Anders: Modal Homotopy Type System

Namdak Tönpa
Groupoid Infinity

Abstract

Here is presented a reincarnation of **cubicaltt** called **anders**.

2012 ACM Subject Classification Theory of computation → Lambda calculus

Keywords and phrases Homotopy Type System, Cubical Type Theory, Modal HoTT

1 Introduction

Anders is a Modal HoTT proof assistant based on: classical MLTT-80 [?] with 0, 1, 2, W types; CCHM [?] in CHM [?] flavour as cubical type system with hcomp/transp Kan operations; HTS [?] strict equality on pretypes; de Rham [?] stack modality primitives. We tend not to touch general recursive higher inductive schemes yet, instead we will try to express as much HIT as possible through W, Coequlizer and HubSpokes Disc in the style of HoTT/Coq homotopy library and Three-HIT theorem.

The HTS language proposed by Voevodsky exposes two different presheaf models of type theory: the inner one is homotopy type system presheaf that models HoTT and the outer one is traditional Martin-Löf type system presheaf that models set theory with UIP. The motivation behind this doubling is to have an ability to express semisemplicial types. Theoretical work on merging inner and outer languages was continued in 2LTT [?].

Installation. While we are on our road to Lean-like tactic language, currently we are at the stage of regular cubical HTS type checker with CHM-style primitives. You may try it from Github sources: groupoid/anders[1] or install through OPAM package manager. Main commands are **check** (to check a program) and **repl** (to enter the proof shell).

```
$ opam install anders
```

Anders is fast, idiomatic and educational (think of optimized Mini-TT). We carefully draw the favourite Lean-compatible syntax to fit 200 LOC in Menhir. The CHM kernel is 1K LOC. Whole Anders compiles under 2 seconds and checks all the base library under 1 second [i7-8700]. Anders proof assistant as Homotopy Type System comes with its own Homotopy Library[2].

2 Syntax

The syntax resembles original syntax of the reference CCHM type checker cubicaltt, is slightly compatible with Lean syntax and contains the full set of Cubical Agda [?] primitives (except generic higher inductive schemes).

Here is given the mathematical pseudo-code notation of the language expressions that come immediately after parsing. The core syntax definition of HTS language corresponds to the type defined in OCaml module:

Further Menhir BNF notation will be used to describe the top-level language E parser.

[1] `https://github.com/groupoid/anders/`
[2] `https://anders.groupoid.space/library/`

$$\begin{array}{rl}
\text{cosmos} := & \mathbf{U}_j \mid \mathbf{V}_k \\
\text{var} := & \mathbf{var}\ \mathit{name} \mid \mathbf{hole} \\
\text{pi} := & \Pi\ \mathit{name}\ E\ E \mid \lambda\ \mathit{name}\ E\ E \mid E\ E \\
\text{sigma} := & \Sigma\ \mathit{name}\ E\ E \mid (E, E) \mid E.1 \mid E.2 \\
0 := & \mathbf{0} \mid \mathbf{ind}_0\ \text{E E E} \\
1 := & \mathbf{1} \mid \star \mid \mathbf{ind}_1\ \text{E E E} \\
2 := & \mathbf{2} \mid 0_2 \mid 1_2 \mid \mathbf{ind}_2\ \text{E E E} \\
\text{W} := & \mathbf{W}\ \text{ident E E} \mid \mathbf{sup}\ \text{E E} \mid \mathbf{ind}_W\ \text{E E} \\
\text{id} := & \mathbf{Id}\ E \mid \mathbf{ref}\ E \mid \mathbf{id}_J\ E \\
\text{path} := & \mathbf{Path}\ E \mid E^i \mid E\ @\ E \\
\text{I} := & \mathbf{I} \mid 0 \mid 1 \mid E \bigvee E \mid E \bigwedge E \mid \neg E \\
\text{part} := & \mathbf{Partial}\ E\ E \mid [\ (E = I) \to E, \ldots\ ] \\
\text{sub} := & \mathbf{inc}\ E \mid \mathbf{ouc}\ E \mid E\ [\ I\ \mapsto\ E\ ] \\
\text{kan} := & \mathbf{transp}\ E\ E \mid \mathbf{hcomp}\ E \\
\text{glue} := & \mathbf{Glue}\ E \mid \mathbf{glue}\ E \mid \mathbf{unglue}\ E\ E \\
\text{Im} := & \mathbf{Im}\ \text{E} \mid \mathbf{Inf}\ \text{E} \mid \mathbf{Join}\ \text{E} \mid \mathbf{ind}_{Im}\ \text{E E} \\
\\
\text{E} := & \text{cosmos} \mid \text{var} \mid \text{MLTT} \mid \text{CCHM} \mid \text{Im} \\
\text{CCHM} := & \text{path} \mid \text{I} \mid \text{part} \mid \text{sub} \mid \text{kan} \mid \text{glue} \\
\text{MLTT} := & \text{pi} \mid \text{sigma} \mid \text{id}
\end{array}$$

Keywords. The words of a top-level language, file or repl, consist of keywords or identifiers. The keywords are following: `module`, `where`, `import`, `option`, `def`, `axiom`, `postulate`, `theorem`, `(`, `)`, `[`, `]`, `<`, `>`, `/`, `.1`, `.2`, Π, Σ, `,`, λ, `V`, ⋁, ⋀, `-`, `+`, `@`, `PathP`, `transp`, `hcomp`, `zero`, `one`, `Partial`, `inc`, ×, →, `:`, `:=`, ↦, `U`, `ouc`, `interval`, `inductive`, `Glue`, `glue`, `unglue`.

Indentifiers. Identifiers support UTF-8. Indentifiers couldn't start with `:`, `-`, →. Sample identifiers: `¬-of-∨`, `1→1`, `is-?`, `=`, `$∼]!005x`, ∞, `x→Nat`.

Modules. Modules represent files with declarations. More accurate, BNF notation of module consists of imports, options and declarations.

```
menhir
    start <Module.file> file
    start <Module.command> repl
    repl : COLON IDENT exp1 EOF | COLON IDENT EOF | exp0 EOF | EOF
    file : MODULE IDENT WHERE line* EOF
    path : IDENT
    line : IMPORT path+ | OPTION IDENT IDENT | declarations
```

Imports. The import construction supports file folder structure (without file extensions) by using reserved symbol / for hierarchy walking.

Options. Each option holds bool value. Language supports following options: 1) girard (enables U : U); 2) pre-eval (normalization cache); 3) impredicative (infinite hierarchy with impredicativity rule); In Anders you can enable or disable language core types, adjust syntaxes or tune inner variables of the type checker.

Declarations. Language supports following top level declarations: 1) axiom (non-computable declaration that breakes normalization); 2) postulate (alternative or inverted axiom that can preserve consistency); 3) definition (almost any explicit term or type in type theory); 4) lemma (helper in big game); 5) theorem (something valuable or complex enough).

```
axiom isProp (A : U) : U
def isSet (A : U) : U := Π (a b : A) (x y : Path A a b), Path (Path A a b) x y
```

Sample declarations. For example, signature isProp (A : U) of type U could be defined as normalization-blocking axiom without proof-term or by providing proof-term as definition.

In this example (A : U), (a b : A) and (x y : Path A a b) are called telescopes. Each telescope consists of a series of lenses or empty. Each lense provides a set of variables of the same type. Telescope defines parameters of a declaration. Types in a telescope, type of a declaration and a proof-terms are a language expressions exp1.

```
menhir
   ident : IRREF | IDENT
   lense : LPARENS ident+ COLON exp1 RPARENS
   telescope : lense telescope
   params : telescope | []
   declarations :
      | DEF IDENT params DEFEQ exp1
      | DEF IDENT params COLON exp1 DEFEQ exp1
      | AXIOM IDENT params COLON exp1
```

Expressions. All atomic language expressions are grouped by four categories: exp0 (pair constructions), exp1 (non neutral constructions), exp2 (path and pi applcations), exp3 (neutral constructions).

```
menhir
   face : LPARENS IDENT IDENT IDENT RPARENS
   part : face+ ARROW exp1
   exp0 : exp1 COMMA exp0 | exp1
   exp1 : LSQ separated(COMMA, part) RSQ
      | LAM telescope COMMA exp1     | PI telescope COMMA exp1
      | SIGMA telescope COMMA exp1   | LSQ IRREF ARROW exp1 RSQ
      | LT ident+ GT exp1            | exp2 ARROW exp1
      | exp2 PROD exp1               | exp2
```

The LR parsers demand to define exp1 as expressions that cannot be used (without a parens enclosure) as a right part of left-associative application for both Path and Pi lambdas. Universe indicies U_j (inner fibrant), V_k (outer pretypes) and S (outer strict omega) are using unicode subscript letters that are already processed in lexer.

```
menhir
   exp2 : exp2 exp3 | exp2 APPFORMULA exp3 | exp3
   exp3 : LPARENS exp0 RPARENS LSQ exp0 MAP exp0 RSQ
      | HOLE           | PRE           | KAN            | IDJ exp3
      | exp3 FST       | exp3 SND      | NEGATE exp3    | INC exp3
      | exp3 AND exp3  | exp3 OR exp3  | ID exp3        | REF exp3
      | OUC exp3       | PATHP exp3    | PARTIAL exp3   | IDENT
      | IDENT LSQ exp0 MAP exp0 RSQ                     | HCOMP exp3
      | LPARENS exp0 RPARENS                            | TRANSP exp3 exp3
```

3 Semantics

The idea is to have a unified layered type checker, so you can disbale/enable any MLTT-style inference, assign types to universes and enable/disable hierachies. This will be done by providing linking API for pluggable presheaf modules. We selected 5 levels of type checker awareness from universes and pure type systems up to synthetic language of homotopy type theory. Each layer corresponds to its presheaves with separate configuration for universe hierarchies. We want to mention here with homage to its authors all categorical models of

```
inductive lang : U   :=  UNI: cosmos → lang
                      |   PI: pure lang → lang
                      |   SIGMA: total lang → lang
                      |   ID: uip lang → lang
                      |   PATH: homotopy lang → lang
                      |   GLUE: gluening lang → lang
                      |   HIT: hit lang → lang
```

dependent type theory: Comprehension Categories (Grothendieck, Jacobs), LCCC (Seely), D-Categories and CwA (Cartmell), CwF (Dybjer), C-Systems (Voevodsky), Natural Models (Awodey). While we can build some transports between them, we leave this excercise for our mathematical components library. We will use here the Coquand's notation for Presheaf Type Theories in terms of restriction maps.

3.1 Universe Hierarchies

Language supports Agda-style hierarchy of universes: prop, fibrant (U), interval pretypes (V) and strict omega with explicit level manipulation. All universes are bounded with preorder

$$Prop_i \prec Fibrant_j \prec Pretypes_k \prec Strict_l, \tag{1}$$

in which i, j, k, l are bounded with equation:

$$i < j < k < l. \tag{2}$$

Large elimination to upper universes is prohibited. This is extendable to Agda model:

```
inductive cosmos : U   :=  prop: nat → cosmos
                        |   fibrant: nat
                        |   pretypes: nat
                        |   strict: nat
                        |   omega
                        |   lock
```

The **anders** model contains only fibrant U_j and pretypes V_k universe hierarchies.

3.2 Dependent Types

▶ **Definition 1** (Type). *A type is interpreted as a presheaf A, a family of sets A_I with restriction maps $u \mapsto u\ f, A_I \to A_J$ for $f : J \to I$. A dependent type B on A is interpreted by a presheaf on category of elements of A: the objects are pairs (I, u) with $u : A_I$ and morphisms $f : (J, v) \to (I, u)$ are maps $f : J \to$ such that $v = u\ f$. A dependent type B is thus given by a family of sets $B(I, u)$ and restriction maps $B(I, u) \to B(J, u\ f)$.*

We think of A as a type and B as a family of presheves $B(x)$ varying $x : A$. The operation $\Pi(x : A)B(x)$ generalizes the semantics of implication in a Kripke model.

▶ **Definition 2** (Pi). *An element $w : [\Pi(x : A)B(x)](I)$ is a family of functions $w_f : \Pi(u : A(J))B(J, u)$ for $f : J \to I$ such that $(w_f u)g = w_{f\ g}(u\ g)$ when $u : A(J)$ and $g : K \to J$.*

```
inductive pure (lang : U) : U   :=  pi: name → nat → lang → lang → pure lang
                                 |  lambda: name → nat → lang → lang
                                 |  app: lang → lang
```

▶ **Definition 3** (Sigma). *The set $\Sigma(x : A)B(x)$ is the set of pairs (u, v) when $u : A(I), v : B(I, u)$ and restriction map $(u, v)\ f = (u\ f, v\ f)$.*

```
inductive total (lang : U) : U   :=  sigma: name → lang → total lang
                                  |  pair: lang → lang
                                  |  fst: lang
                                  |  snd: lang
```

The presheaf with only Pi and Sigma is called **MLTT-72** [?]. Its internalization in **anders** is as follows:

```
def MLTT-72 (A : U) (B : A → U) : U := Σ
    (Π-form₁ : U)
    (Π-ctor₁ : Pi A B → Pi A B)
    (Π-elim₁ : Pi A B → Pi A B)
    (Π-comp₁ : (a : A) (f : Pi A B), Π-elim₁ (Π-ctor₁ f) a = f a)
    (Π-comp₂ : (a : A) (f : Pi A B), f = λ (x : A), f x)
    (Σ-form₁ : U)
    (Σ-ctor₁ : Π (a : A) (b : B a) , Sigma A B)
    (Σ-elim₁ : Π (p : Sigma A B), A)
    (Σ-elim₂ : Π (p : Sigma A B), B (pr₁ A B p))
    (Σ-comp₁ : Π (a : A) (b: B a), a = Σ-elim₁ (Σ-ctor₁ a b))
    (Σ-comp₂ : Π (a : A) (b: B a), b = Σ-elim₂ (a, b))
    (Σ-comp₃ : Π (p : Sigma A B), p = (pr₁ A B p, pr₂ A B p)), 1
```

3.3 Path Equality

The fundamental development of equality inside MLTT provers led us to the notion of ∞-groupoid as spaces. In this way Path identity type appeared in the core of type checker along with De Morgan algebra on built-in interval type.

```
inductive homotopy (lang : U) : U   :=  PathP: lang → lang → lang
                                    |   plam: name → lang → lang
                                    |   papp: lang → lang
                                    |   I
                                    |   zero
                                    |   one
                                    |   meet: lang → lang
                                    |   join: lang → lang
                                    |   neg: lang
                                    |   system: lang
                                    |   Partial: lang
                                    |   transp: lang → lang
                                    |   hcomp: lang
                                    |   Sub: lang
                                    |   inc: lang
                                    |   ouc: lang
```

▶ **Definition 4** (Cubical Presheaf $\mathbb{I}$). *. The identity types modeled with another presheaf, the presheaf on Lawvere category of distributive lattices (theory of De Morgan algebras) denoted with* $\square$ *—* $\boldsymbol{I} : \square^{op} \rightarrow Set$*.*

▶ **Definition 5** (Properties of **I**. The presheaf **I**). *: i) has to distinct global elements* 0 *and* 1 *(B_1); ii) **I**(I) has a decidable equality for each I (B_2); iii) **I** is tiny so the path functor* $X \mapsto X^{\boldsymbol{I}}$ *has right adjoint (B_3).; iv) **I** has meet and join (connections).*

Interval Pretypes. While having pretypes universe V with interval and associated De Morgan algebra ($\wedge$, $\vee$, -, 0, 1, I) is enough to perform DNF normalization and proving some basic statements about path, including: contractability of singletons, homotopy transport, congruence, functional extensionality; it is not enough for proving β rule for Path type or path composition.

Generalized Transport. Generalized transport transp adresses first problem of deriving the computational β rule for Path types:

```
theorem Path_β (A : U) (a : A) (C : D A) (d: C a a (refl A a))
    : Equ (C a a (refl A a)) d (J A a C d a (refl A a))
    := λ (A : U), λ (a : A),
       λ (C : Π (x : A), Π (y : A), PathP (<_> A) x y → U),
       λ (d : C a a (<_> a)), <j> transp (<_> C a a (<_> a)) -j d
```

Transport is defined on fibrant types (only) and type checker should cover all the cases Note that transp^i (Path^j A v w) φ u_0 case is relying on comp operation which depends on hcomp primitive. Here is given the first part of Simon Huber equations [**?**] for **transp**:

```
transp^i N φ u0 = u0
transp^i U φ A = A
transp^i (Π (x : A), B) φ u0 v = transp^i B(x/w) φ (u0 w(i/0))
transp^i (Σ (x : A), B) φ u0 = (transp^i A φ (u0.1),transp^i B(x/v) φ (u0.2))
transp^i (Path^j v w) φ u0 = <j> comp^i A [ϕ u0 j, (j=0) ↦ v, (j=1) ↦ w] (u0 j)
transp^i (Glue [φ ↦ (T,w)] A) ψ u0 = glue [ϕ(i/1) ↦ t'1] a'1 : B(i/1)
```

Partial Elements. In order to explicitly define hcomp we need to specify n-cubes where some faces are missing. Partial primitives isOne, 1=1 and UIP on pretypes are derivable in Anders due to landing strict equality Id in V universe. The idea is that (Partial A r) is the type of cubes in A that are only defined when IsOne r holds. (Partial A r) is a special version of the function space IsOne r → A with a more extensional equality: two of its elements are considered judgmentally equal if they represent the same subcube of A. They are equal whenever they reduce to equal terms for all the possible assignment of variables that make r equal to 1.

```
def Partial' (A : U) (i : I) := Partial A i
def isOne : I -> V := Id I 1
def 1=>1 : isOne 1 := ref 1
def UIP (A : V) (a b : A) (p q : Id A a b) : Id (Id A a b) p q := ref p
```

Cubical Subtypes. For (A : U) (i : I) (Partial A i) we can define subtype A [i ↦ u]. A term of this type is a term of type A that is definitionally equal to u when (IsOne i) is satisfied. We have forth and back fusion rules ouc (inc v) = v and inc (outc v) = v. Moreover, ouc v will reduce to u 1=1 when i=1.

```
def sub' (A : U) (i : I) (u : Partial A i) : V := A [i ↦ u ]
def inc' (A : U) (i : I) (a : A) : A [i ↦ [(i = 1) → a]] := inc A i a
def ouc' (A : U) (i : I) (u : Partial A i) (a : A [i ↦ u]) : A := ouc a
```

Homogeneous Composition. hcomp is the answer to second problem: with hcomp and transp one can express path composition, groupoid, category of groupoids (groupoid interpretation and internalization in type theory). One of the main roles of homogeneous composition is to be a carrier in [higher] inductive type constructors for calculating of homotopy colimits and direct encoding of CW-complexes. Here is given the second part of Simon Huber equations [?] for **hcomp**:

```
hcomp^i N [φ ↦ 0] 0 = 0
hcomp^i N [φ ↦ S u] (S u₀) = S (hcomp^i N [φ ↦ u] u₀)
hcomp^i U [φ ↦ E] A = Glue [φ ↦ (E(i/1), equiv^i E(i/1-i))] A
hcomp^i (Π (x : A), B) [φ ↦ u] u₀ v = hcomp^i B(x/v) [φ ↦ u v] (u₀ v)
hcomp^i (Σ (x : A), B) [φ ↦ u] u₀ = (v(i/1), comp^i B(x/v) [φ ↦ u.2] u₀.2)

hcomp^i (Path^j A v w) [φ ↦ u] u₀
       = <j> hcomp^i A[φ ↦ u j, (j=0) ↦ v, (j=1) ↦ w] (u₀ j)

hcomp^i (Glue [φ ↦ (T,w)] A) [ψ ↦ u] u₀
       = glue [φ ↦ t₁] a₁
       = glue [φ ↦ u(i/1)] (unglue u(i/1))
       = u(i/1) : Glue [φ ↦ (T,w)] A
```

3.4 Strict Equality

To avoid conflicts with path equalities which live in fibrant universes strict equalities live in pretypes universes.

```
inductive strict (lang : U) : U   :=   Id: name → lang
                                   |   ref: lang → lang
                                   |   idJ: lang → lang → lang
```

We use strict equality in HTS for pretypes and partial elements which live in V. The presheaf configuration with Pi, Sigma and Id is called **MLTT-73** [?]. The presheaf configuration with Pi, Sigma, Id and Path is called **HTS**.

3.5 Glue Types

The main purpose of Glue types is to construct a cube where some faces have been replaced by equivalent types. This is analogous to how hcomp lets us replace some faces of a cube by composing it with other cubes, but for Glue types you can compose with equivalences instead of paths. This implies the univalence principle and it is what lets us transport along paths built out of equivalences.

```
inductive glue (lang : U) : U  :=  Glue: lang → lang → lang
                                |  glue: lang → lang
                                |  unglue: lang → lang
```

Basic Fibrational HoTT core by Pelayo, Warren, and Voevodsky (2012).

```
def fiber (A B : U) (f: A → B) (y : B): U := Σ (x : A), Path B y (f x)
def isEquiv (A B : U) (f : A → B) : U := Π (y : B), isContr (fiber A B f y)
def equiv (A B : U) : U := Σ (f : A → B), isEquiv A B f
def contrSingl (A : U) (a b : A) (p : Path A a b)
    : Path (Σ (x : A), Path A a x) (a,<i>a) (b,p) := <i> (p @ i, <j> p @ i ∨ j)
def idIsEquiv (A : U) : isEquiv A A (id A) :=
    λ (a : A), ((a,<i>a), λ (z : fiber A A (id A) a), contrSingl A a z.1 z.2)
def idEquiv (A : U) : equiv A A := (id A, isContrSingl A)
```

The notion of Univalence was discovered by Vladimir Voevodsky as forth and back transport between fibrational equivalence as contractability of fibers and homotopical multi-dimentional heterogeneous path equality. The Equiv → Path type is called Univalence type, where univalence intro is obtained by Glue type and elim (Path → Equiv) is obtained by sigma transport from constant map.

```
def univ-formation (A B : U) := equiv A B → PathP (<i> U) A B
def univ-intro (A B : U) : univ-formation A B := λ (e : equiv A B),
    <i> Glue B (∂ i) [(i = 0) → (A, e), (i = 1) → (B, idEquiv B)]
def univ-elim (A B : U) (p : PathP (<i> U) A B)
  : equiv A B := transp (<i> equiv A (p @ i)) 0 (idEquiv A)
def univ-computation (A B : U) (p : PathP (<i> U) A B)
  : PathP (<i> PathP (<i> U) A B) (univ-intro A B (univ-elim A B p)) p
  := <j i> Glue B (j ∨ ∂ i)
     [(i = 0) → (A, univ-elim A B p), (i = 1) → (B, idEquiv B),
      (j = 1) → (p @ i, univ-elim (p @ i) B (<k> p @ (i ∨ k)))]
```

Similar to Fibrational Equivalence the notion of Retract/Section based Isomorphism could be introduced as forth-back transport between isomorphism and path equality. This notion is somehow cannonical to all cubical systems and is called Unimorphism here.

```
def iso-Form (A B: U) : U_1 := iso A B -> PathP (<i>U) A B
def iso-Intro (A B: U) : iso-Form A B :=
   λ (x : iso A B), isoPath A B x.f x.g x.s x.t
def iso-Elim (A B : U) : PathP (<i> U) A B -> iso A B
 := λ (p : PathP (<i> U) A B),
   ( coerce A B p, coerce B A (<i> p @ -i),
     trans^{-1}-trans A B p, λ (a : A), <k> trans-trans^{-1} A B p a @-k, ⋆)
```

Orton-Pitts basis for univalence computability (2017):

```
def ua (A B : U) (p : equiv A B) : PathP (<i> U) A B := univ-intro A B p
def ua−β (A B : U) (e : equiv A B) : Path (A → B) (trans A B (ua A B e)) e.1
:= <i> λ (x : A), (idfun=idfun" B @ -i)
    ( (idfun=idfun" B @ -i) ((idfun=idfun' B @ -i) (e.1 x)) )
```

3.6 de Rham Stack

Stack de Rham or Infinitezemal Shape Modality is a basic primitive for proving theorems from synthetic differential geometry. This type-theoretical framework was developed for the first time by Felix Cherubini under the guidance of Urs Schreiber. The Anders prover implements the computational semantics of the de Rham stack.

```
def ι (A : U) (a : A) : ℑ A := ℑ—unit a
def μ (A : U) (a : ℑ (ℑ A)) := ℑ—join a

def is-coreduced (A : U) : U := isEquiv A (ℑ A) (ι A)
def ℑ-coreduced (A : U) : is-coreduced (ℑ A) := isoToEquiv
    (ℑ A) (ℑ (ℑ A)) (ι (ℑ A)) (μ A) (λ (x : ℑ (ℑ A)), <i>x) (λ (y : ℑ A),<i>y)
def ind—ℑβ (A : U) (B : ℑ A → U) (f : Π (a : A), ℑ (B (ι A a))) (a : A)
  : Path (ℑ (B (ι A a))) (ind—ℑ A B f (ι A a)) (f a) := <i> f a
def ind—ℑ—const (A B : U) (b : ℑ B) (x : ℑ A)
  : Path (ℑ B) (ind—ℑ A (λ (i : ℑ A), B) (λ (i : A), b) x) b := <i> b
```

Coreduced induction and its $\beta-$quation.

```
def ℑ—ind (A : U) (B : ℑ A → U) (c : Π (a : ℑ A),
    is-coreduced (B a)) (f : Π (a : A), B (ι A a)) (a : ℑ A) : B a
    := (c a (ind—ℑ A B (λ (x : A), ι (B (ι A x)) (f x)) a)).1.1
def ℑ—indβ (A : U) (B : ℑ A → U) (c : Π (a : ℑ A),
    is-coreduced (B a)) (f : Π (a : A), B (ι A a)) (a : A)
  : Path (B (ι A a)) (f a) ((ℑ—ind A B c f) (ι A a))
 := <i> sec-equiv (B (ι A a)) (ℑ (B (ι A a))) (ι (B (ι A a)), c (ι A a)) (f a) @-i
```

Geometric Modal HoTT Framework: Infinitesimal Proximity, Formal Disk, Formal Disk Bundle, Differential.

```
def ∼ (X : U) (a x' : X) : U := Path (ℑ X) (ι X a) (ι X x')
def 𝔻 (X : U) (a : X) : U := Σ (x' : X), ∼ X a x'
def inf-prox-ap (X Y : U) (f : X → Y) (x x' : X) (p : ∼ X x x')
  : ∼ Y (f x) (f x') := <i> ℑ—app X Y f (p @ i)
def T∞ (A : U) : U := Σ (a : A), 𝔻 A a
def inf-prox-ap (X Y : U) (f : X → Y) (x x' : X) (p : ∼ X x x')
  : ∼ Y (f x) (f x') := <i> ℑ—app X Y f (p @ i)
def d (X Y : U) (f : X → Y) (x : X) (ε : 𝔻 X x)
  : 𝔻 Y (f x) := (f ε.1, inf-prox-ap X Y f x ε.1 ε.2)
def T∞-map (X Y : U) (f : X → Y) (τ : T∞ X) : T∞ Y := (f τ.1, d X Y f τ.1 τ.2)
```

3.7 Higher Inductive Types

Anders currently don't support Lean-compatible generic inductive schemes definition. So instead of generic inductive schemes Anders supports well-founded trees (W-types). Basic data types like List, Nat, Fin, Vec are implemented as W-types in base library. As for higher inductive types Anders has Three-HIT basis (Coequalizer, HubSpoke and Colimit) to express other HIT.