

# Issue II: Inductive Types

Maksym Sokhatskyi <sup>1</sup>

<sup>1</sup> National Technical University of Ukraine  
Igor Sikorsky Kyiv Polytechnical Institute  
June 1, 2025

## Abstract

Impredicative Encoding of Inductive Types in HoTT.

**Keywords:** Formal Methods, Type Theory, Programming Languages,  
Theoretical Computer Science, Applied Mathematics, Cubical Type Theory,  
Martin-Löf Type Theory

## Contents

<b>1</b>	<b>Inductive Encodings</b>	<b>2</b>
1.1	Church Encoding . . . . .	2
1.2	Scott Encoding . . . . .	2
1.3	Parigot Encoding . . . . .	2
1.4	CPS Encoding . . . . .	2
1.5	Interaction Networks Encoding . . . . .	2
1.6	Impredicative Encoding . . . . .	2
1.7	Lambek Encoding: Homotopy Initial Algebras . . . . .	3
<b>2</b>	<b>Inductive Types</b>	<b>4</b>
2.1	W . . . . .	4
2.2	M . . . . .	5
2.3	Empty . . . . .	6
2.4	Unit . . . . .	7
2.5	Bool . . . . .	8
2.6	Maybe . . . . .	8
2.7	Either . . . . .	8
2.8	Nat . . . . .	8
2.9	List . . . . .	8
2.10	Vector . . . . .	8
2.11	Stream . . . . .	8
2.12	Interpreter . . . . .	8

# 1 Inductive Encodings

## 1.1 Church Encoding

You know Church encoding which also has its dependent analogue in CoC, however in Coq it is impossible to derive Inductive Principle as type system lacks fixpoint and functional extensionality. The example of working compiler of PTS languages are Om and Morte. Assume we have Church encoded NAT:

$\text{nat} = (X:U) \rightarrow (X \rightarrow X) \rightarrow X \rightarrow X$

where first parameter  $(X \rightarrow X)$  is a *succ*, the second parameter  $X$  is *zero*, and the result of encoding is landed in  $X$ . Even if we encode the parameter

$\text{list } (A: U) = (X:U) \rightarrow X \rightarrow (A \rightarrow X) \rightarrow X$

and parameter  $A$  let's say live in 42 universe and  $X$  live in 2 universe, then by the signature of encoding the term will be landed in  $X$ , thus 2 universe. In other words such dependency is called impredicative displaying that landed term is not a predicate over parameters. This means that Church encoding is incompatible with predicative type checkers with predicative of predicative-cumulative hierarchies.

## 1.2 Scott Encoding

## 1.3 Parigot Encoding

## 1.4 CPS Encoding

## 1.5 Interaction Networks Encoding

## 1.6 Impredicative Encoding

In HoTT  $n$ -types is encoded as  $n$ -groupoids, thus we need to add a predicate in which  $n$ -type we would like to land the encoding:

$\text{NAT } (A: U) = (X:U) \rightarrow \text{isSet } X \rightarrow X \rightarrow (A \rightarrow X) \rightarrow X$

Here we added *isSet* predicate. With this motto we can implement propositional truncation by landing term in *isProp* or even HIT by landing in *isGroupoid*:

$\text{TRUN } (A:U) \text{ type} = (X: U) \rightarrow \text{isProp } X \rightarrow (A \rightarrow X) \rightarrow X$   
 $\text{S1} = (X:U) \rightarrow \text{isGroupoid } X \rightarrow ((x:X) \rightarrow \text{Path } X \ x \ x) \rightarrow X$   
 $\text{MONOPL} (A:U) = (X:U) \rightarrow \text{isSet } X \rightarrow (A \rightarrow X) \rightarrow X$   
 $\text{NAT} = (X:U) \rightarrow \text{isSet } X \rightarrow X \rightarrow (A \rightarrow X) \rightarrow X$

The main publication on this topic could be found at [11] and [10].

### The Unit Example

Here we have the implementation of Unit impredicative encoding in HoTT.

```

upPath      (X Y:U)(f:X→Y)(a:X→X): X → Y = o X X Y f a
downPath    (X Y:U)(f:X→Y)(b:Y→Y): X → Y = o X Y Y b f
naturality  (X Y:U)(f:X→Y)(a:X→X)(b:Y→Y): U
            = Path (X→Y)(upPath X Y f a)(downPath X Y f b)

unitEnc': U = (X: U) → isSet X → X → X
isUnitEnc (one: unitEnc'): U
          = (X Y:U)(x:isSet X)(y:isSet Y)(f:X→Y) →
            naturality X Y f (one X x)(one Y y)

unitEnc: U = (x: unitEnc') * isUnitEnc x
unitEncStar: unitEnc = (\(X:U)(_:isSet X) →
  idfun X,\(X Y: U)(_:isSet X)(_:isSet Y)→refl(X→Y))
unitEncRec  (C: U) (s: isSet C) (c: C): unitEnc → C
            = \ (z: unitEnc) → z.1 C s c
unitEncBeta (C: U) (s: isSet C) (c: C)
            : Path C (unitEncRec C s c unitEncStar) c = refl C c
unitEncEta  (z: unitEnc): Path unitEnc unitEncStar z = undefined
unitEncInd  (P: unitEnc → U) (a: unitEnc): P unitEncStar → P a
            = subst unitEnc P unitEncStar a (unitEncEta a)
unitEncCondition (n: unitEnc'): isProp (isUnitEnc n)
            = \ (f g: isUnitEnc n) →
              <h> \ (x y: U) → \ (X: isSet x) → \ (Y: isSet y)
              → \ (F: x → y) → <i> \ (R: x → Y (F (n x X R))) (n y Y (F R))
              (<j> f x y X Y F @ j R) (<j> g x y X Y F @ j R) @ h @ i

```

## 1.7 Lambek Encoding: Homotopy Initial Algebras

## 2 Inductive Types

### 2.1 W

Well-founded trees without mutual recursion represented as W-types.

**Definition 1.** (W-Formation). For  $A : \mathcal{U}$  and  $B : A \rightarrow \mathcal{U}$ , type  $W$  is defined as  $W(A, B) : \mathcal{U}$  or

$$W_{(x:A)}B(x) : \mathcal{U}.$$

$\text{def } W \ (A : \mathcal{U}) \ (B : A \rightarrow \mathcal{U}) : \mathcal{U} := W \ (x : A), B \ x$

**Definition 2.** (W-Introduction). Elements of  $W_{(x:A)}B(x)$  are called well-founded trees and created with single  $\text{sup}$  constructor:

$$\text{sup} : W_{(x:A)}B(x).$$

$\text{def } \text{sup}\$'\$ \ (A : \mathcal{U}) \ (B : A \rightarrow \mathcal{U}) \ (x : A) \ (f : B \ x \rightarrow W \ A \ B)$   
 $\quad : W \ A \ B$   
 $\quad := \text{sup } A \ B \ x \ f$

**Theorem 1.** (Induction Principle  $\text{ind}_W$ ). The induction principle states that for any types  $A : \mathcal{U}$  and  $B : A \rightarrow \mathcal{U}$  and type family  $C$  over  $W(A, B)$  and the function  $g : G$ , where

$$G = \prod_{x:A} \prod_{f:B(x) \rightarrow W(A,B)} \prod_{b:B(x)} C(f(b)) \rightarrow C(\text{sup}(x, f))$$

there is a dependent function:

$$\text{ind}_W : \prod_{C:W(A,B) \rightarrow \mathcal{U}} \prod_{g:G} \prod_{a:A} \prod_{f:B(a) \rightarrow W(A,B)} \prod_{b:B(a)} C(f(b)).$$

$\text{def } W\text{-ind} \ (A : \mathcal{U}) \ (B : A \rightarrow \mathcal{U})$   
 $\quad (C : (W \ (x : A), B \ x) \rightarrow \mathcal{U})$   
 $\quad (g : \prod_{(x : A)} (f : B \ x \rightarrow (W \ (x : A), B \ x)) \rightarrow C \ (\text{sup } A \ B \ x \ f))$   
 $\quad (\prod_{(b : B \ x)} C \ (f \ b)) \rightarrow C \ (\text{sup } A \ B \ x \ f))$   
 $\quad (a : A) \ (f : B \ a \rightarrow (W \ (x : A), B \ x)) \ (b : B \ a)$   
 $\quad : C \ (f \ b) := \text{ind}^W \ A \ B \ C \ g \ (f \ b)$

**Theorem 2.** ( $\text{ind}_W$  Computes). The induction principle  $\text{ind}^W$  satisfies the equation:

$$\text{ind}_W\text{-}\beta : g(a, f, \lambda b. \text{ind}^W(g, f(b))) \\ =_{\text{def}} \text{ind}_W(g, \text{sup}(a, f)).$$

$\text{def } \text{ind}^W\text{-}\beta \ (A : \mathcal{U}) \ (B : A \rightarrow \mathcal{U})$   
 $\quad (C : (W \ (x : A), B \ x) \rightarrow \mathcal{U}) \ (g : \prod_{(x : A)} (f : B \ x \rightarrow (W \ (x : A), B \ x)) \rightarrow C \ (\text{sup } A \ B \ x \ f))$   
 $\quad (f : B \ x \rightarrow (W \ (x : A), B \ x)) \rightarrow C \ (\text{sup } A \ B \ x \ f))$   
 $\quad (a : A) \ (f : B \ a \rightarrow (W \ (x : A), B \ x)) \ (b : B \ a)$   
 $\quad : \text{PathP} \ (<->) \ C \ (\text{sup } A \ B \ a \ f))$   
 $\quad (\text{ind}^W \ A \ B \ C \ g \ (\text{sup } A \ B \ a \ f))$   
 $\quad (g \ a \ f \ (\lambda (b : B \ a), \text{ind}^W \ A \ B \ C \ g \ (f \ b)))$   
 $\quad := <-> \ g \ a \ f \ (\lambda (b : B \ a), \text{ind}^W \ A \ B \ C \ g \ (f \ b))$

## 2.2 M

## 2.3 Empty

The Empty type represents False-type logical  $\mathbf{0}$ , type without inhabitants, void or  $\perp$  (Bottom). As it has not inhabitants it lacks both constructors and eliminators, however, it has induction.

**Definition 3.** (Formation). Empty-type is defined as built-in  $\mathbf{0}$ -type:

$$\mathbf{0} : \mathcal{U}.$$

**Theorem 3.** (Induction Principle  $\text{ind}_0$ ).  $\mathbf{0}$ -type is satisfying the induction principle:

$$\text{ind}_0 : \prod_{C : \mathbf{0} \rightarrow \mathcal{U}} \prod_{z : \mathbf{0}} C(z).$$

`def Empty-ind (C:  $\mathbf{0} \rightarrow \mathcal{U}$ ) (z:  $\mathbf{0}$ ) : C z := ind0 (C z) z`

**Definition 4.** (Negation or isEmpty). For any type A negation of A is defined as arrow from A to  $\mathbf{0}$ :

$$\neg A := A \rightarrow \mathbf{0}.$$

`def isEmpty (A:  $\mathcal{U}$ ):  $\mathcal{U}$  := A  $\rightarrow \mathbf{0}$`

The witness of  $\neg A$  is obtained by assuming A and deriving a contradiction. This techniques is called proof of negation and is applicable to any types in constrast to proof by contradiction which implies  $\neg\neg A \rightarrow A$  (double negation elimination) and is applicable only to decidable types with  $\neg A + A$  property.

## 2.4 Unit

Unit type is the simplest type equipped with full set of MLTT inference rules. It contains single inhabitant  $\star$  (star).

- 2.5 Bool
- 2.6 Maybe
- 2.7 Either
- 2.8 Nat
- 2.9 List
- 2.10 Vector
- 2.11 Stream
- 2.12 Interpreter



## References

- [1] Frank Pfenning and Christine Paulin-Mohring, *Inductively Defined Types in the Calculus of Constructions*, in *Proc. 5th Int. Conf. Mathematical Foundations of Programming Semantics*, 1989, pp. 209–228. doi:10.1007/BFb0040259
- [2] Christine Paulin-Mohring, *Inductive Definitions in the System Coq: Rules and Properties*, in *Typed Lambda Calculi and Applications (TLCA)*, 1993, pp. 328–345. doi:10.1007/BFb0037116
- [3] Christine Paulin-Mohring, *Defining Inductive Sets in Type Theory*, in: G. Huet and G. Plotkin (eds), *Logical Environments*, Cambridge University Press, 1994, pp. 249–272.
- [4] Peter Dybjer, *Inductive Sets and Families in Martin-Löf’s Type Theory and Their Set-Theoretic Semantics*, *Lecture Notes in Computer Science*, 530, 1991, pp. 280–306. doi:10.1007/BFb0014059
- [5] Peter Dybjer, *Inductive Families*, *Formal Aspects of Computing*, 6(4), 1994, pp. 440–465. doi:10.1007/BF01211308
- [6] Peter Dybjer, *Representing inductively defined sets by wellorderings in Martin-Löf’s type theory*, *Theoretical Computer Science*, 176(1–2), 1997, pp. 329–335. doi:10.1016/S0304-3975(96)00145-4
- [7] Martin Hofmann, *Extensional Constructs in Intensional Type Theory*, PhD thesis, University of Edinburgh, 1995. <https://www2.informatik.uni-freiburg.de/~mhofmann/phdthesis.pdf>
- [8] Martin Hofmann, *Syntax and Semantics of Dependent Types*, in: *Semantics and Logics of Computation*, 1995, pp. 79–130.
- [9] Newstead, C. (2018). *Algebraic Models of Dependent Type Theory*. PhD thesis, Carnegie Mellon University. Available at <https://arxiv.org/abs/2103.06155>.
- [10] Sam Speight, *Impredicative Encoding of Inductive Types in HoTT*, 2017. <https://github.com/sspeight93/Papers/>
- [11] Steve Awodey, *Impredicative Encodings in HoTT*, 2017. <https://www.newton.ac.uk/files/seminar/20170711090010001-1009680.pdf>
- [12] Steve Awodey. *Type theory and homotopy*, 2010. <https://arxiv.org/abs/1010.1810>
- [13] Thorsten Altenkirch, Neil Ghani, Peter Hancock, Conor McBride, and Peter Morris. *Indexed Containers*. *Logical Methods in Computer Science*, 18(2), 2022, pp. 15:1–15:37. <https://lmcs.episciences.org/>

- [14] Marcelo P. Fiore, Andrew M. Pitts, and S. C. Steenkamp. *Quotients, Inductive Types, & Quotient Inductive Types*. University of Cambridge, 2022. <https://arxiv.org/pdf/1705.07088>
- [15] Thorsten Altenkirch, Neil Ghani, and Peter Morris. *Containers—Constructively*, 2012. <https://arxiv.org/pdf/1201.3898>
- [16] Thorsten Altenkirch, Conor McBride, and James Chapman. *Towards Observational Type Theory*, 2013. <https://arxiv.org/pdf/1307.2765>
- [17] Peter Dybjer, *Representing inductively defined sets by wellorderings in Martin-Löf’s type theory*, *Theoretical Computer Science*, 176(1–2), 1997, pp. 329–335. doi:10.1016/S0304-3975(96)00145-4
- [18] Ieke Moerdijk and Erik Palmgren, *Wellfounded trees in categories*, *Annals of Pure and Applied Logic*, 104(1–3), 2000, pp. 189–218. doi:10.1016/S0168-0072(00)00012-9
- [19] Michael Abbott, Thorsten Altenkirch, and Neil Ghani, *Containers: Constructing strictly positive types*, *Theoretical Computer Science*, 342(1), 2005, pp. 3–27. doi:10.1016/j.tcs.2005.06.002
- [20] Benno van den Berg and Ieke Moerdijk, *W-types in sheaves*, 2008. <https://arxiv.org/abs/0810.2398>
- [21] Nicola Gambino and Martin Hyland, *Wellfounded Trees and Dependent Polynomial Functors*, in *TYPES 2003*, LNCS 3085, Springer, 2004, pp. 210–225. doi:10.1007/978-3-540-24849-1\_14
- [22] Michael Abbott, Thorsten Altenkirch, and Neil Ghani, *Representing Nested Inductive Types using W-types*, in *ICALP 2004*, LNCS 3142, Springer, 2004, pp. 124–135. doi:10.1007/978-3-540-27836-8\_8
- [23] Steve Awodey, Nicola Gambino, and Kristina Sojakova, *Inductive types in homotopy type theory*, *LICS 2012*, pp. 95–104. doi:10.1109/LICS.2012.21, <https://arxiv.org/abs/1201.3898>
- [24] Benno van den Berg and Ieke Moerdijk, *W-types in Homotopy Type Theory*, *Mathematical Structures in Computer Science*, 25(5), 2015, pp. 1100–1115. doi:10.1017/S0960129514000516, <https://arxiv.org/abs/1307.2765>
- [25] Kristina Sojakova, *Higher Inductive Types as Homotopy-Initial Algebras*, *ACM SIGPLAN Notices*, 50(1), 2015, pp. 31–42. doi:10.1145/2775051.2676983, <https://arxiv.org/abs/1402.0761>
- [26] Steve Awodey, Nicola Gambino, and Kristina Sojakova, *Homotopy-initial algebras in type theory*, *Journal of the ACM*, 63(6), 2017, Article 45. doi:10.1145/3006383, <https://arxiv.org/abs/1504.05531>

- [27] Christian Sattler, *On relating indexed W-types with ordinary ones*, in *TYPES 2015*, pp. 71–72. <https://types2015.inria.fr/slides/sattler.pdf>
- [28] Per Martin-Löf, *Constructive Mathematics and Computer Programming*, in: Proc. 6th Int. Congress of Logic, Methodology and Philosophy of Science, 1979. *Studies in Logic and the Foundations of Mathematics* 104 (1982), pp. 153–175. doi:10.1016/S0049-237X(09)70189-2
- [29] Per Martin-Löf (notes by Giovanni Sambin), *Intuitionistic type theory*, Lecture notes Padua 1984, Bibliopolis, Napoli (1984).
- [30] Jasper Hugunin, *Why Not W?*, *LIPICs*, 188 (TYPES 2020), 2021. doi:10.4230/LIPICs.TYPES.2020.8
- [31] Nils Anders Danielsson, *Positive h-levels are closed under W*, 2012. <https://www.cse.chalmers.se/~nad/listings/w-level/WLevel.html>
- [32] Jasper Hugunin, *IWTypes Repository*. <https://github.com/jashug/IWTypes>