mathbf{1} \to$ sending $*$ to 0 and 1 respectively. Also, we define $\delta : \mathbf{1} + \mathbf{1} \to I^1$ by $\delta_0 + \delta_1$.

Next we define the homotopy pushout.

```
Inductive $\mathbf{hpushout}\;(A, B, C : \textsc{Type})\;(f : A \to B)\;(g : A \to C)$ :=
| $\mathbf{inl} : B \to \mathbf{hpushout}\;A\;B\;C\;f\;g$
| $\mathbf{inr} : C \to \mathbf{hpushout}\;A\;B\;C\;f\;g$
| $\mathbf{glue} : \prod(a : A), \mathbf{inl}(f\;a) = \mathbf{inr}(g\;a)$
```

Note the similarities with the construction of the pushout. Also, we can define the homotopy colimit as a higher inductive type in much the same way.

```
Inductive hocolim $(F : \mathbb{N} \to \text{Type})\ (f : \prod(n : \mathbb{N}, F\,n \to F(n+1)))$ :=
| inc : $\prod(n : \mathbb{N}), F\,n \to$ hocolim $F\ f$
| com : $\prod(n : \mathbb{N})(x : F\,n)$, inc $n\,x =$ inc $(n+1)\ (f\,n\,x)$
```

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 think about how it should be given. By Adámek's theorem, every inductive type can be given as a colimit. An inductive type T is given by a polynomial functor F and a constructor $c : F\,T \to T$, and then T is the colimit of the sequence

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

To understand what this does, let us assume that $F\,X = 1 + X$, 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 $\mathbb{N}$.

For higher inductive types one would like to do a similar construction. The first difference is that instead of starting with nothing and adding each constructor at every step, we start with the nonrecursive constructors, and add recursive constructors at every step. Also, 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 right paths.

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

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

The first approximation just has a constructor 0, and after that we add a constructor for $S\,0$ to obtain the second approximation. In the third approximation, which we call $F\,3$, we found inhabitants 0, $S\,0$, and $S(S\,0)$, and now we can make the first identification. To do so, we take the following homotopy pushout

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

to obtain the actual third approximation $P\,3$, 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, one needs to refer back to a previous approximation.

Note that one always has to go back a fixed number of steps due to the usage of constructor terms. By extending the syntax, one can also think of examples where one needs to go back an arbitrary amount of steps. For that, we consider the following example.

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

This example might seem like it is allowed in the syntax, but it is not. The term S^n is defined as a polymorphic function which needs the type as argument, and that is not allowed with constructor terms. In this extension the construction will be more complicated, and the construction is not predicative, so this will not be considered in this paper.

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

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

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

If n is clear from the context, then we shall not write it down.

▶ **Lemma 6.** *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 $c'_{\text{nonrec}} : A_{\text{nonrec}} \to P$. Then t induces a map $\bar{t} : F\,P \to G\,H^n_{\mathbf{Con}}$ 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 $\bar{t}\,y = a$.
- $t = x$ with $x : F\,T$. Then we define $\bar{t}\,y = \text{in}_P y$.
- $t = c_{\text{nonrec}} a$ with $a : A_{\text{nonrec}}$. Then we define $\bar{t}\,y = c'_{\text{nonrec}} a$.
- $t = c_{\text{rec}} r$ with $r : A_{\text{rec}}\ T$ where r uses at most $n - 1$ constructors. By induction we have a map $\bar{r} : F\,P \to A_{\text{rec}}\ H^{n-1}_{\mathbf{Con}}$. Then we define $\bar{t}\,y = c'_{\text{rec}}(\bar{r}\,x)$.

- For the rules for the projection, pairing and injection it is trivial. ◀

Now we have enough to define the approximator. During the construction one also needs to pay attention to coherency. For example, in P we have c_{nonrec}, so in $H_{\mathbf{Approx}}\ P$ we might have duplicates of some terms. By tweaking the pushout a little bit, one can easily solve these coherency problems.

▶ **Definition 7.** Let a higher inductive type H be given as before, and let the type $H_{\mathbf{Con}}$ be defined as before. Then we define the *approximator* $H_{\mathbf{Approx}}$, which has a parameters P, Q and $f : Q \to P$, of H the following pushout

$$\begin{tikzcd}
((\mathbf{1}+\mathbf{1}) \times B_i\ P) + H_{\mathbf{Con}}\ Q \arrow[r, "\delta+\mathbf{inl}"] \arrow[d, "\overline{t_i}+\overline{r_i}+H_{\mathbf{Con}}\ f"'] & (I^1 \times B_i\ P) + H_{\mathbf{Approx}} Q \arrow[d, "\mathbf{inr}"] \\
H_{\mathbf{Con}}\ P \arrow[r, "\mathbf{inl}"'] & H_{\mathbf{Approx}}\ P
\end{tikzcd}$$

Note that this can be written as the homotopy pushout.

▶ **Definition 8.** 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

$$\begin{aligned}
F\ 0 &= H_{\text{nonrec}}, \\
F\ 1 &= H_{\mathbf{Approx}}\ (F\ 0)\ \mathbf{0}\ (\mathbf{0}\text{rec}(F\ 0)) \\
F(n+2) &= H_{\mathbf{Approx}}\ (F(n+1))\ (F\ n)\ (f\ n)
\end{aligned}$$

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

$$\begin{tikzcd}
P \arrow[r] & H_{\mathbf{Con}}\ P \arrow[r] & H_{\mathbf{Approx}} P.
\end{tikzcd}$$

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 **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 $A_{\text{nonrec}} \to$ **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.

$$\begin{tikzcd}
A_{\text{nonrec}} \arrow[r, "c'_{\text{nonrec}}"] & F\ 0 \arrow[r, "\mathbf{inc}\ 0"] & \mathbf{hocolim}\ F\ f
\end{tikzcd}$$

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

▶ **Lemma 9.** *The types $A\,(\textbf{hocolim}\ F\ f)$ and $\textbf{hocolim}\,(A \circ F)\,(A\ f)$ are isomorphic for all polynomials A.*

Proof. ◀

Now we will construct the map $A_{\text{rec}}\ (\textbf{hocolim}\ F\ f) \to \textbf{hocolim}\ F\ f$, and by Lemma 9 it suffices to make a map $\textbf{hocolim}\ (A_{\text{rec}} \circ F)\ (A\ f) \to \textbf{hocolim}\ F\ f$. For this we use the recursion rule of **hocolim**, and we start with the following string of maps

$$A_{\text{rec}}(F\ n) \xrightarrow{c'_{\text{rec}}} H_{\text{rec}}(F\ n) \xrightarrow{\textbf{term} \circ \iota_2} H_{\textbf{Con}}(F\ n) \xrightarrow{\textbf{inl}} H_{\textbf{Approx}}(F\ n) = F(n+1)$$

where the map $H_{\text{rec}}(F\ n) \to H_{\textbf{Con}}(F\ n)$ is the inclusion and the map $H_{\textbf{Con}}(F\ n) \to H_{\textbf{Approx}}(F\ n)$ is the pushout map. Composing this map with **inc**, gives maps $A_{\text{rec}}(F\ n) \to \textbf{hocolim}\ F\ f$ for all $n : \mathbb{N}$.

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

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

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

$$\begin{tikzcd}
A_{\text{rec}}(F\ n) \arrow[r, "c'_{\text{rec}}"] \arrow[d, "A_{\text{rec}}(f\ n)"'] & H_{\text{rec}}(F\ n) \arrow[r, "\textbf{term} \circ \iota_2"] \arrow[d, "H_{\text{rec}}(f\ n)"'] & H_{\textbf{Con}}(F\ n) \arrow[d, "H_{\textbf{Con}}(f\ n)"'] \\
A_{\text{rec}}(F(n+1)) \arrow[r, "c'_{\text{rec}}"'] & H_{\text{rec}}(F(n+1)) \arrow[r, "\textbf{term} \circ \iota_2"'] & H_{\textbf{Con}}(F(n+1))
\end{tikzcd}$$

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 triangle commutes as well. Hence, it suffices to show that the following diagram commutes.

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

Note that $H_{\textbf{Approx}}(F(n+1)) = F(n+2)$ and $H_{\textbf{Approx}}(F\,n) = F(n+1)$. Hence, this diagram commutes, because of the pushout.

Next we need to define the introduction rules for the paths. Let us start by making a map $\sum_{i=1}^{m} I^1 \times B_i\ (\textbf{hocolim}\ F\ f) \to \textbf{hocolim}\ F\ f$. Since homotopy colimits commute with polynomials, it suffices to make a map $\textbf{hocolim}\ (I^1 \times B_i\ F)\ (I^1\ B_i\ f) \to \textbf{hocolim}\ F\ f$.

For this we use the eliminator of the homotopy colimit. First, we need to make for all n a map $I^1 \times B_i\,(F\,n) \to \mathbf{hocolim}\ F\,f$. By definition we have the following string of maps

$$\begin{tikzcd} I^1 \times B_i\,(F\,n) \arrow[r, "\mathbf{inr}"] & F(n+1) \arrow[r, "\mathbf{inc}(n+1)"] & \mathbf{hocolim}\ F\,f. \end{tikzcd}$$

Next we need to show that the following diagram commutes.

$$\begin{tikzcd} I^1 \times B_i\,(F\,n) \arrow[r, "\mathbf{inr}"] \arrow[d, "I^1 \times B_i\,(f\,n)"'] & F(n+1) \arrow[d, "f(n+1)"] \\ I^1 \times B_i\,(F(n+1)) \arrow[r, "\mathbf{inr}"] & F(n+2) \end{tikzcd}$$

This can be proven by induction on B_i, and using the definition of $I^1 \times B_i\,(f\,n)$. We can use the path **refl**. All in all, we get a map $p_i : \mathbf{hocolim}\,(I^1 \times B_i\,F)\,(I^1 \times B_i\,f) \to \mathbf{hocolim}\ F\,f$

Now we have the following diagram

$$\begin{tikzcd} 2 \times B_i(\mathbf{hocolim}\ F\,f) \arrow[r] \arrow[dr, "\overline{t_i}+\overline{r_i}"'] & I^1 \times B_i(\mathbf{hocolim}\ F\,f) \arrow[r] & \mathbf{hocolim}\,(I^1 \times B_i\,F)\,(I^1 \times B_i\,f) \arrow[dl] \\ & \mathbf{hocolim}\ F\,f & \end{tikzcd}$$

$2 \times B_i(\mathbf{hocolim}\ F\,f)$ is isomorphic to $\mathbf{hocolim}\,(2 \times B_i\,F)\,2(\times B_i\,f)$.

Induction to second, so we have $(0, x)$ (similar for 1) and $x : F\,n$. This gets mapped to $(0, x)$ which is equal to t_i via the path $\mathbf{glue}(\mathbf{inl}(0, x))$.

Remaining:

$$\mathbf{com}_*(\mathrm{ap}\,(\mathbf{inc}\ n)\ \mathbf{glue}(\mathbf{inl}(0, x))) = \mathrm{ap}\,(\mathbf{inc}(n+1)))\ \mathbf{glue}(\mathbf{inl}(0, f\,n\,x))$$
$$(\mathrm{ap}\ \overline{t_i}\ \mathbf{com})^{-1} \bullet \mathrm{ap}\,(\mathbf{inc}\ n)\ \mathbf{glue}(\mathbf{inl}(0, x)) \bullet \mathrm{ap}\ p_i\ \mathbf{com}$$

$(\mathrm{ap}\ \overline{t_i}\ \mathbf{com})^{-1}$ is like $\mathrm{ap}\,(\mathbf{inc}\ n)\,(\mathrm{ap}\ c \ldots (\ldots \mathrm{ap}$ coherence path$))$, because of the computation rule of c_i.

This is the coherence path which should be added on level n.

4.2 Elimination Rule

For the next step we define the right eliminator for $\mathbf{hocolim}\ F\,f$. For this we suppose that we have

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

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

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

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

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

First we need to make a map $\prod(x : F\,0), Y(\mathbf{inc}\ 0\,x)$. 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 $c_{\mathrm{nonrec}}\,a = \mathbf{inc}\ 0\,(c\_nonreca)$, 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 $\prod(x : F\,0), Y(\mathbf{inc}\ 0\,x)$.

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

Let us do this in the general case. Suppose that we have $g : P \to \mathbf{hocolim}\, F\, f$, and that we have $h : \prod(x : P), Y(gx)$. Our goal is to extended the map into a $h' : \prod(x : H_{\text{rec}}P), Y(c_{Y,\text{rec}}x)$, and for that we use H_{rec} induction. Note that for each $c'_{\text{rec}}\ x$ with $x : A_{\text{rec}}\ P$ we have the type $Y(c_{\text{rec}}\ (A_{\text{rec}}\ g\ x))$, and thus we have a type family on H_{rec}. Now let $x : A_{\text{rec}}\ P$ and $y : \overline{A_{\text{rec}}}\ Y\ x$ be given. Then we need to give an element of the type $Y(c_{\text{rec}}\ (A_{\text{rec}}\ g\ x))$ for which we take

$$c_{Y,\text{rec}}\ (A_{\text{rec}}\ g\ x)\ y$$

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 h, and on the other components we define it via extension.

Next we need to extend the map to $H_{\mathbf{Approx}}\ P\ Q$. For that we need to make a map $h' : \prod(x : I^1 \times B_i\ P), Y()$

Let $(t, x) : I^1 \times B_i\ P$, and now we use I^1 induction. If $t = 0$, we map it to $\widehat{t_i}(x)$, and for $t = 1$ we map it to $\widehat{r_i}(x)$. Now the path $q_{Y,i}$ gives the image of $\mathbf{seg}$, and that finishes the induction.

For the commutativity of the diagram note that by definition the dependent map from $H_{\mathbf{Con}}\ P$ to Y sends the constructor term t_i to the lift $\widehat{t_i}$. This is because the map replaces each c_{rec} by $c_{Y,\text{rec}}$ which is precisely how the lift was constructed.

Next we do it for the coherency.

Induction on the level (so, ultimately some kind of simultaneous induction?)

Then induction on the depth.

Base case: just induction assumption.

Step case: then it can be worked out from the diagram.

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

$$\begin{tikzcd}
((\mathbf{1}+\mathbf{1}) \times B_i\ (F(n+1))) + H_{\mathbf{Con}}\ (F\ n) \arrow[r, "\delta+\mathbf{inl}"] \arrow[d, "\overline{t_i}+\overline{r_i}+H_{\mathbf{Con}}\ f"'] & (I^1 \times B_i\ (F(n+1))) + F(n+1) \arrow[d, "\mathbf{inr}"] \\
H_{\mathbf{Con}}\ (F(n+1)) \arrow[r, "\mathbf{inl}"'] & F(n+2)
\end{tikzcd}$$

Since the map to Y was defined as the pushout map, it will commute automatically.

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

4.3 Computation Rules

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

Let $n : \mathbb{N}$ and $x : A_{\text{rec}}(F\ n)$. Then by the computation rules we have

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

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 closed terms, we have a definitional equality. This is because $c_{\mathrm{nonrec}}\ a$ for $a : A_{\mathrm{nonrec}}$ is defined to be**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.

5 Lemmata

▶ **Lemma 10.** ***hocolim*** $(\lambda n.A)$ Id *is* A*.*

▶ **Lemma 11.** *Colimits commute with coproducts.*

▶ **Lemma 12.** *Colimits commute with products.*

6 Conclusion and Further Work

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 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.

5 Henning Basold, Herman Geuvers, and Niels van der Weide. Higher inductive types in programming.

6 Andrej Bauer, Jason Gross, Peter LeFanu Lumsdaine, Mike Shulman, Matthieu Sozeau, and Bas Spitters. The hott library: A formalization of homotopy type theory in coq. *arXiv preprint arXiv:1610.04591*, 2016.

7 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.

8 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.

9 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.

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

11 Peter Dybjer and Anton Setzer. Induction–recursion and initial algebras. *Annals of Pure and Applied Logic*, 124(1-3):1–47, 2003.

12 Nicolai Kraus. The general universal property of the propositional truncation. *arXiv preprint arXiv:1411.2682*, 2014.

13 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.

14 Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 2013.

15 Kristina Sojakova. Higher inductive types as homotopy-initial algebras. In *ACM SIGPLAN Notices*, volume 50, pages 31–42. ACM, 2015.

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