Lecture Notes for HoTT Course

Maxim Sokhatsky

[1] National Technical University of Ukraine
Igor Sikorsky Kyiv Polytechnical Institute
August 12, 2018

Abstract

This seminar supposed to be a contemporary introduction to modern type theory that is suitable for mathematicians needs, in the sense that this theory forms a foundation of mathematics and the language for theorems formulation and their proofs.

This course is dedicated to type checkers with cubical syntax, based on interval with functional extensionality, higher equalities, and higher inductive types on top of MLTT as a core. Please refer to CCHM paper for more information. The base library used in this course is founded on top of cubical modules each falling into one of 15 categories:

(i) MLTT Types: pi, sigma, path; (ii) Set Theory: prop, set, ordinal, hedberg; (iii) Run-Time Inductive Types: proto, maybe, bool, nat, list, stream, vector; (iv) Abstract Algebra: algebra; (v) Control Structures: control; (vi) Recursion Schemes: recursion; (vii) Univalent Foundations: equiv, retract, iso, univ; (viii) Higher Inductive Types: interval, interval, s1, s2, pullback, pushout, suspension, quotient, trunc; (ix) Process Calculus: process; (x) Category Theory and Topos Theory: cat, fun, sip, adj, ump, cones, topos, category; (xi) Contextual Categories: cwf, csystem; (xii) Languages: mltt, infinity; (xiii) Algebraic Geometry: pointed, euler, seq, hopf, cw; (xiv) Cartan Geometry in Cohesive Topos: infinitesimal, etale, manifold, bundle, shape, sharp; (xv) K-Theory: k_theory, swap-trans, bishop, subtype; This library is best to read with HoTT book and Felix Wellen dissertation.

Keywords: Cubical Type Theory, Homotopy Type Theory, Category Theory, Topos Theory

Contents

Intro

Cubical Syntax

The BNF notation consists of 1) telescopes (contexts or sigma chains); 2) inductive data definitions (sum chains); 3) split eliminator; 4) branches of split eliminators; 5) pure dependent type theory syntax. It also has where, import, module constructions.

```
def := data id tele = sum + id tele : exp = exp +
       id tele : exp where def

exp := cotele*exp + cotele→exp + exp→exp + (exp) + app + id +
       (exp,exp) + \ cotele→exp + split cobrs + exp.1 + exp.2

     0 := #empty          imp    := [ import id ]
   brs := 0 + cobrs       tele   := 0 + cotele
   app := exp exp         cotele := ( exp : exp ) tele
    id := [ #nat ]        sum    := 0 + id tele + id tele | sum
   ids := [ id ]          br     := ids → exp
 codec := def dec         mod    := module id where imp dec
   dec := 0 + codec       cobrs  := | br brs
```

Note that the syntax lacks HITs as for this chapter we don't need ones.

1 Martin-Löf Type Theory

Martin-Löf Type Theory contains Pi, Sigma, Id, Nat, List types. Id types were added in 1984 while original MLTT was introduced in 1972. Now Id types are treated as heterogeneous Path interval types.

1.1 Pi

Definition 1. (Formation).

```
Pi (A: U) (P: B -> U): U = (x: A) -> B x
```

Definition 2. (Introduction).

```
lambda (A B: U) (b: B): A -> B = \ (x: A) -> b
lam (A: U) (B: A -> U)
    (a: A) (b: B a) : A -> B a = \ (x: A) -> b
```

Definition 3. (Elimination).

```
apply (A B: U) (f: A -> B) (x: A): B = f(x)
app (A: U) (B: A -> U)
    (a: A) (f: A -> B a): B a = f a
```

Definition 4. (Computation).

```
Beta (A: U) (B: A -> U) (a: A) (f: A -> B a)
   : Path (B a) (app A B a (lam A B a (f a))) (f a)
```

Definition 5. (Uniqueness).

```
Eta (A: U) (B: A -> U) (a: A) (f: A -> B a)
  : Path (A -> B a) f (\(x:A) -> f x)
```

Examples from Mathematics

Pi is a dependent function type, the generalization of functions. As a function it can serve the wide range of mathematical constructions, objects, types, or spaces. The known domain and codomain spaces could be: sets, functions, polynomial functors, ∞-groupoids, topological ∞-groupoid, cw-complexes, categories, languages, etc. We give here immediate isomorphism of Pi types, the fibrations or fiber bundles.

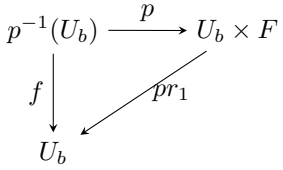
The adjoints Pi and Sigma is not the only adjoints could be presented in type system. Axiomatic cohesions could contain a set of adjoint pairs as a core type checker operations.

Geometrically, Pi type is a space of sections, while the dependent codomain is a space of fibrations. Lambda functions are sections or points in these spaces, while the function result is a fibration. Pi type also represents the cartesian family of sets, generalizing the cartesian product of sets.

Definition 6. (Section). A section of morphism $f : A \to B$ in some category is the morphism $g : B \to A$ such that $f \circ g : B \xrightarrow{g} A \xrightarrow{f} B$ equals the identity morphism on B.

Definition 7. (Fiber). The fiber of the map $p : E \to B$ in a point $y : B$ is all points $x : E$ such that $p(x) = y$.

Definition 8. (Fiber Bundle). The fiber bundle $F \to E \xrightarrow{p} B$ on a total space E with fiber layer F and base B is a structure (F, E, p, B) where $p : E \to B$ is a surjective map with following property: for any point $y : B$ exists a neighborhood U_b for which a homeomorphism $f : p^{-1}(U_b) \to U_b \times F$ making the following diagram commute.



Definition 9. (Cartesian Product of Family over B). Is a set F of sections of the bundle with elimination map $app : F \times B \to E$ such that

$$F \times B \xrightarrow{app} E \xrightarrow{pr_1} B \tag{1}$$

pr_1 is a product projection, so pr_1, app are moriphisms of slice category $Set_{/B}$. The universal mapping property of F: for all A and morphism $A \times B \to E$ in $Set_{/B}$ exists unique map $A \to F$ such that everything commute. So a category with all dependent products is necessarily a category with all pullbacks.

Definition 10. (Trivial Fiber Bundle). When total space E is cartesian product $\Sigma(B, F)$ and $p = pr_1$ then such bundle is called trivial $(F, \Sigma(B, F), pr_1, B)$.

Definition 11. (Dependent Product). The dependent product along morphism $g : B \to A$ in category C is the right adjoint $\Pi_g : C_{/B} \to C_{/A}$ of the base change functor.

Definition 12. (Space of Sections). Let $\mathbf{H}$ be a $(\infty, 1)$-topos, and let $E \to B : \mathbf{H}_{/B}$ a bundle in $\mathbf{H}$, object in the slice topos. Then the space of sections $\Gamma_\Sigma(E)$ of this bundle is the Dependent Product:

$$\Gamma_\Sigma(E) = \Pi_\Sigma(E) \in \mathbf{H}.$$

Theorem 1. (Functions Preserve Paths). For a function $f : (x : A) \to B(x)$ there is an $ap_f : Path_A(x, y) \to Path_{B(x)}(f(x), f(y))$. This is called application of f to path or congruence property (for non-dependent case — *cong* function). This property behaves functoriality as if paths are groupoid morphisms and types are objects.

Theorem 2. (Trivial Fiber equals Family of Sets). Inverse image (fiber) of fiber bundle $(F, B * F, pr_1, B)$ in point $y : B$ equals $F(y)$.

```
FiberPi (B: U) (F: B -> U) (y: B)
      : Path U (fiber (Sigma B F) B (pi1 B F) y) (F y)
```

Theorem 3. (Homotopy Equivalence). If fiber space is set for all base, and there are two functions $f, g : (x : A) \to B(x)$ and two homotopies between them, then these homotopies are equal.

```
setPi (A: U) (B: A -> U) (h: (x: A) -> isSet (B x)) (f g: Pi A B)
      (p q: Path (Pi A B) f g) : Path (Path (Pi A B) f g) p q
```

Theorem 4. (HomSet). If codomain is set then space of sections is a set.

```
setFun (A B : U) (_: isSet B) : isSet (A -> B)
```

Theorem 5. (Contractability). If domain and codomain is contractible then the space of sections is contractible.

```
piIsContr (A: U) (B: A -> U) (u: isContr A)
          (q: (x: A) -> isContr (B x)) : isContr (Pi A B)
```

1.2 Sigma

Definition 13. (Formation).

```
Sigma (A : U) (B : A -> U) : U = (x : A) * B x
```

Sigma is a dependent product type, the generalization of products. Sigma type is a total space of fibration. Element of total space is formed as a pair of basepoint and fibration.

Definition 14. (Introduction).

```
dpair (A: U) (B: A -> U) (a: A) (b: B a) : Sigma A B = (a,b)
```

Definition 15. (Elimination).

```
pr1 (A: U) (B: A -> U)
    (x: Sigma A B): A = x.1

pr2 (A: U) (B: A -> U)
    (x: Sigma A B): B (pr1 A B x) = x.2

sigInd (A: U) (B: A -> U) (C: Sigma A B -> U)
       (g: (a: A) (b: B a) -> C (a, b))
       (p: Sigma A B) : C p = g p.1 p.2
```

Definition 16. (Computation).

```
Beta1 (A: U) (B: A -> U)
      (a:A) (b: B a)
    : Equ A a (pr1 A B (a,b))
    = refl A a

Beta2 (A: U) (B: A -> U)
      (a: A) (b: B a)
    : Equ (B a) b (pr2 A B (a,b))
    = reflect (B a)
```

Definition 17. (Uniqueness).

```
Eta12 (A: U) (B: A -> U) (p: Sigma A B)
    : Equ (Sigma A B) p (pr1 A B p,pr2 A B p)
    = refl (Sigma A B) p
```

Examples from Mathematics

Definition 18. (Dependent Sum). The dependent sum along the morphism $f : A \to B$ in category C is the left adjoint $\Sigma_f : C_{/A} \to C_{/B}$ of the base change functor.

Theorem 6. (Axiom of Choice). If for all $x : A$ there is $y : B$ such that $R(x, y)$, then there is a function $f : A \to B$ such that for all $x : A$ there is a witness of $R(x, f(x))$.

```
ac (A B: U) (R: A -> B -> U)
 : (p: (x:A) -> (y:B)*(R x y)) -> (f:A->B) * ((x:A)->R(x)(f x))
```

Theorem 7. (Total). If fiber over base implies another fiber over the same base then we can construct total space of section over that base with another fiber.

```
total (A:U) (B C: A -> U)
      (f: (x:A) -> B x -> C x) (w: Sigma A B)
    : Sigma A C = (w.1,f (w.1) (w.2))
```

Theorem 8. (Path Between Sigmas). Path between two sigmas $t, u : \Sigma(A, B)$ could be decomposed to sigma of two paths $p : t_1 =_A u_1)$ and $(t_2 =_{B(p@i)} u_2)$.

```
pathSig (A:U) (B : A -> U) (t u : Sigma A B)
      : Path U (Path (Sigma A B) t u)
               ((p: Path A t.1 u.1) * PathP (<i>B(p@i)) t.2 u.2)
```

1.3 Path

The Path identity type defines a Path space with elements and values. Elements of that space are functions from interval $[0, 1]$ to a values of that path space. This ctt file reflects [1]CCHM cubicaltt model with connections. For [2]ABCFHL yacctt model with variables please refer to ytt file. You may also want to read [3]BCH, [4]AFH. There is a [5]PO paper about CCHM axiomatic in a topos.

Definition 19. (Formation).

```
HeteroPath (A B: U) (a: A) (b: B) (P: Path U A B) : U = PathP P a b
Path (A: U) (a b: A) : U = PathP (<i> A) a b
```

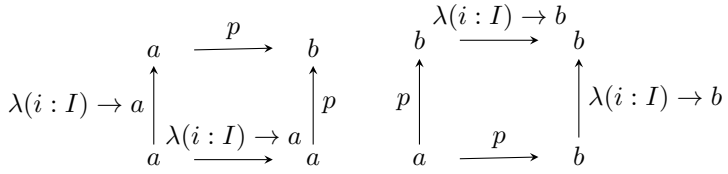
Definition 20. (Introduction). Returns an element of reflexivity path space for a given value of the type. The inhabitant of that path space is the lambda on the homotopy interval $[0, 1]$ that returns a constant value a. Written in syntax as $< i > a = \lambda(i : I) \to a$.

```
refl (A: U) (a: A) : Path A a a
```

Definition 21. (Path Application). You can apply face to path.

```
app1 (A: U) (a b: A) (p: Path A a b): A = p @ 0
app2 (A: U) (a b: A) (p: Path A a b): A = p @ 1
```

Definition 22. (Connections). Connections allows you to build square with given only one element of path: i)
$(i, j : I) \to p \ @ \ min(i, j)$; ii)
$(i, j : I) \to p \ @ \ max(i, j)$.




```
connection1 (A: U) (a b: A) (p: Path A a b)
          : PathP (<x> Path A (p@x) b) p (<i>b)
          = <y x> p @ (x \/ y)

connection2 (A: U) (a b: A) (p: Path A a b)
          : PathP (<x> Path A a (p@x)) (<i>a) p
          = <x y> p @ (x /\ y)
```

[1]https://arxiv.org/pdf/1611.02108.pdf, Cohen, Coquand, Huber, Mörtberg (2015)
[2]https://guillaumebrunerie.github.io/pdf/cart-cube.pdf, Angiuli, Brunerie, Coquand, Favonia, Harper, Licata (2017)
[3]http://www.cse.chalmers.se/ coquand/mod1.pdf, Bezem, Coquand, Huber (2014)
[4]https://www.cs.cmu.edu/čangiuli/papers/ccctt.pdf, Angiuli, Favonia, Harper (2018)
[5]https://arxiv.org/pdf/1712.04864.pdf, Pitts, Orton (2016)

Theorem 9. (Congruence). Is a map between values of one type to path space of another type by an encode function between types. Implemented as lambda defined on $[0, 1]$ that returns application of encode function to path application of the given path to lamda argument $\lambda(i : I) \to f(p@i)$ for both cases.

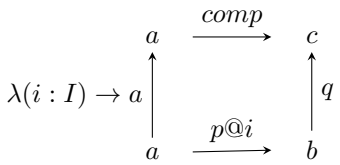
```
ap   (A B: U) (f: A -> B)
     (a b: A) (p: Path A a b)
  : Path B (f a) (f b)

apd (A: U) (a x:A) (B:A->U) (f: A->B a)
     (b: B a) (p: Path A a x)
  : Path (B a) (f a) (f x)
```

Theorem 10. (Transport). Transports a value of the domain type to the value of the codomain type by a given path element of the path space between domain and codomain types. Defined as path composition with [] of a over a path p — $comppa[]$.

```
trans (A B: U) (p: Path U A B) (a: A) : B
```

Definition 23. (Composition). Composition operation allows to build a new path by given to paths in a connected point. The proofterm is $comp(< i > PathAa(q@i))p[]$.




```
composition (A: U) (a b c: A)
            (p: Path A a b) (q: Path A b c) : Path A a c
```

Theorem 11. (Eliminator, Paulin-Mohring). J is formulated in a form of Paulin-Mohring and implemented using two facts that singleton are contractible and dependent function transport.

```
J (A: U) (a b: A)
  (P: singl A a -> U)
  (u: P (a,refl A a))
  (p: Path A a b) : P (b,p)
```

Theorem 12. (Eliminator, HoTT). J from HoTT book.

```
J (A: U) (a b: A)
  (C: (x: A) -> Path A a x -> U)
  (d: C a (refl A a))
  (p: Path A a b) : C b p
```

Theorem 13. (Eliminator, Diagonal).

```
D (A: U) : U = (x y: A) -> Path A x y -> U
J (A: U) (x y: A) (C: D A)
  (d: C x x (refl A x))
  (p: Path A x y) : C x y p
```

Theorem 14. (Computation).

```
trans_comp (A: U) (a: A)
  : Path A a (trans A A (<_> A) a)
subst_comp (A: U) (P: A -> U) (a: A) (e: P a)
  : Path (P a) e (subst A P a a (refl A a) e)
J_comp (A: U) (a: A) (C: (x: A) -> Path A a x -> U) (d: C a (refl A a))
  : Path (C a (refl A a)) d (J A a C d a (refl A a))
```

2 Homotopy Type Theory

2.1 Functional Extensionality

Definition 24. (Formation).

Theorem 15. (Introduction).

Theorem 16. (Elimination).

Theorem 17. (Computation).

Theorem 18. (Uniqueness).