# Issue III: Homotopy Type Theory

Maxim Sokhatsky [1]

[1] National Technical University of Ukraine
Igor Sikorsky Kyiv Polytechnical Institute

April 25, 2025

**Abstract**

Here is presented destinctive points of Homotopy Type Theory as an extension of Martin-Löf Type Theory but without higher inductive types which will be given in the next issue. The fibrational (geometric) interpretation of equivalence type is introduced with following univalence releation between equivalence and Path equality. Groupoid (categorical) interpretation is presented as categories of spaces and paths between them as invertible morphisms. At last constructive proof $\Omega(S^1) = \mathbb{Z}$ is given through helix.

**Keywords**: Homotopy Theory, Type Theory

# Contents

# 1 Introduction

## 1.1 Introduction: Type Theory

Type theory is a universal programming language for pure mathematics, designed for theorem proving. It supports an arbitrary number of consistent axioms, structured as pseudo-isomorphisms consisting of *encode* functions (methods for constructing type elements), *decode* functions (dependent eliminators of the universal induction principle), and their equations—beta and eta rules governing computability and uniqueness. As a programming language, type theory includes basic primitives (axioms as built-in types) and accompanying documentation, such as lecture notes or textbooks, explaining their applications, including:

- Functions ($\mathbf{\Pi}$)
- Contexts ($\mathbf{\Sigma}$)
- Identifications ($=$)
- Polynomials ($\mathbf{W}$)
- Paths ($\Xi$)
- Gluings (**Glue**)
- Infinitesimals ($\Im$)
- Complexes (**HIT**)

Students (10) are tasked with applying type theory to prove an initial but non-trivial result addressing an open problem in one of the following areas offered by the Department of Pure Mathematics (KM-111):

$$\begin{cases} \text{Homotopy Theory} \\ \text{Homological Algebra} \\ \text{Category Theory} \\ \text{Functional Analysis} \\ \text{Differential Geometry} \end{cases}$$

The course is divided into four parts, each exploring type-axioms and their meta-theoretical adjunctions.

## 1.2 Motivation: Homotopy Type Theory

The primary motivation of homotopy type theory is to provide computational semantics for homotopic types and CW-complexes. The central idea, as described in, is to combine function spaces (Π), context spaces (Σ), and path spaces (Ξ) to form a fibrational equality, proven within the theory to coincide with the path space.

Key definitions include:

```
def isContr (A: U) : U
 := Σ (x: A), Π (y: A), Path A x y

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 (X Y: U): U
 := Σ (f: X → Y), isEquiv X Y f

def ua (A B : U) (p : Path U A B) : equiv A B
 := transp (<i> equiv A (p @ i)) 0 (idEquiv A)
```

The absence of an eta-rule for equality implies that not all proofs of the same path space are equal, resulting in a multidimensional $\infty$-groupoid structure for path spaces. Further definitions include:

```
def isProp (A : U) : U
 := Π (a b : A), Path A a b

def isSet (A : U) : U
 := Π (a b : A) (a0 b0 : Path A a b),
    Path (Path A a b) a0 b0

def isGroupoid (A : U) : U
 := Π (a b : A) (x y : Path A a b)
    (i j : Path (Path A a b) x y),
    Path (Path (Path A a b) x y) i j
```

The groupoid interpretation raises questions about the existence of a language for mechanically proving all properties of the categorical definition of a groupoid:

```
def CatGroupoid (X : U) (G : isGroupoid X)
  : isCatGroupoid (PathCat X)
 := ( idp X,
      comp-Path X,
      G,
      sym X,
      comp-inv-Path⁻¹ X,
      comp-inv-Path X,
      comp-Path-left X,
      comp-Path-right X,
      comp-Path-assoc X,
      ⋆
    )
```

## 1.3 Fibrational Proofs

$$\Sigma \dashv f_{\star} \dashv \Pi$$

Fibrational proofs are modeled by primitive axioms, which are type-theoretic representations of categorical meta-theoretical models of adjunctions of three Cockett-Reit functors, giving rise to function spaces ($\Pi$) and pair spaces ($\Sigma$). These proof methods enable direct analysis of fibrations.

## 1.4 Equality Proofs

$$\mathrm{Q} \dashv \Xi \dashv \mathrm{C}$$

In intensional type theory, the equality type is embedded as type-theoretic primitives of categorical meta-theoretical models of adjunctions of three Jacobs-Lambek functors: quotient space (Q), identification system ($\Xi$), and contractible space (C). These methods allow direct manipulation of identification systems, strict for set theory and homotopic for homotopy theory.

## 1.5 Inductive Proofs

$$\mathrm{W} \dashv \odot \dashv \mathrm{M}$$

Inductive types in type theory can be embedded as polynomial functors (W, M) or general inductive type schemes (Calculus of Inductive Constructions), with properties including: 1) Verification of program finiteness; 2) Verification of strict positivity of parameters; 3) Verification of mutual recursion.

In this course, induction and coinduction are introduced as type-theoretic primitives of categorical meta-theoretical models of adjunctions of polynomial functors (Lambek-Bohm), enabling manipulation of initial and terminal algebras, algebraic recursive data types, and infinite processes. Higher inductive proofs, where constructors include path spaces, are modeled by polynomial functors using monad-algebras and comonad-coalgebras (Lumsdaine-Shulman-Voevodsky).

## 1.6 Geometric Proofs

$$\Re \dashv \Im \dashv \&$$

For differential geometry, type theory incorporates primitive axioms of categorical meta-theoretical models of three Schreiber-Shulman functors: infinitesimal neighborhood ($\Im$), reduced modality ($\Re$), and infinitesimal discrete neighborhood (&).

## 1.7 Linear Proofs

$$\otimes \dashv x \dashv\!\multimap$$

For engineering applications (e.g., Milner's $\pi$-calculus, quantum computing) and linear type theory, type theory embeds linear proofs based on the adjunction of the tensor and linear function spaces:

$$(A \otimes B) \multimap A \simeq A \multimap (B \multimap C),$$

represented in a symmetric monoidal category $\mathbf{D}$ for a functor $[A, B]$ as:

$$\mathbf{D}(A \otimes B, C) \simeq \mathbf{D}(A, [B, C]).$$

## 1.8 Historical Notes

Homotypy Type Theory takes its origins in 1996 from groupoid interpretation by Hofmann and Streicher's, and later (in 10 years) was formalized by Awodey, Warren and Voevodsky. Voevodsky constrtucted Kan simplicial sets interpretation of type theory and discovered the property of this model, that was named univalence. This property allows to identify isomorphic structures in terms of type theory.

Homotopy type theory to classical homotopy theory is like Euclidian syntethic geometry (points, lines, axioms and deduction rules) to analytical geometry with cartesian coordinates on $\mathbb{R}^n$ (geometric and algebraic) [1]

In the same way as inductive types extends MLTT for inductive programming, the higher inductive types (HIT) extend homotopy type theory for geometry programming. You can directly encode CW-complexes by using HIT. The definition of HIT syntax will be given in the next **Issue IV: Higher Inductive Types**.

[1] We will denote geometric, type theoretical and homotopy constants bold font $\mathbf{R}$ while analitical will be denoted with double lined letters $\mathbb{R}$.

# 2 Homotopy Type Theory

## 2.1 Homotopies

The first higher equality we meet in homotopy theory is a notion of homotopy, where we compare two functions or two path spaces (which is sort of dependent families). The homotopy interval $\mathrm{I} = [0, 1]$ is the perfect foundation for definition of homotopy.

**Definition 1** *(Interval). Compact interval.*

```
def I : U := inductive { i0 | i1 | seg : i0 ≡ i1 }
```

You can think of $\mathbf{I}$ as isomorphism of equality type, disregarding carriers on the edges. By mapping $i0, i1 : \mathbf{I}$ to $x, y : A$ one can obtain identity or equality type from classic type theory.

**Definition 2** *(Interval Split). The convertion function from* I *to a type of comparison is a direct eliminator of interval. The interval is also known as one of primitive higher inductive types which will be given in the next* **Issue IV: Higher Inductive Types***.*

```
def pathToHtpy (A: U) (x y: A) (p: Path A x y) : I → A
 := split { i0 → x | i1 → y | seg @ i → p @ i }
```

**Definition 3** *(Homotopy). The homotopy between two function $f, g : X \to Y$ is a continuous map of cylinder $H : X \times \mathbf{I} \to Y$ such that*

$$\begin{cases} H(x,0) = f(x), \\ H(x,1) = g(x). \end{cases}$$

```
homotopy (X Y: U) (f g: X -> Y)
         (p: (x: X) -> Path Y (f x) (g x))
         (x: X): I -> Y = pathToHtpy Y (f x) (g x) (p x)
```

## 2.2 Groupoid Interpretation

The first text about groupoid interpretation of type theory can be found in Francois Lamarche: A proposal about Foundations[2]. Then Martin Hofmann and Thomas Streicher wrote the initial document on groupoid interpretation of type theory[3].

| **Equality** | **Homotopy** | **$\infty$-Groupoid** |
|---|---|---|
| reflexivity | constant path | identity morphism |
| symmetry | inversion of path | inverse morphism |
| transitivity | concatenation of paths | composition of mopphisms |

There is a deep connection between higher-dimentinal groupoids in category theory and spaces in homotopy theory, equipped with some topology. The category or groupoid could be built where the objects are particular spaces or types, and morphisms are path types between these types, composition operation is a path concatenation. We can write this groupoid here recalling that it should be category with inverted morphisms.

```
cat: U = (A: U) * (A -> A -> U)
groupoid: U = (X: cat) * isCatGroupoid X
PathCat (X: U): cat = (X,\(x y:X)->Path X x y)

def isCatGroupoid (C: cat): U := Σ
    (id:        Π (x: C.ob), C.hom x x)
    (c:         Π (x y z:C.ob), C.hom x y -> C.hom y z -> C.hom x z)
    (HomSet:    Π (x y: C.ob), isSet (C.hom x y))
    (inv:       Π (x y: C.ob), C.hom x y -> C.hom y x)
    (inv-left:  Π (x y: C.ob) (p: C.hom x y),
                Ξ (C.hom x x) (c x y x p (inv x y p)) (id x))
    (inv-right: Π (x y: C.ob) (p: C.hom x y),
                Ξ (C.hom y y) (c y x y (inv x y p) p) (id y))
    (left:      Π (x y: C.ob) (f: C.hom x y),
                Ξ (C.hom x y) f (c x x y (id x) f))
    (right:     Π (x y: C.ob) (f: C.hom x y),
                Ξ (C.hom x y) f (c x y y f (id y)))
    (assoc:     Π (x y z w: C.ob) (f: C.hom x y)
                  (g: C.hom y z) (h: C.hom z w),
                Ξ (C.hom x w) (c x z w (c x y z f g) h)
                              (c x y w f (c y z w g h))), ⋆
```

[2]http://www.cse.chalmers.se/~coquand/Proposal.pdf

[3]Martin Hofmann and Thomas Streicher. The Groupoid Interpretation of Type Theory. 1996.

```
def isProp (A : U) : U
 := Π (a b : A), Path A a b

def isSet (A : U) : U
 := Π (a b : A) (a0 b0 : Path A a b),
    Path (Path A a b) a0 b0

def isGroupoid (A : U) : U
 := Π (a b : A) (x y : Path A a b)
    (i j : Path (Path A a b) x y),
    Path (Path (Path A a b) x y) i j

def CatGroupoid (X : U) (G : isGroupoid X)
  : isCatGroupoid (PathCat X)
 := ( idp X,
      comp-Path X,
      G,
      sym X,
      comp-inv-Path^-1 X,
      comp-inv-Path X,
      comp-Path-left X,
      comp-Path-right X,
      comp-Path-assoc X,
      ⋆
    )
```

## 2.3 Identity Systems

**Definition 4** *(Identity System). An identity system over type $A$ in universe $X_i$ is a family $R : A \to A \to X_i$ with a function $r_0 : \Pi_{a:A} R(a,a)$ such that any type family $D : \Pi_{a,b:A} R(a,b) \to X_i$ and $d : \Pi_{a:A} D(a,a,r_0(a))$, there exists a function $f : \Pi_{a,b:A} \Pi_{r:R(a,b)} D(a,b,r)$ such that $f(a,a,r_0(a)) = d(a)$ for all $a : A$.*

```
def IdentitySystem (A : U) : U
 := Σ (=-form : A → A → U)
      (=-ctor : Π (a : A), =-form a a)
      (=-elim : Π (a : A) (C: Π (x y : A)
                  (p : =-form x y), U)
                  (d : C a a (=-ctor a)) (y : A)
                  (p : =-form a y), C a y p)
      (=-comp : Π (a : A) (C: Π (x y : A)
                  (p : =-form x y), U)
                  (d : C a a (=-ctor a)),
                  Path (C a a (=-ctor a)) d
                       (=-elim a C d a (=-ctor a))), 1
```

**Example 1** *There are number of equality signs used in this tutorial, all of them listed in the following table of identity systems:*

| Sign | Meaning |
|---|---|
| $=_{def}$ | Definition |
| $=$ | Id |
| $\equiv$ | Path |
| $\simeq$ | Equivalence |
| $\cong$ | Isomorphism |
| $\sim$ | Homotopy |
| $\approx$ | Bisimulation |

**Theorem 1** *(Fundamental Theorem of Identity System).*

**Definition 5** *(Strict Identity System). An identity system over type $A$ and universe of pretypes $V_i$ is called strict identity system ($=$), which respects UIP.*

**Definition 6** *(Homotopy Identity System). An identity system over type $A$ and universe of homotopy types $U_i$ is called homotopy identity system ($\equiv$), which models discrete infinity groupoid.*

## 2.4 Functional Extensionality

**Definition 7** *(funExt-Formation)*

```
funext_form (A B: U) (f g: A -> B): U
  = Path (A -> B) f g
```

**Definition 8** *(funExt-Introduction)*

```
funext (A B: U) (f g: A -> B) (p: (x:A) -> Path B (f x) (g x))
  : funext_form A B f g
  = <i> \(a: A) -> p a @ i
```

**Definition 9** *(funExt-Elimination)*

```
happly (A B: U) (f g: A -> B) (p: funext_form A B f g) (x: A)
  : Path B (f x) (g x)
  = cong (A -> B) B (\(h: A -> B) -> apply A B h x) f g p
```

**Definition 10** *(funExt-Computation)*

```
funext_Beta (A B: U) (f g: A -> B) (p: (x:A) -> Path B (f x) (g x))
  : (x:A) -> Path B (f x) (g x)
  = \(x:A) -> happly A B f g (funext A B f g p) x
```

**Definition 11** *(funExt-Uniqueness)*

```
funext_Eta (A B: U) (f g: A -> B) (p: Path (A -> B) f g)
  : Path (Path (A -> B) f g) (funext A B f g (happly A B f g p)) p
  = refl (Path (A -> B) f g) p
```

## 2.5 Fibrations

**Definition 12** *(Fibration-1) Dependent fiber bundle derived from Path contractability.*

```
isFBundle1 (B: U) (p: B -> U) (F: U): U
  = (_: (b: B) -> isContr (Path U (p b) F))
  * ((x: Sigma B p) -> B)
```

**Definition 13** *(Fibration-2). Dependent fiber bundle derived from surjective function.*

```
isFBundle2 (B: U) (p: B -> U) (F: U): U
  = (V: U)
  * (v: surjective V B)
  * ((x: V) -> Path U (p (v.1 x)) F)
```

**Definition 14** *(Fibration-3). Non-dependent fiber bundle derived from fiber truncation.*

```
im1 (A B: U) (f: A -> B): U = (b: B) * pTrunc ((a:A) * Path B (f a) b)
BAut (F: U): U = im1 unit U (\(x: unit) -> F)
unitIm1 (A B: U) (f: A -> B): im1 A B f -> B = \(x: im1 A B f) -> x.1
unitBAut (F: U): BAut F -> U = unitIm1 unit U (\(x: unit) -> F)

isFBundle3 (E B: U) (p: E -> B) (F: U): U
  = (X: B -> BAut F)
  * (classify B (BAut F) (\(b: B) -> fiber E B p b) (unitBAut F) X) where
  classify (A' A: U) (E': A' -> U) (E: A -> U) (f: A' -> A): U
    = (x: A') -> Path U (E'(x)) (E(f(x)))
```

**Definition 15** *(Fibration-4). Non-dependen fiber bundle derived as pullback square.*

```
isFBundle4 (E B: U) (p: E -> B) (F: U): U
  = (V: U)
  * (v: surjective V B)
  * (v': prod V F -> E)
  * pullbackSq (prod V F) E V B p v.1 v' (\(x: prod V F) -> x.1)
```

## 2.6 Equivalence

**Definition 16** *(Equivalence).*

```
fiber (A B: U) (f: A -> B) (y: B): U = (x: A) * Path B y (f x)
isSingleton (X:U): U = (c:X) * ((x:X) -> Path X c x)
isEquiv (A B: U) (f: A -> B): U = (y: B) -> isContr (fiber A B f y)
equiv (A B: U): U = (f: A -> B) * isEquiv A B f
```

**Definition 17** *(Surjective).*

```
isSurjective (A B: U) (f: A -> B): U
  = (b: B) * pTrunc (fiber A B f b)

surjective (A B: U): U
  = (f: A -> B)
  * isSurjective A B f
```

**Definition 18** *(Injective).*

```
isInjective' (A B: U) (f: A -> B): U
  = (b: B) -> isProp (fiber A B f b)

injective (A B: U): U
  = (f: A -> B)
  * isInjective A B f
```

**Definition 19** *(Embedding).*

```
isEmbedding (A B: U) (f: A -> B) : U
  = (x y: A) -> isEquiv (Path A x y) (Path B (f x) (f y)) (cong A B f x y)

embedding (A B: U): U
  = (f: A -> B)
  * isEmbedding A B f
```

**Definition 20** *(Half-adjoint Equivalence).*

```
isHae (A B: U) (f: A -> B): U
  = (g: B -> A)
  * (eta_: Path (id A) (o A B A g f) (idfun A))
  * (eps_: Path (id B) (o B A B f g) (idfun B))
  * ((x: A) -> Path B (f ((eta_ @ 0) x)) ((eps_ @ 0) (f x)))

hae (A B: U): U
  = (f: A -> B)
  * isHae A B f
```

## 2.7 Isomorphism

**Definition 21** *(iso-Formation)*

```
iso_Form (A B: U): U = isIso A B -> Path U A B
```

**Definition 22** *(iso-Introduction)*

```
iso_Intro (A B: U): iso_Form A B
```

**Definition 23** *(iso-Elimination)*

```
iso_Elim (A B: U): Path U A B -> isIso A B
```

**Definition 24** *(iso-Computation)*

```
iso_Comp (A B : U) (p : Path U A B)
  : Path (Path U A B) (iso_Intro A B (iso_Elim A B p)) p
```

**Definition 25** *(iso-Uniqueness)*

```
iso_Uniq (A B : U) (p: isIso A B)
  : Path (isIso A B) (iso_Elim A B (iso_Intro A B p)) p
```

## 2.8 Univalence

**Definition 26** *(uni-Formation)*

```
univ_Formation (A B: U): U = equiv A B -> Path U A B
```

**Definition 27** *(uni-Introduction)*

```
equivToPath (A B: U): univ_Formation A B
  = \(p: equiv A B) -> <i> Glue B [(i=0) -> (A,p),
    (i=1) -> (B, subst U (equiv B) B B (<_>B) (idEquiv B)) ]
```

**Definition 28** *(uni-Elimination)*

```
pathToEquiv (A B: U) (p: Path U A B) : equiv A B
  = subst U (equiv A) A B p (idEquiv A)
```

**Definition 29** *(uni-Computation)*

```
eqToEq (A B : U) (p : Path U A B)
  : Path (Path U A B) (equivToPath A B (pathToEquiv A B p)) p
  = <j i> let Ai: U = p@i in Glue B
    [ (i=0) -> (A,pathToEquiv A B p),
      (i=1) -> (B,pathToEquiv B B (<k> B)),
      (j=1) -> (p@i,pathToEquiv Ai B (<k> p @ (i \/ k))) ]
```

**Definition 30** *(uni-Uniqueness)*

```
transPathFun (A B : U) (w: equiv A B)
  : Path (A -> B) w.1 (pathToEquiv A B (equivToPath A B w)).1
```

## 2.9 Loop Spaces

**Definition 31** *(Pointed Space). A pointed type $(A, a)$ is a type $A : U$ together with a point $a : A$, called its basepoint.*

```
pointed: U = (A: U) * A
point (A: pointed): A.1 = A.2
space (A: pointed): U = A.1
```

**Definition 32** *(Loop Space).*

$$\Omega(A, a) =_{def} ((a =_A a), refl_A(a)).$$

```
omega1 (A: pointed) : pointed
  = (Path (space A) (point A) (point A), refl A.1 (point A))
```

**Definition 33** *(n-Loop Space).*

$$\begin{cases} \Omega^0(A, a) =_{def} (A, a) \\ \Omega^{n+1}(A, a) =_{def} \Omega^n(\Omega(A, a)) \end{cases}$$

```
omega : nat -> pointed -> pointed = split
  zero -> idfun pointed
  succ n -> \(A: pointed) -> omega n (omega1 A)
```

### 2.10 Homotopy Groups

**Definition 34** *(n-th Homotopy Group of m-Sphere).*

$$\pi_n S^m = ||\Omega^n(S^m)||_0.$$

```
piS (n: nat): (m: nat) -> U = split
    zero    -> sTrunc (space (omega n (bool,false)))
    succ x -> sTrunc (space (omega n (Sn (succ x),north)))
```

**Theorem 2** *($\Omega(S^1) = \mathbb{Z}$).*

```
data S1 = base
        | loop <i> [ (i=0) -> base ,
                     (i=1) -> base ]

loopS1 : U = Path S1 base base

encode (x:S1) (p:Path S1 base x)
  : helix x
  = subst S1 helix base x p zeroZ

decode : (x:S1) -> helix x -> Path S1 base x = split
  base -> loopIt
  loop @ i -> rem @ i where
    p : Path U (Z -> loopS1) (Z -> loopS1)
      = <j> helix (loop1@j) -> Path S1 base (loop1@j)
    rem : PathP p loopIt loopIt
      = corFib1 S1 helix (\(x:S1)->Path S1 base x) base
        loopIt loopIt loop1 (\(n:Z) ->
        comp (<i> Path loopS1 (oneTurn (loopIt n))
              (loopIt (testIsoPath Z Z sucZ predZ
                       sucpredZ predsucZ n @ i)))
              (<i>(lem1It n)@-i) [])

loopS1eqZ : Path U Z loopS1
  = isoPath Z loopS1 (decode base) (encode base)
    sectionZ retractZ
```