Issue III: Homotopy Type Theory

Maxim Sokhatsky [1]

[1] National Technical University of Ukraine
Igor Sikorsky Kyiv Polytechnical Institute
September 22, 2018

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 Type Theory

Contents

1 Homotypy Type Theory

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**.

1.1 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.

```
data I = i0
       | i1
       | seg <i> [(i=0) -> i0,
                  (i=1) -> 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**.

```
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

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

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

```
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)
```

1.2 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**	**∞-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)

isCatGroupoid (C: cat): U
  = (id: (x: C.1) -> C.2 x x)
  * (c: (x y z:C.1) -> C.2 x y -> C.2 y z -> C.2 x z)
  * (inv: (x y: C.1) -> C.2 x y -> C.2 y x)
  * (inv_left:  (x y: C.1) (p: C.2 x y) ->
    Path (C.2 x x) (c x y x p (inv x y p)) (id x))
  * (inv_right: (x y: C.1) (p: C.2 x y) ->
    Path (C.2 y y) (c y x y (inv x y p) p) (id y))
  * (left: (x y: C.1) (f: C.2 x y) ->
    Path (C.2 x y) (c x x y (id x) f) f)
  * (right: (x y: C.1) (f: C.2 x y) ->
    Path (C.2 x y) (c x y y f (id y)) f)
  * ((x y z w:C.1)(f:C.2 x y)(g:C.2 y z)(h:C.2 z w)->
    Path (C.2 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.

```
PathGrpd (X: U)
  : groupoid
  = ((Ob,Hom),id,c,sym X,compPathInv X,compInvPath X,L,R,Q) where
    Ob: U = X
    Hom (A B: Ob): U = Path X A B
    id (A: Ob): Path X A A = refl X A
    c (A B C: Ob) (f: Hom A B) (g: Hom B C): Hom A C
      = comp (<i> Path X A (g@i)) f []
```

From here should be clear what it meant to be groupoid interpretation of path type in type theory. In the same way we can construct categories of $\prod$ and $\sum$ types. In **Issue VIII: Topos Theory** such categories will be given.

1.3 Functional Extensionality

Definition 4. (funExt-Formation)

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

Definition 5. (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 6. (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 7. (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 8. (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
```

1.4 Pullbacks

1.5 Fibrations

Definition 9. (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 10. (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 11. (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 12. (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)
```

1.6 Equivalence

Definition 13. (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 14. (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 15. (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 16. (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 17. (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
```

1.7 Isomorphism

Definition 18. (iso-Formation)

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

Definition 19. (iso-Introduction)

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

Definition 20. (iso-Elimination)

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

Definition 21. (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 22. (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
```

1.8 Univalence

Definition 23. (uni-Formation)

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

Definition 24. (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 25. (uni-Elimination)

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

Definition 26. (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 27. (uni-Uniqueness)

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

1.9 Loop Spaces

Definition 28. (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 29. (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 30. (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)
```

1.10 Homotopy Groups

Definition 31. (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 1. $(\Omega(S^1) = \mathbb{Z})$.

```
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
```