Issue I: Internalizing Martin-Löf Type Theory

Maxim Sokhatsky

[1] National Technical University of Ukraine
Igor Sikorsky Kyiv Polytechnical Institute
September 21, 2018

Abstract

Each language implementation needs to be checked. The one of possible test cases for 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 by using 5 types of rules, we construct aliases for host language primitives and use type checker to prove theorems about its core types by using only equality reflection.

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. As a prerequisite for this exercise and a bonus we give along the way some interpretations of Π, Σ and Path types from other areas of mathematics.

Keywords: Martin-Löf Type Theory, Cubical Type Theory

Contents

Intro

Each language implementation needs to be checked. The one of possible test cases for 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 types of rules (formation, introduction, elimination, computation, uniqueness), 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.

Also this issue opens a series of articles dedicated to formalization in cubical type theory the foundations of mathematics. This issue is dedicated to MLTT modeling and its verification. Also as many may not be familiar with Π and Σ types, this issue presents different interpretation of MLTT types.

Language Syntax

The BNF notation of type checker language used in code samples consists of: i) telescopes (contexts or sigma chains) and definitions; ii) pure dependent type theory syntax; iii) inductive data definitions (sum chains) and split eliminator; iv) cubical face system; v) module system. It is slightly based on **cubicaltt**[1].

```
   sys := [ coside ]          side := (id=0)→exp+(id=1)→exp
  form := form\/f1+f1+f2    coside := #empty+sides
 sides := side+side+coside     mod := module id where imps dec
    f1 := f1/\f2                f2 := -f2+id+0+1
   imp := import id            brs := #empty+cobrs
   app := exp exp              tel := #empty+cotel
  imps := #list imp          cotel := (exp:exp) tel
    id := #list #nat           dec := #empty+codec
    u2 := glue+unglue+Glue      u1 := fill+comp
   ids := #list id              br := ids→exp+ids@ids→exp
 codec := def dec
 cobrs := | br brs
   sum := #empty+id tel+id tel|sum+id tel<ids>sys
   def := data id tel=sum+id tel:exp=exp+id tel:exp where def
   exp := cotel*exp+cotel→exp+exp→exp+(exp)+id
          (exp,exp)+\cotele→exp+split cobrs+exp.1+exp.2+
          ⟨ids⟩exp+exp@form+app+u2 exp exp sys+u1 exp sys
```

Here := (definition), + (disjoint sum), #empty, #nat, #list are parts of BNF language and |, :, *, ⟨, ⟩, (,), =, \, /, -, →, 0, 1, @, [,], **module**, **import**, **data**, **split**, **where**, **comp**, **fill**, **Glue**, **glue**, **unglue**, **.1**, **.2**, and , are terminals of type checker language. This language includes inductive types, higher inductive types and gluening operations needed for both, the constructive homotopy type theory and univalence. All these concepts as a part of the languages will be described in the upcoming Issues II—V.

[1] http://github.com/mortberg/cubicaltt

1 Martin-Löf Type Theory

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

Any new type in MLTT presented with set of 5 rules: i) formation rules, the signature of type; ii) the set of constructors which produce the elements of formation rule signature; iii) the dependent eliminator or induction principle for this type; iv) the beta-equality or computational rule; v) the eta-equality or uniquness principle. Π, Σ, and Path types will be given shortly. This interpretation or rather way of modeling is MLTT specific.

The most interesting are Id types. Id types were added in [2]1984 while original MLTT was introduced in [3]1972. Predicative Universe Hierarchy was added in [4]1975. While original MLTT contains Id types that preserve uniquness of identity proofs (UIP) or eta-rule of Id type, HoTT refutes UIP (eta rule desn't hold) and introduces univalent heterogeneous Path equality ([5]∞-Groupoid interpretation). Path types are essential to prove computation and uniquness rules for all types (needed for building signature and terms), so we will be able to prove all the MLTT rules as a whole.

1.1 Interpretations

MLTT	Proof Theory	Set Theory	Homotopy Theory
A	proposition	set	space
$a : A$	proof	element	point
$B(x)$	predicate	family of sets	fibration
$b(x) : B(x)$	conditional proof	family of elements	section
$\emptyset, \mathbf{1}$	$\bot, \top$	$\emptyset, \{\emptyset\}$	$\emptyset, *$
$A + B$	$A \vee B$	disjoint union	coproduct
$A \times B$	$A \wedge B$	set of pairs	product space
$A \to B$	$A \Rightarrow B$	set of functions	function space
$\sum x : A, B(x)$	$\exists_{x:A} B(x)$	disjoint sum	total space
$\prod x : A, B(x)$	$\forall_{x:A} B(x)$	product	space of sections
$\mathbf{Id}_A$	equality $=$	$\{< x, x > \| x \in A\}$	path space A^I

Table 1: Comparing points of view on type-theoretic operations

1.2 Contexts

Definition 1. (Empty Context).

[2]P. Martin-Löf, G. Sambin. Intuitionistic type theory. 1984.
[3]P. Martin-Löf, G. Sambin. The Theory of Types. 1972.
[4]P. Martin-Löf. An intuitionistic theory of types: predicative part. 1975.
[5]M. Hofmann, T. Streicher. The groupoid interpretation of type theory. 1996.

Definition 2. (Context Comprehension).

Definition 3. (Context Derivability).

1.3 Universes

This introduction is a bit wild stives to be simple yet precise. As we defined a language BNF we could define a language AST by using inductive types which is yet to be defined in **Issue II: Inductive Types and Models**.

Definition 4. (Terms). Point in initial object of language AST inductive definition is called a term. If type theory or language is defined as an inductive type (AST) then the term is defined as its instance.

Definition 5. (Sorts). N-indexed set of universes $\mathrm{U}_{n\in\mathrm{N}}$. Could have any number of elements which defines different type systems. All built-in types as long as user defined types are landed usually by default in U_0 universe. Sorts represented in type checker as a separate constructor.

Definition 6. (Axioms). The inclusion rules $\mathrm{U_i} : \mathrm{U_j}, i, j \in \mathrm{N}$, that define which universe is element of another given universe. You may attach any rules that joins i, j in some way. Axioms with sorts define universe hierarchy.

Definition 7. (Rules). The set of landings $\mathrm{U_i} \to \mathrm{U_j} : \mathrm{U}_{\lambda(\mathrm{i,j}),\mathrm{i,j}\in\mathrm{N}}$, where $\lambda : \mathrm{N} \times \mathrm{N} \to \mathrm{N}$. These rules define term dependence or how we land (in which universe) formation rules in definitions.

Definition 8. (Predicative hierarchy). If λ in Rules is an uncurried function $\max : \mathrm{N} \times \mathrm{N} \to \mathrm{N}$ then such universe hierarchy is called predicative.

Definition 9. (Impredicative hierarchy). If λ in Rules is a second projection of a tuple $\mathrm{snd} : \mathrm{N} \times \mathrm{N} \to \mathrm{N}$ then such universe hierarchy is called impredicative.

Definition 10. (Definitional Equality). For any $\mathrm{U}_i, i \in \mathrm{N}$ there is defined an equality between its members and between its instances. For all x,y ∈ A, there is defined a x=y. Definitional equality compares normalized term instances.

Definition 11. (SAR). The universum space is configured with a triple of: i) sorts, a set of universes $\mathrm{U}_{n\in\mathrm{N}}$ indexed over set N; ii) axioms, a set of inclusions $\mathrm{U_i} : \mathrm{U_j}, i, j \in \mathrm{N}$; iii) rules of term dependence universe landing, a set of landings $\mathrm{U_i} \to \mathrm{U_j} : \mathrm{U}_{\lambda(\mathrm{i,j}),\mathrm{i,j}\in\mathrm{N}}$, where λ could be function *max* (predicative) or *snd* (impredicative).

Example 1. (CoC). $\mathrm{SAR} = \{\{\star, \Box\}, \{\star : \Box\}, \{i \to j : j; i, j \in \{\star, \Box\}\}$. Terms live in universe $\star$, and types live in universe $\Box$. In CoC $\lambda = \mathrm{snd}$.

Example 2. (PTS^∞). $\mathrm{SAR} = \{\mathrm{U}_{i\in\mathrm{N}}, \mathrm{U_i} : \mathrm{U}_{\mathrm{j;i<j;i,j}\in\mathrm{N}}, \mathrm{U_i} \to \mathrm{U_j} : \mathrm{U}_{\lambda(\mathrm{i,j});\mathrm{i,j}\in\mathrm{N}}\}$. Where U_i is a universe of i-level or i-category in categorical interpretation. The working prototype of PTS^∞ is given in **Addon I: Pure Type System for Erlang**[6].

[6] M.Sokhatsky,P.Maslianko. The Systems Engineering of Consistent Pure Language with Effect Type System for Certified Applications and Higher Languages. AIP Conference Proceedings. 2018. doi:10.1063/1.5045439

1.4 Π interpretations

Π is a dependent function type, the generalization of functions. As a function it can serve the wide range of mathematical constructions as its domain and codomain, which are in general: objects, types, or spaces; and could have as its instance: sets, functions, polynomial functors, infinitesimals, ∞-groupoids, topological ∞-groupoid, CW-complexes, categories, languages, etc.

At this light there could be many interpretation of Π types from different areas of mathematics. We give here three: i) logical interpretation of Π as ∀ quantifier from higher order logic that forms a ground of type theory; ii) geomeric intepretation of Π as fiber bundle; iii) categorical interpretation of functions as functors.

1.4.1 Type-theoretical

Definition 12. (Π-Formation).

$$(x : A) \to B(x) =_{def} \prod_{x:A} B(x) : U.$$

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

Definition 13. (Π-Introduction).

$$\backslash(x : A) \to b =_{def} \prod_{A:U} \prod_{B:A\to U} \prod_{a:A} \prod_{b:B(a)} \lambda x.b : \prod_{y:A} B(a).$$

```
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 14. (Π-Elimination).

$$f\ a =_{def} \prod_{A:U} \prod_{B:A\to U} \prod_{a:A} \prod_{f:\prod_{x:A} B(a)} f(a) : B(a).$$

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

Theorem 1. (Π-Computation).

$$f(a) =_{B(a)} (\lambda(x : A) \to f(a))(a).$$

```
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)
```

Theorem 2. (Π-Uniqueness).

$$f =_{(x:A)\to B(a)} (\lambda(y : A) \to f(y)).$$

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

1.4.2 Categorical

Definition 15. (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 16. (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 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)
```

Definition 17. (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.

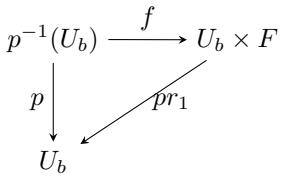
1.4.3 Fibrational (geometric)

The adjoints Π and Σ 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, Π 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. Π type also represents the cartesian family of sets, generalizing the cartesian product of sets.

Definition 18. (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 19. (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 20. (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 morphisms 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 21. (Trivial Fiber Bundle). When total space E is cartesian product $\Sigma(B, F)$ and $p = pr_1$ then such bundle is called trivial $(F, \Sigma(B, F), pr_1, B)$.

Theorem 6. (Functions Preserve Paths). For a function $f : (x : A) \to B(x)$ there is an $ap_f : x =_A y \to f(x) =_{B(x)} f(y)$. This is called application of f to path or congruence property (for non-dependent case — *cong* function). This property behaves functoriality as if paths are groupoid morphisms and types are objects.

Theorem 7. (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.

1.5 Σ interpretations

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

1.5.1 Type-theoretical

Definition 22. (Σ-Formation).

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

Definition 23. (Σ-Introduction).

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

Definition 24. (Σ-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
```

Theorem 8. (Σ-Computation).

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

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

Theorem 9. (Σ-Uniqueness).

```
Eta2 (A: U) (B: A -> U) (p: Sigma A B)
   : Equ (Sigma A B) p (pr1 A B p,pr2 A B p)
```

1.5.2 Categorical

Definition 25. (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.

1.5.3 Set-theoretical

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

1.6 Path interpretations

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 [7]CCHM cubicaltt model with connections. For [8]ABCFHL yacctt model with variables please refer to ytt file. You may also want to read [9]BCH, [10]AFH. There is a [11]PO paper about CCHM axiomatic in a topos.

1.6.1 Cubical

Definition 26. (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 27. (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 $\lambda\ (i : I) \to a$.

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

Definition 28. (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
```

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

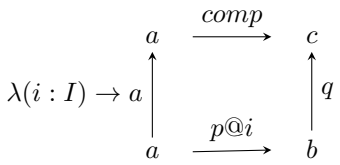
[8]Carlo Angiuli, Brunerie, Coquand, Kuen-Bang Hou (Favonia), Robert Harper, Dan Licata. Cartesian Cubical Type Theory. 2017. https://5ht.co/cctt.pdf

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

[10]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

[11]Andrew Pitts, Ian Orton. Axioms for Modelling Cubical Type Theory in a Topos. 2016. https://arxiv.org/pdf/1712.04864.pdf

Definition 29. (Path Composition). Composition operation allows to build a new path by given to paths in a connected point.

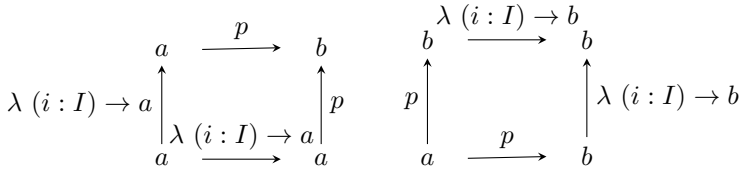



```
composition (A: U) (a b c: A) (p: Path A a b) (q: Path A b c)
          : Path A a c = comp (<i>Path A a (q@i)) p []
```

Theorem 14. (Path Inversion).

```
inv (A: U) (a b: A) (p: Path A a b): Path A b a = <i> p @ -i
```

Definition 30. (Connections). Connections allows you to build square with given only one element of path: i) $\lambda\ (i, j : I) \to p\ @\ min(i, j)$; ii) $\lambda\ (i, j : I) \to p\ @\ max(i, j)$.




```
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)
```

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 lamda argument `λ (i : I) → 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 — comp p a [].

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

1.6.2 Type-theoretical

Definition 31. (Singleton).

```
singl (A: U) (a: A): U = (x: A) * Path A a x
```

Theorem 17. (Singleton Instance).

```
eta (A: U) (a: A): singl A a = (a, refl A a)
```

Theorem 18. (Singleton Contractability).

```
contr (A: U) (a b: A) (p: Path A a b)
  : Path (singl A a) (eta A a) (b,p)
  = <i> (p @ i,<j> p @ i/\j)
```

Theorem 19. (Path Elimination, 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
= subst (singl A x) T (eta A x) (y, p) (contr A x y p) d where
  T (z: 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: singl A a -> U)
  (u: P (a, refl A a))
  (p: 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) -> Path A a x -> U)
  (d: C a (refl A a))
  (p: Path A a b) : C b p
```

Theorem 22. (Path Computation).

```
trans_comp (A: U) (a: A)
  : Path A a (trans A A (<_> A) a)
  = fill (<i> 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)
  = trans_comp (P 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))
  = subst_comp (singl A a) T (eta A a) d where T (z: singl A a)
  : U = C a (z.1) (z.2)
```

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

1.6.3 Groupoid (categorical)

The groupoid interpretation of type theory is well known article by Martin Hoffman and Thomas Streicher, more specific interpretation of identity type as infinity groupoid. The groupoid interpretation of Path equality will be given along with category theory library in **Issue VIII: Category Theory**.

1.7 MLTT

Here we combine 4 Path rules (no eta), 5 Π rules, and 6 Σ rules (two elims).

Definition 32. (MLTT). The MLTT as a Type is defined by taking all rules for Π, Σ and Path types into one Σ 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
```

1.8 Exercises

Exercise 1. (Composition). Define composition of functions $A \to B$, functors $U \to U$ and composition operation for sigma types $A \times B \to B \times C \to A \times C$. Also write a generator of composition signature $(A \to B) \to (B \to C) \to (A \to C)$.

Exercise 2. (Constants). Define constant type and identity function.

Exercise 3. (Categorical Laws). Show that any function of $\prod$-type equals its left and right composition with identity function. Prove associativity of composition.

Exercise 4. (Swap). Define swap function

$$\prod_{x:A} \prod_{y:A} C(x,y) \to \prod_{y:A} \prod_{x:A} C(x,y)$$

Exercise 5. (Curry, Uncurry). Define curry and uncury functions.

Exercise 6. (Sigma). Define (by definition here and below we mean all 5 rules of MLTT) $\sum$-type by using only $\prod$-type.

Exercise 7. (Fin). Define the **Fin**-type by using only $\sum$-type and recursion. Define function that returns max element of **Fin**-set.

Exercise 8. (W-types). Define **W**-type by using only $\sum$-type.

Exercise 9. (Nat). Define **Nat**-type as **W**-type. Also definee a **Nat** algebra: multiplication, power, factorial by using $\mathbf{rec_{Nat}}$.

Exercise 10. (List). Define **List**-type as **W**-type.

Exercise 11. (Ack). Define Ackermann function by using only $\mathbf{rec_{Nat}}$.

Exercise 12. (Eliminators Extensionality). After **Issue V: Many Faces of Equality**. Prove that three **J** eliminators are equal each other.

Conclusion

This article opens door to a series that will unvail 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: Intenalizing 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 tales 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 attension 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.

The main intention of Foundation volume is to show the internal language of working topos of CW-complexes in fibrational sheaf type theory.

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: Topos Theory. Formal packaging of set theory in a topos.

Issue X: Differential Geometry. Modules: etale, infinitesimal, manifold, shape.

Issue XI: Hopf Fibrations. Modules: pointed, euler, hopf.

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

Issue XIII: K-Theory. Modules: k_theory, spawptrans, 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 languge with infinite number of universes with switcheable SAR rules.

Addon I: Pure Type System for Erlang.