The Three-HIT 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 Dummy classification – please refer to `http://www.acm.org/about/class/ccs98-html`

Keywords and phrases Dummy keyword – please provide 1–5 keywords

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Higher inductive types extend normal inductive types by allowing constructors for equations as well rather than just for the points. Numerous examples and definitions of higher inductive 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 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 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, 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 for three types.

In Section 2 we shall discuss some of the required notions for this paper. More concretely, we recall the definition of a higher inductive, and show the interval, the pushout, and the homotopy colimit. Next we define the approximating sequence of a higher inductive type in

* This work was partially supported by someone.

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 1.** Let X be a variable. Then a *polynomial* is given by the following grammar.

$$F, G ::= A : \textsc{Type} \mid X \mid F \times G \mid F + G$$

A possible extension would be to use arbitrary containers as in [1], but we shall refrain to do so When we tried to do the proof with that extension, we needed the axiom of choice which does not hold in general in type theory. One can prove that polynomials are functors, and that given a type family $Y : T \to \textsc{Type}$ and a polynomial F, we get a lift $\bar{F} : F\,T \to \textsc{Type}$. We will write $F\,f$ to denote the application of a polynomial F to a map f.

Next we define the notion of a constructor term.

▶ **Definition 2.** Given are functions $c_i : H_i\,T \to T$ for $i = 1, \ldots, k$. Then we say t is a *constructor term* over the c_i if we can find polynomials F and G such that $x : F\,T \Vdash t : G\,T$ can be derived using the following rules.

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

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

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

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

▶ **Definition 3.** 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 4.** Given is 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.

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

The introduction rules for H 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.

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

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 $t : A\, H$

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

and for each $a : B_i\, H$

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

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

```
Inductive I¹ :=
| 0 : I¹
| 1 : I¹
| 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 hpushout (A, B, C : TYPE) (f : A → B) (g : A → C) :=
| inl : B → hpushout A B C f g
| inr : C → hpushout A B C f g
| glue : ∏(a : A), inl(f a) = 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 : 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)
```

3 The Approximator

Let us assume that we some higher inductive type H is given. In order to construct H as a colimit, we first need to give how the approximations in the colimit are given, 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

$$\mathbf{0} \longrightarrow F\,\mathbf{0} \longrightarrow F(F\,\mathbf{0}) \longrightarrow \dots$$

To understand what this does, let us assume that $F\,X = X + 1$, 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. However, since extra equalities might be present in the higher inductive type, several tweaks need to be made to the construction. Rather than just adding points at every step, identifications need to be made as well.

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

```
Inductive ℕ₂ :=
| 0 : ℕ₂
| S : ℕ₂ → ℕ₂
| p : S(S 0) = 0
```

The first approximation is the empty type, and after that we add a constructor for 0 to obtain the first approximation. After the third step we found inhabitants 0, $S\,0$, and $S(S\,0)$ in the approximation $F(3)$, and now we can make the first identification. At this step we can do a homotopy pushout

$$\begin{array}{ccc} \mathbf{1} + \mathbf{1} & \xrightarrow{\delta} & I^1 \\ {\scriptstyle 0+S(S\,0)}\downarrow & & \downarrow \\ F(3) & \longrightarrow & F'(3) \end{array}$$

to obtain the fixed third approximation $F'(3)$, and we continue our construction with that. To glue in $F\,n$, we need to use the elements from $F(n-2)$. So, the approximator F the parameter should be one of the previous approximations of which the identifications can be taken.

Note that one always have 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 ℕ₂ :=
| 0 : ℕ₂
| S : ℕ₂ → ℕ₂
| p : ∏(n : ℕ), Sⁿ 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 more precise, and for that we start with a higher inductive type which is given as follows.

```
Inductive H :=
| c_nonrec : A_nonrec → H
| c_rec : A_rec H → H
| p_i : ∏(x : B_i H), t_i = r_i    (i = 1, ..., m)
```

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

```
Inductive H_nonrec :=
| c'_nonrec : A_nonrec → H_nonrec
```

Next we need to generate the further approximations, and that will be done in two steps. First, we note that using the recursive constructors we can extend a type by using its inhabitants as the arguments.

```
Inductive H_rec (P : TYPE):=
| c'_rec : A_rec P → H_rec P
```

To do the identifications, we need to be able to interpret the constructor terms. For that we realize that each constructor terms uses each constructor only a finite amount of times, and thus there is a maximum number n of times a constructor is used. From this we can define the *approximator*. With this we can define

```
Inductive H_Con (P : TYPE):=
| term : P + H_rec P + ... + H^n_rec P → H_Con P
```

▶ **Lemma 5.** *A constructor term t_i using a variable $x : B_i\, P$ with depth at most n induces a map $B_i\, P \to H_{\mathbf{Con}}$.*

▶ **Definition 6.** Let a higher inductive type H be given as before, and let the types $H_{\mathbf{Con}}$ be defined as before. Then we define the *approximator* $H_{\mathbf{Approx}}$, which has a parameter P, of H the following pushout

$$\begin{array}{ccc} \sum_{i=1}^{m}((\mathbf{1}+\mathbf{1})\times B_i\, P) & \xrightarrow{\delta} & \sum_{i=1}^{m}(I^1\times B_i\, P) \\ {\scriptstyle t_i+r_i}\downarrow & & \downarrow \\ H_{\mathbf{Con}}\, P & \xrightarrow[q_1]{} & H_{\mathbf{Approx}}\, P \end{array}$$

Note that this can be written as the homotopy pushout.

▶ **Definition 7.** In the setting as described, we can define a sequence of approximations to H as follows

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

For the colimit we also need maps $f\,n : F\,n \to F(n+1)$. Note that we always have the following sequence of maps

$$P \longrightarrow H_{\mathbf{Con}}\, P \longrightarrow H_{\mathbf{Approx}} P.$$

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 a candidate for the higher inductive type, namely **hocolim** $F\ f$. In this section we shall prove that we can define the introduction rules, the eliminator, and show the computation rules. We will do this step by step, and refer to lemmata in Section 5.

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 $A_{\text{nonrec}} \to \mathbf{hocolim}\ F\ f$ which gives the introduction rule for the nonrecursive point constructors. 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.

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

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

▶ **Lemma 8.** *The types* $A\ (\textbf{\textit{hocolim}}\ F\ f)$ *and* $\textbf{\textit{hocolim}}\ (A \circ F)\ (A\ f)$ *are isomorphic for all polynomials* A.

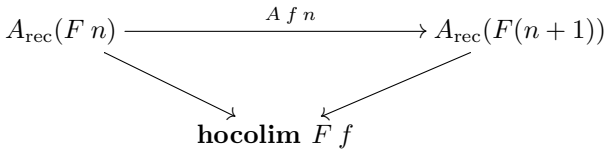
Proof. ◀

Now we will construct the map $A_{\text{rec}}\ (\mathbf{hocolim}\ F\ f) \to \mathbf{hocolim}\ F\ f$, and by Lemma 8 it suffices to make a map $\mathbf{hocolim}\ (A_{\text{rec}} \circ F)\ (A\ f) \to \mathbf{hocolim}\ F\ f$. For this we use the recursion rule of **hocolim**, and we start with the following string of maps

$$A_{\text{rec}}(F\ n) \xrightarrow{c'_{\text{rec}}} H_{\text{rec}}(F\ n) \xrightarrow{\mathbf{term} \circ \iota_1} H_{\mathbf{Con}}(F\ n) \xrightarrow{q_1} H_{\mathbf{Approx}}(F\ n) = F(n+1)$$

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

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



Let us start with the following rectangle

$$\begin{array}{ccccc} A_{\text{rec}}(F\ n) & \xrightarrow{c'_{\text{rec}}} & H_{\text{rec}}(F\ n) & \xrightarrow{\mathbf{term} \circ \iota_1} & H_{\mathbf{Con}}(F\ n) \\ \downarrow{\scriptstyle A\ f\ n} & & \downarrow{\scriptstyle H_{\text{rec}}\ f\ n} & & \downarrow{\scriptstyle H_{\mathbf{Con}}\ f\ n} \\ A_{\text{rec}}(F(n+1)) & \xrightarrow[c'_{\text{rec}}]{} & H_{\text{rec}}(F(n+1)) & \xrightarrow[\mathbf{term} \circ \iota_1]{} & H_{\mathbf{Con}}(F(n+1)) \end{array}$$

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.

$$\begin{array}{ccc} H_{\mathbf{Con}}(F\ n) & \xrightarrow{q_1} & H_{\mathbf{Approx}}(F\ n) \\ {\scriptstyle H_{\mathbf{Con}}\ f\ n}\downarrow & & \downarrow{\scriptstyle f\ n} \\ H_{\mathbf{Con}}(F(n+1)) & \xrightarrow[q_1]{} & H_{\mathbf{Approx}}(F(n+1)) \end{array}$$

By expanding the definitions, we see that $H_{\mathbf{Approx}}(F\ n) \to H_{\mathbf{Approx}}(F(n+1))$ is defined by $p_1 \circ \mathbf{term} \circ \iota_1$. This means that it suffices to show that the upper triangle in the following square commutes.

$$\begin{array}{ccc} H_{\mathbf{Con}}(F\ n) & \xrightarrow{p_1} & H_{\mathbf{Approx}}(F\ n) \\ {\scriptstyle H_{\mathbf{Con}}\ f\ n}\downarrow & \swarrow{\scriptstyle \mathbf{term}\circ\iota_0} & \downarrow{\scriptstyle f\ n} \\ H_{\mathbf{Con}}(F(n+1)) & \xrightarrow[p_1]{} & H_{\mathbf{Approx}}(F(n+1)) \end{array}$$

The upper triangle commutes by definition. All in all, we get the desired map

$$A\ (\mathbf{hocolim}\ F\ f) \longrightarrow \mathbf{hocolim}\ (A \circ F)\ (A\ f) \longrightarrow \mathbf{hocolim}\ F\ f$$

Next we need to define the introduction rules for the paths. For this we first need to observe that a type X has the paths $p_i : \prod(x : B_i\ X), t_i = r_i$ iff the following diagram is a pushout

$$\begin{array}{ccc} \sum_{i=1}^m \mathbf{2} \times (B_i\ X) & \longrightarrow & \sum_{i=1}^m B_i\ X \times I^1 \\ {\scriptstyle t_i + r_i}\downarrow & & \downarrow \\ X & \longrightarrow & X \end{array}$$

We shall use this observation to show that $\mathbf{hocolim}\ F\ f$ has the right paths. Taking X to be $\mathbf{hocolim}\ F\ f$, we thus need to show the following diagram is a pushout.

$$\begin{array}{ccc} \sum_{i=1}^m \mathbf{2} \times (B_i\ (\mathbf{hocolim}\ F\ f)) & \longrightarrow & \sum_{i=1}^m I^1 \times (B_i\ (\mathbf{hocolim}\ F\ f)) \\ \downarrow & & \downarrow \\ \mathbf{hocolim}\ F\ f & \longrightarrow & \mathbf{hocolim}\ F\ f \end{array}$$

Note that homotopy colimits commute with products, so we can rewrite it to where the summations are implicit.

$$\begin{array}{ccc} \mathbf{hocolim}\ ((2 \times B_i) \circ F)\ ((2 \times B_i)\ f) & \longrightarrow & \mathbf{hocolim}\ ((B_i \times I) \circ F)\ ((B_i \times I)\ f) \\ \downarrow & & \downarrow \\ \mathbf{hocolim}\ F\ f & \longrightarrow & \mathbf{hocolim}\ F\ f \end{array}$$

Since homotopy colimits commute with pushouts, it suffices to show that

$$\begin{array}{ccc} \sum_{i=1}^m 2 \times (B_i\ (F\ n)) & \longrightarrow & \sum_{i=1}^k I^1 \times B_i(F\ n) \\ \downarrow & & \downarrow \\ F(n+1) & \longrightarrow & F(n+1) \end{array}$$

This holds by definition of $F(n+1)$.

4.2 Elimination Rule

The next step will be to show that **hocolim** $F\ f$ has the right eliminator.

4.3 Computation Rules

5 Lemmata

▶ **Lemma 9.** *Colimits commute with coproducts.*

▶ **Lemma 10.** *Colimits commute with products.*

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

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.