

# Issue I: Internalizing Martin-Löf Type Theory

Maxim Sokhatsky

<sup>1</sup> National Technical University of Ukraine

Igor Sikorsky Kyiv Polytechnical Institute

August 22, 2018

## Abstract

Each language implementation needs to be checked. The one of possible test cases for cubical type checkers is the direct embedding of type theory model into the language of type checker. As types in Martin-Löf Type Theory are formulated using 5 rules, we construct aliases for host language primitives and use type checker to prove theorems about its core types.

This could be seen as ultimate test sample for type checker as introduction-elimination fusion resides in beta-eta rules, so by proving them we prove properties of the host type checker.

**Keywords:** Cubical Type Theory, Martin-Löf Type Theory

## Contents

<b>1</b>	<b>Martin-Löf Type Theory</b>	<b>3</b>
1.1	Pi . . . . .	3
1.2	Sigma . . . . .	6
1.3	Path . . . . .	8
1.4	MLTT . . . . .	11

## Intro

Each language implementation needs to be checked. The one of possible test cases for cubical type checkers is the direct embedding of type theory model into the language of type checker. As types in Martin-Löf Type Theory (MLTT) are formulated using 5 rules, we construct aliases for host language primitives and use type checker to prove that it is MLTT.

This could be seen as ultimate test sample for type checker as intro-elimination fusion resides in beta-eta rules, so by proving them we prove properties of the host type checker.

This technique of direct embedding of the model into the type checker primitives was also used to prove that Category of Sets is Cartesian Closed.

## Cubical Syntax

The BNF notation of `cubicaltt` 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 article we don't need ones.

# 1 Martin-Löf Type Theory

Martin-Löf Type Theory (MLTT) contains  $\Pi$ ,  $\Sigma$ ,  $\text{Id}$ ,  $W$ ,  $\text{Nat}$ ,  $\text{List}$  types. For simplicity we wouldn't take into account  $W$ ,  $\text{Nat}$ ,  $\text{List}$  types as  $W$  type could be encoded through  $\Sigma$  and  $\text{Nat}/\text{List}$  through  $W$ . Despite  $\Sigma$  types could be encoded through  $\Pi$  we include  $\Sigma$  type into the MLTT model.

The most interesting are  $\text{Id}$  types.  $\text{Id}$  types were added in <sup>1</sup>1984 while original MLTT was introduced in <sup>2</sup>1972. Predicative Universe Hierarchy was added in <sup>3</sup>1975. While original MLTT contains  $\text{Id}$  types that preserve uniqueness of identity proofs (UIP), we introduce here homotopical univalent heterogeneous Path interval types with higher equalities (<sup>4</sup> $\infty$ -Groupoid interpretation). Path types are essential to prove computation and uniqueness rules for all types, so we will be able to prove all the MLTT rules as a whole.

## 1.1 $\Pi$

$\Pi$  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: sets, functions, polynomial functors, infinitesimals,  $\infty$ -groupoids, topological  $\infty$ -groupoid, CW-complexes, categories, languages, etc. We give here nearest isomorphism of  $\Pi$ -types, the fibrations or fiber bundles. The next isomorphism of functions are functors. The most notable application of  $\Pi$  type in mathematics is the  $\forall$  quantifier as its synonym in higher order logic.

**Definition 1.** ( $\Pi$ -Formation).

$\Pi (A: U) (B: A \rightarrow U): U = (x: A) \rightarrow B\ x$

**Definition 2.** ( $\Pi$ -Introduction).

$\text{lambda } (A\ B: U) (b: B): A \rightarrow B = \lambda (x: A) \rightarrow b$   
 $\text{lam } (A:U) (B: A \rightarrow U) (a:A) (b:B\ a) : A \rightarrow B\ a = \lambda (x: A) \rightarrow b$

**Definition 3.** ( $\Pi$ -Elimination).

$\text{apply } (A\ B: U) (f: A \rightarrow B) (a: A) : B = f\ a$   
 $\text{app } (A: U) (B: A \rightarrow U) (a: A) (f: A \rightarrow B\ a) : B\ a = f\ a$

**Theorem 1.** ( $\Pi$ -Computation).

$\text{Beta } (A: U) (B: A \rightarrow U) (a: A) (f: A \rightarrow B\ a)$   
 $: \text{Path } (B\ a) (\text{app } A\ B\ a (\text{lam } A\ B\ a (f\ a))) (f\ a)$

**Theorem 2.** ( $\Pi$ -Uniqueness).

$\text{Eta } (A: U) (B: A \rightarrow U) (a: A) (f: A \rightarrow B\ a)$   
 $: \text{Path } (A \rightarrow B\ a) f (\lambda (x:A) \rightarrow f\ x)$

<sup>1</sup>P. Martin-Löf, G. Sambin. Intuitionistic type theory. 1984.

<sup>2</sup>P. Martin-Löf, G. Sambin. The Theory of Types. 1972.

<sup>3</sup>P. Martin-Löf. An intuitionistic theory of types: predicative part. 1975.

<sup>4</sup>M. Hofmann, T. Streicher. The groupoid interpretation of type theory. 1996.

## Examples from Mathematics

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

**Definition 5.** (Fiber). The fiber of the map  $p : E \rightarrow B$  in a point  $y : B$  is all points  $x : E$  such that  $p(x) = y$ .

**Definition 6.** (Fiber Bundle). The fiber bundle  $F \rightarrow E \xrightarrow{p} B$  on a total space  $E$  with fiber layer  $F$  and base  $B$  is a structure  $(F, E, p, B)$  where  $p : E \rightarrow B$  is a surjective map with following property: for any point  $y : B$  exists a neighborhood  $U_b$  for which a homeomorphism  $f : p^{-1}(U_b) \rightarrow U_b \times F$  making the following diagram commute.

$$\begin{array}{ccc} p^{-1}(U_b) & \xrightarrow{f} & U_b \times F \\ p \downarrow & \swarrow pr_1 & \\ U_b & & \end{array}$$

**Definition 7.** (Cartesian Product of Family over B). Is a set  $F$  of sections of the bundle with elimination map  $app : F \times B \rightarrow E$  such that

$$F \times B \xrightarrow{app} E \xrightarrow{pr_1} B \quad (1)$$

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

**Definition 8.** (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 9.** (Dependent Product). The dependent product along morphism  $g : B \rightarrow A$  in category  $C$  is the right adjoint  $\Pi_g : C_{/B} \rightarrow C_{/A}$  of the base change functor.

**Definition 10.** (Space of Sections). Let  $\mathbf{H}$  be a  $(\infty, 1)$ -topos, and let  $E \rightarrow B : \mathbf{H}_{/B}$  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 3.** (Functions Preserve Paths). For a function  $f : (x : A) \rightarrow B(x)$  there is an  $ap_f : x =_A y \rightarrow f(x) =_{B(x)} f(y)$ . This is called application of  $f$  to path or congruence property (for non-dependent case — *cong* function). This property behaves functorially as if paths are groupoid morphisms and types are objects.

**Theorem 4.** (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)

Note that we will not be able to prove this theorem until **Issue V: Many Faces of Equality** because bi-invertible iso type will be announced there.

**Theorem 5.** (Homotopy Equivalence). If fiber space is set for all base, and there are two functions  $f, g : (x : A) \rightarrow B(x)$  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 6.** (HomSet). If codomain is set then space of sections is a set.

setFun (A B : U) ( \_ : isSet B) : isSet (A → B)

**Theorem 7.** (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

$\Sigma$  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 11.** ( $\Sigma$ -Formation).

$\text{Sigma } (A : U) (B : A \rightarrow U) : U = (x : A) * B x$

**Definition 12.** ( $\Sigma$ -Introduction).

$\text{dpair } (A : U) (B : A \rightarrow U) (a : A) (b : B a) : \text{Sigma } A B = (a, b)$

**Definition 13.** ( $\Sigma$ -Elimination).

$\text{pr1 } (A : U) (B : A \rightarrow U)$   
 $(x : \text{Sigma } A B) : A = x.1$

$\text{pr2 } (A : U) (B : A \rightarrow U)$   
 $(x : \text{Sigma } A B) : B (\text{pr1 } A B x) = x.2$

$\text{sigInd } (A : U) (B : A \rightarrow U) (C : \text{Sigma } A B \rightarrow U)$   
 $(g : (a : A) (b : B a) \rightarrow C (a, b))$   
 $(p : \text{Sigma } A B) : C p = g p.1 p.2$

**Theorem 8.** ( $\Sigma$ -Computation).

$\text{Beta1 } (A : U) (B : A \rightarrow U)$   
 $(a : A) (b : B a)$   
 $: \text{Equ } A a (\text{pr1 } A B (a, b))$

$\text{Beta2 } (A : U) (B : A \rightarrow U)$   
 $(a : A) (b : B a)$   
 $: \text{Equ } (B a) b (\text{pr2 } A B (a, b))$

**Theorem 9.** ( $\Sigma$ -Uniqueness).

$\text{Eta2 } (A : U) (B : A \rightarrow U) (p : \text{Sigma } A B)$   
 $: \text{Equ } (\text{Sigma } A B) p (\text{pr1 } A B p, \text{pr2 } A B p)$

## Examples from Mathematics

**Definition 14.** (Dependent Sum). The dependent sum along the morphism  $f : A \rightarrow B$  in category  $C$  is the left adjoint  $\Sigma_f : C/A \rightarrow C/B$  of the base change functor.

**Theorem 10.** (Axiom of Choice). If for all  $x : A$  there is  $y : B$  such that  $R(x, y)$ , then there is a function  $f : A \rightarrow B$  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 11.** (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 12.** ( $\Sigma$ -Contractability). If the fiber is set then the  $\Sigma$  is set.

```
setSig (A:U) (B: A -> U) (sA: isSet A)
  (sB : (x:A) -> isSet (B x)) : isSet (Sigma A B)
```

**Theorem 13.** (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 <sup>5</sup>CCHM cubicaltt model with connections. For <sup>6</sup>ABCFHL yacctt model with variables please refer to ytt file. You may also want to read <sup>7</sup>BCH, <sup>8</sup>AFH. There is a <sup>9</sup>PO paper about CCHM axiomatic in a topos.

**Definition 15.** (Path Formation).

Hetero (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 16.** (Path Reflexivity). 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 which equals to  $\lambda (i : I) \rightarrow a$ .

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

**Definition 17.** (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 18.** (Path Composition). Composition operation allows to build a new path by given to paths in a connected point.

$$\begin{array}{ccc}
 & a & \xrightarrow{\text{comp}} c \\
 \lambda(i : I) \rightarrow a \uparrow & & \uparrow q \\
 a & \xrightarrow{p@i} & b
 \end{array}$$

composition (A: U) (a b c: A) (p: Path A a b) (q: Path A b c)  
 : Path A a c = comp (<i>Path A a (q@i)) p []

<sup>5</sup>Cyril Cohen, Thierry Coquand, Simon Huber, Anders Mörtberg. Cubical Type Theory: a constructive interpretation of the univalence axiom. 2015. <https://5ht.co/cubicaltt.pdf>

<sup>6</sup>Carlo Angiuli, Brunerie, Coquand, Kuen-Bang Hou (Favonia), Robert Harper, Dan Licata. Cartesian Cubical Type Theory. 2017. <https://5ht.co/ccctt.pdf>

<sup>7</sup>Marc Bezem, Thierry Coquand, Simon Huber. A model of type theory in cubical sets. 2014. <http://www.cse.chalmers.se/~coquand/mod1.pdf>

<sup>8</sup>Carlo Angiuli, Kuen-Bang Hou (Favonia), Robert Harper. Cartesian Cubical Computational Type Theory: Constructive Reasoning with Paths and Equalities. 2018. <https://www.cs.cmu.edu/~cangiuli/papers/ccctt.pdf>

<sup>9</sup>Andrew Pitts, Ian Orton. Axioms for Modelling Cubical Type Theory in a Topos. 2016. <https://arxiv.org/pdf/1712.04864.pdf>



**Theorem 14.** (Path Inversion).

$\text{inv } (A: U) \ (a \ b: A) \ (p: \text{Path } A \ a \ b): \text{Path } A \ b \ a = \langle i \rangle \ p \ @ \ -i$

**Definition 19.** (Connections). Connections allows you to build square with given only one element of path: i)  $\lambda \ (i, j: I) \rightarrow p \ @ \ \text{min}(i, j)$ ; ii)  $\lambda \ (i, j: I) \rightarrow p \ @ \ \text{max}(i, j)$ .

$$\begin{array}{ccc}
 & a & \xrightarrow{p} \quad b \\
 \lambda \ (i: I) \rightarrow a \uparrow & & \uparrow p \\
 & a & \xrightarrow{\quad} \quad a
 \end{array}
 \qquad
 \begin{array}{ccc}
 & b & \xrightarrow{\lambda \ (i: I) \rightarrow b} \quad b \\
 p \uparrow & & \uparrow \lambda \ (i: I) \rightarrow b \\
 & a & \xrightarrow{p} \quad b
 \end{array}$$

```

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)

```

**Theorem 15.** (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 lambda argument  $\lambda \ (i: I) \rightarrow f \ (p \ @ \ i)$  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 16.** (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  $p$  —  $\text{comp } p \ a \ []$ .

```

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

```

**Definition 20.** (Singleton).

$\text{singl } (A: U) (a: A): U = (x: A) * \text{Path } A \ a \ x$

**Theorem 17.** (Singleton Instance).

$\text{eta } (A: U) (a: A): \text{singl } A \ a = (a, \text{refl } A \ a)$

**Theorem 18.** (Singleton Contractability).

$\text{contr } (A: U) (a \ b: A) (p: \text{Path } A \ a \ b)$   
 $: \text{Path } (\text{singl } A \ a) (\text{eta } A \ a) (b, p)$   
 $= \langle i \rangle (p @ i, \langle j \rangle p @ i / \backslash j)$

**Theorem 19.** (Path Elimination, Diagonal).

$D (A: U) : U = (x \ y: A) \rightarrow \text{Path } A \ x \ y \rightarrow U$   
 $J (A: U) (x \ y: A) (C: D \ A)$   
 $(d: C \ x \ x (\text{refl } A \ x))$   
 $(p: \text{Path } A \ x \ y) : C \ x \ y \ p$   
 $= \text{subst } (\text{singl } A \ x) T (\text{eta } A \ x) (y, p) (\text{contr } A \ x \ y \ p) \ d \text{ where}$   
 $T (z: \text{singl } A \ x) : U = C \ x \ (z.1) (z.2)$

**Theorem 20.** (Path Elimination, Paulin-Mohring).  $J$  is formulated in a form of Paulin-Mohring and implemented using two facts that singleton are contractible and dependent function transport.

$J (A: U) (a \ b: A)$   
 $(P: \text{singl } A \ a \rightarrow U)$   
 $(u: P \ (a, \text{refl } A \ a))$   
 $(p: \text{Path } A \ a \ b) : P \ (b, p)$

**Theorem 21.** (Path Elimination, HoTT).  $J$  from HoTT book.

$J (A: U) (a \ b: A)$   
 $(C: (x: A) \rightarrow \text{Path } A \ a \ x \rightarrow U)$   
 $(d: C \ a (\text{refl } A \ a))$   
 $(p: \text{Path } A \ a \ b) : C \ b \ p$

**Theorem 22.** (Path Computation).

$\text{trans\_comp } (A: U) (a: A)$   
 $: \text{Path } A \ a (\text{trans } A \ A (\langle \_ \rangle A) \ a)$   
 $= \text{fill } (\langle i \rangle A) \ a \ []$   
 $\text{subst\_comp } (A: U) (P: A \rightarrow U) (a: A) (e: P \ a)$   
 $: \text{Path } (P \ a) \ e (\text{subst } A \ P \ a \ a (\text{refl } A \ a) \ e)$   
 $= \text{trans\_comp } (P \ a) \ e$   
 $\text{J\_comp } (A: U) (a: A) (C: (x: A) \rightarrow \text{Path } A \ a \ x \rightarrow U) (d: C \ a (\text{refl } A \ a))$   
 $: \text{Path } (C \ a (\text{refl } A \ a)) \ d (J \ A \ a \ C \ d \ a (\text{refl } A \ a))$   
 $= \text{subst\_comp } (\text{singl } A \ a) T (\text{eta } A \ a) \ d \text{ where } T (z: \text{singl } A \ a)$   
 $: U = C \ a (z.1) (z.2)$

Note that Path type has no Eta rule due to groupoid interpretation.

## 1.4 MLTT

Here we combine 4 Path rules (no eta), 5  $\Pi$  rules, and 6  $\Sigma$  rules (two elims).

**Definition 21.** (MLTT). The MLTT as a Type is defined by taking all rules for  $\Pi$ ,  $\Sigma$  and Path types into one  $\Sigma$  telescope or context.

```
MLTT (A: U): U
= (Pi_Former: (A -> U) -> U)
  * (Pi_Intro: (B: A -> U) (a: A) -> B a -> (A -> B a))
  * (Pi_Elim: (B: A -> U) (a: A) -> (A -> B a) -> B a)
  * (Pi_Comp1: (B: A -> U) (a: A) (f: A -> B a) ->
    Path (B a) (Pi_Elim B a (Pi_Intro B a (f a))) (f a))
  * (Pi_Comp2: (B: A -> U) (a: A) (f: A -> B a) ->
    Path (A -> B a) f (\(x:A) -> f x))
  * (Sigma_Former: (A -> U) -> U)
  * (Sigma_Intro: (B: A -> U) (a: A) -> (b: B a) -> Sigma A B)
  * (Sigma_Elim1: (B: A -> U) (._: Sigma A B) -> A)
  * (Sigma_Elim2: (B: A -> U) (x: Sigma A B) -> B (pr1 A B x))
  * (Sigma_Comp1: (B: A -> U) (a: A) (b: B a) ->
    Path A a (Sigma_Elim1 B (Sigma_Intro B a b)))
  * (Sigma_Comp2: (B: A -> U) (a: A) (b: B a) ->
    Path (B a) b (Sigma_Elim2 B (a,b)))
  * (Sigma_Comp3: (B: A -> U) (p: Sigma A B) ->
    Path (Sigma A B) p (pr1 A B p, pr2 A B p))
  * (Id_Former: A -> A -> U)
  * (Id_Intro: (a: A) -> Path A a a)
  * (Id_Elim: (x: A) (C: D A) (d: C x x (Id_Intro x))
    (y: A) (p: Path A x y) -> C x y p)
  * (Id_Comp: (a:A)(C: D A) (d: C a a (Id_Intro a)) ->
    Path (C a a (Id_Intro a)) d (Id_Elim a C d a (Id_Intro a))) * U
```

**Theorem 23.** (Model Check). There is an instance of MLTT.

```
instance (A: U): MLTT A
= (Pi A, lam A, app A, Beta A, Eta A,
   Sigma A, dpair A, pr1 A, pr2 A, Beta1 A, Beta2 A, Eta2 A,
   Path A, refl A, J A, J_comp A, A)
```

## Cubical Model Check

The result of the work is a `mltt.ctt` file which can be runned using `cubicaltt`. Note that computation rules take a seconds to type check.

```
$ time cubical -b mltt.ctt
```

```
Checking: MLTT
```

```
Checking: instance
```

```
File loaded.
```

```
real    0m6.308s
```

```
user    0m6.278s
```

```
sys     0m0.014s
```

## Conclusion

This article opens door to a series that will unveil the topics of homotopy type theory with practical emphasis to cubical type checkers. The article names are subject to change and are based on course structure. A number of articles could be issued under the same chapter number.

## Foundations

The Foundations volume of articles define formal programming language with geometric foundations and show how to prove properties of such constructions. The foundations contain only programming system overview disregarding specific mathematical models or theories which will be given in the second volume entitled Mathematics.

**Issue I: Internalizing Martin-Löf Type Theory.** The first volume of definitions gathered into one article dedicated to various  $\prod$  and  $\sum$  properties and internalization of MLTT in the host language typechecker.

**Issue II: Inductive Types and Encodings.** This episode tells a story of inductive types, their encodings, induction principle and its models.

**Issue III: Homotopy Type Theory.** This issue is try to present the Homotopy Type Theory without higher inductive types to neglect the core and principles of homotopical proofs.

**Issue IV: Higher Inductive Types.** The metamodel of HIT is a theory of CW-complexes. The category of HIT is a homotopy category. This volume finalizes the building of the computational theory.

**Issue V: Many Faces of Equality.** This article pay attention to different forms of equalities and kick the tower of higher equalities.

**Issue VI: Modalities.** What if something couldn't be constructively presented? We can wrap this into modalities and interface it with 5 types of MLTT rules, making system sound but without computational semantics.

## Mathematics

The second volume of article is dedicated to cover the mathematical programming and modeling.

**Issue VII: Set Theory.** The set theory and mere propositions: set, prop.

**Issue VIII: Category Theory.** The model of Category Theory definitions. It includes: cat, adj, cones, fun, category, sip, ump, cwf.

**Issue IX: Formal Topos on Category of Sets.** Formal packaging of set theory in a topos.

**Issue X: Abstract Algebra.** Abstract algebra, such as Monoid, Group, Semigroup, Monad, etc: algebra, control.

**Issue XI: Ordinals.** Modules: ordinal, mahlo.

**Issue XII: Differential Geometry.** Modules: etale, infinitesimal, manifold, shape.

**Issue XIII: Hopf Fibrations.** Modules: pointed, euler, hopf.

**Issue XIV: K-Theory.** Modules: k.theory, spawntrans, subtype, bishop.

## **Addons**

A number of application will be issued during this series. At the time of first volume only one appendix is available, the PTS language with infinite number of universes with switchable SAR rules.

**Addon I: Pure Type System for Erlang.**