M.E. Sokhatsky*

Issue I: Internalizing Martin-Löf Type Theory

Igor Sikorsky Kyiv Polytechnical Institute, Kyiv, Ukraine
*Corresponding author: maxim@synrc.com

This article demonstrates formal MLTT embedding into the host type system with constructive proofs of all inference rules. This was recently made possible by introducing cubical type theory in 2017. The internalization itself is a great tool for testing the type checkers. 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 type core.

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

Introduction

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 [1, 2] (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.

This test is fully made possible only after 2017 when new constructive **cubicaltt**[1] HoTT [3] prover was presented [4].

We should note that this is only entrance to internalization technique, and after formal MLTT embedding we need to go further through CiC [5, 6] and towards CW-complexes embedding as the higher inductive type system. This is yet an open problem.

Problem Statement

The problem is simple: create full self-contained MLTT internalization in the host typechecker, where all theorems are being checked constructively.

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.

```
  sys := [ sides ]              side := (id=0)→exp+(id=1)→exp
 form := form\/f1+f1+f2        sides := #empty+cos+side
  cos := side,side+side,cos     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
```

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

```
    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 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] while original MLTT was introduced in [1]. Predicative Universe Hierarchy was added in [7]. 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 ([8]). 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

In contexts you can bind to variables (through de Brujin indexes or string names): i) indexed universes; ii) built-in types; iii) user constructed types, and ask questions about type derivability, type checking and code extraction. This system defines the core type checker within its language.
By using this languages it is possible to encode different interpretations of type theory itself and its syntax by construction. Usually the issues will refer to following interpretations: i) type-theoretical; ii) categorical; iii) set-theoretical; iv) homotopical; v) fibrational or geometrical.

1.1.1 Logical or Type-theoretical interpretation

According to type theoretical interpretation of MLTT for any type should be provided 5 formal inference rules: i) formation; ii) introduction; iii) dependent elimination principle; iv) beta rule or computational rule; v) eta rule or uniqueness rule. The last one could be exceptional for Path types. The formal representation of all rules of MLTT are given according to type-theoretical interpretation as a final result in this Issue I. It was proven that classical Logic could be embedded into intuitionistic propositional logic (IPL) which is directly embedded into MLTT.
Logical and type-theoretical interpretations could be distincted. Also set-theoretical interpretation is not presented in the Table.

1.1.2 Categorical or Topos-theoretical interpretation

Categorical interpretation [9] is a modeling through categories and functors. First category is defined as objects, morphisms and their properties, then we define functors, etc. In particular, as an example, according to categorical interpretation Π and Σ types of MLTT are presented as adjoint functors, and forms itself a locally closed cartesian category, which will be given a intermediate result in **Issue VIII: Topos Theory**. In some sense we include

Table. Interpretations correspond to mathematical theories

Type Theory	Logic	Category Theory	Homotopy Theory
A type	class	object	space
isProp A	proposition	(-1)-truncated object	space
a:A program	proof	generalized element	point
$B(x)$	predicate	indexed object	fibration
$b(x) : B(x)$	conditional proof	indexed elements	section
$\emptyset$	$\bot$ false	initial object	empty space
1	$\top$ true	terminal object	singleton
$A + B$	$A \vee B$ disjunction	coproduct	coproduct space
$A \times B$	$A \wedge B$ conjunction	product	product space
$A \to B$	$A \Rightarrow B$	internal hom	function space
$\sum x : A, B(x)$	$\exists_{x:A} B(x)$	dependent sum	total space
$\prod x : A, B(x)$	$\forall_{x:A} B(x)$	dependent product	space of sections
$\mathbf{Path}_A$	equivalence $=_A$	path space object	path space A^I
quotient	equivalence class	quotient	quotient
W-type	induction	colimit	complex
type of types	universe	object classifier	universe
quantum circuit	proof net	string diagram	

here topos-theoretical interpretations, with presheaf model of type theory as example (in this case fibrations are constructes as functors, categorically).

1.1.3 Homotopical interpretation

In classical MLTT uniquness rule of Id type do holds strictly. In Homotopical interpretation of MLTT we need to allow a path space as Path type where uniqueness rule doesn't hold. Groupoid interpretation of Path equality that doesn't hold UIP generally was given in 1996 by Martin Hofmann and Thomas Streicher [8].
When objects are defined as fibrations, or dependent products, or indexed-objects this leds to fibrational semantics and geometric sheaf interpretation. Several definition of fiber bundles and trivial fiber bindle as direct isomorphisms of Π types is given here as theorem. As fibrations study in homotopical interpretation, geometric interpretation could be treated as homotopical.

1.1.4 Set-theoretical interpretation

Set-theoretical interpretations could replace first-order logic, but could not allow higher equalities, as long as inductive types to be embedded directly. Set is modelled in type theory according to homotopical interpretation as n-type.

1.2 Types

MLTT could be reduced to Π, Σ, Path types, as W-types could be modeled through Σ and Fin/Nat/List/Maybe types could be modeled on W. In this issue Π, Σ, Path are given as a core MLTT and W-types are given as exercise. List, Nat, Fin types are defined in next **Issue II: Inductive Types**.

1.2.1 Π-type

Π is a dependent product 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.

Type-theoretical interpretation

As a logical system dependent type theory could correspond to higher order logic. However here only type-theoretical model is given completely.

Definition 1. (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 2. (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 3. (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)
```

Categorical interpretation

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.

Definition 4. (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 5. (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. (HomSet). If codomain is set then space of sections is a set.

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

Theorem 4. (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)
```

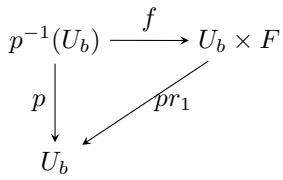
Definition 6. (Section). A section of morphism $f : A \to B$ in some category is the morphism $g : B \to A$ such that $f \circ g : B \xrightarrow{g} A \xrightarrow{f} B$ equals the identity morphism on B.

Homotopical interpretation

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

Theorem 5. (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 6. (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 7. (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
```

Note that we will not be able to prove this theorem until **Issue III: Homotopy Type Theory** because bi-invertible iso type will be announced there.

1.2.2 Σ-type

Σ is a dependent sum 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.

Type-theoretical interpretation

Definition 11. (Σ-Formation).

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

Definition 12. (Σ-Introduction).

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

Definition 13. (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)
```

Categorical interpretation

Definition 14. (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.

Set-theoretical interpretation

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)
```

1.2.3 Path-type

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 [2]CCHM cubicaltt model with connections. For [3]ABCFHL yacctt model with variables please refer to ytt file. You may also want to read [4]BCH, [5]AFH. There is a [6]PO paper about CCHM axiomatic in a topos.

Cubical interpretation

Cubical interpretation was first given by Simon Huber [10] and later was written first constructive type checker in the world by Anders Mörtberg [4].

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
```

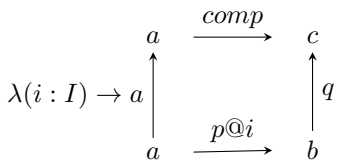
Definition 16. (Path Reflexivity). Returns an element of reflexivity path space for a given value of the type. The inhabitant of that path space is the lambda on the homotopy interval $[0, 1]$ that returns a constant value a. Written in syntax as <i>a which equals to $\lambda\ (i : I) \to a$.

```
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.

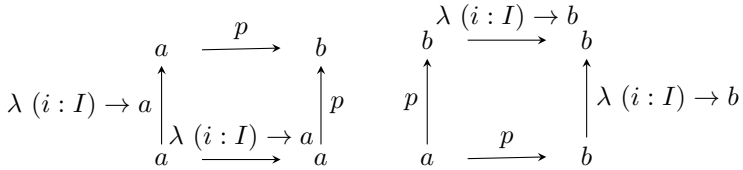



```
composition (A: U) (a b c: A) (p: Path A a b) (q: Path A b c)
          : Path A a c = comp (<i>Path A a (q@i)) p []
```

Theorem 14. (Path Inversion).

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

Definition 19. (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) $\lambda\ (i, j : I) \to p\ @\ max(i, j)$.



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

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

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

[5]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

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

```
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
```

Type-theoretical interpretation

Definition 20. (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.

Groupoid interpretation

The groupoid interpretation of type theory is well known article by Martin Hofmann 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 VII: Category Theory**.

1.3 Universes

This introduction is a bit wild strives 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**. This SAR notation is due Barendregt.

Definition 21. (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 22. (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 23. (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 24. (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 25. (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 26. (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 27. (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 28. (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). 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^∞). 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 **Addendum I: Pure Type System for Erlang** [11].

1.4 Contexts

Speaking of type checker execution, we introduce context or dictionary with types and terms, from which we can derive typed variables. This chain could be implemented as nested sigma types (due to R.A.G.Seely) or list types (due to Voevodsky). Categorically dependent type theory is built upon categories of contexts.

Definition 29. (Empty Context).

$$\gamma_0 : \Gamma =_{def} \star.$$

Definition 30. (Context Comprehension).

$$\Gamma\ ; A =_{def} \sum_{\gamma:\Gamma} A(\gamma).$$

Definition 31. (Context Derivability).

$$\Gamma \vdash A =_{def} \prod_{\gamma:\Gamma} A(\gamma).$$

1.5 MLTT

Here is given formal model of type-theoretical interpretation of Martin-Löf Type Theory. It combines 4 Path rules (no eta), 5 Π rules, and 6 Σ rules (two elims). The proof is provided by direct embedding (internalizing) the model intro the model of type checker which is even more powerful.

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
```

1.6 Exercises

In order to learn cubicaltt language you can use following exercises to improve your cubical skills.

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.

Conclusions

In this issue the type-theoretical model (interpretation) of MLTT was presented in cubical syntax and type checked in cubicaltt. This is the first constructive proof of internalization of MLTT. As a bonus the landspace of possible interpretation was shown corresponding different mathematical theories for those who are new to type theory. The brief description of the previous attempts to internalize MLTT could be found as canonical example in MLTT works, but none of them give the constructive J eliminator or its equality rule. As a selected prover for the article wa chosen cubicaltt but this excersise was implemented on all current cubical type checkers[7]: Arend[8], Agda[9],

[7] https://cubical.systems
[8] https://github.com/groupoid/arend
[9] https://github.com/groupoid/agda

cubicaltt[10], yacctt, redtt, RedPRL, Lean[11]. Type theoretical cubicaltt constructions was given along the article for other interpretations, all of them were taken from our Groupoing Infinity[12] base library.

Further Research

This article opens the door to a series that will unvail the different 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. The custom interpretations are to be issued under the volume of addendums. The volumes are: Foundations, Mathematics, Addendums.

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. This issue covers core modules: pi, sigma, path, mltt.
Issue II: Inductive Types and Encodings. This episode tales a story of inductive types, their encodings, induction principle and its models. This issue covers inductive base library: proto, bool, maybe, nat, list, int, stream, control, recursion.
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. This issue covers following modules: pullback, equiv, iso, retract.
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. This issue covers higher inductive base library: s1, s2, helix, trunc, quotient.
Issue V: 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, the construction of fibrational sheaf type theory.

Mathematics

The second volume of article is dedicated to cover the mathematical programming and modeling.
Issue VI: Set Theory. The set theory and mere propositions: set, prop.
Issue VII: Category Theory. The model of Category Theory definitions. It includes: cat, adj, cones, fun, category, sip, ump, cwf.
Issue VIII: Topos Theory. Formal packaging of set theory in a topos. Formal Topos and Formal Sheaf. It also includes sheaf embedding of type theory in type theory in sole module: topos.
Issue IX: Algebraic Topology. This branch of study of topological spaces with abstract algebra includes followin areas: Homotopy Theory, Homological Algebra, Complexes, This issue covers following modules: pointed, algebra, euler, hopf, seq, homology. cw.
Issue X: Differential Geometry. This branch of study includes infinitesimal constructions and Cartan geometry, the chapter is slightly base on Felix Wellen dissertation. This issue covers following modules: etale, infinitesimal, manifold, bundle.

Addendums

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.
Addendum I: Pure Type System for Erlang. This article overviews Erlang implementation of Om type checker with PTS$^\infty$ custom type system. The inductive base library in Boehm-Berarducci-Church encoding is given in the Github repository[13].

[10] https://github.com/groupoid/cubical
[11] https://github.com/groupoid/lean
[12] https://groupoid.space/mltt/types/
[13] `https://github.com/groupoid/pts`

Addendum II: Many Faces of Equality. This article pays attension to different forms of equalities and builds the tower of higher equalities.

Referenses

[1] P. Martin-Löf and G. Sambin, "The theory of types," in *Studies in proof theory*, 1972.

[2] P. Martin-Löf and G. Sambin, "Intuitionistic type theory," in *Studies in proof theory*, 1984.

[3] V. Voevodsky *et al.*, "Homotopy type theory," in *Univalent Foundations of Mathematics*, 2013.

[4] T. C. A. Mörtberg *et al.*, "Cubical type theory: a constructive interpretation of the univalence axiom," arXiv:1611.02108, 2017.

[5] F. Pfenning and C. Paulin-Mohring, "Inductively defined types in the calculus of constructions," in *Proc. 5th Int. Conf. Mathematical Foundations of Programming Semantics, Tulane University, New Orleans, Louisiana, USA, March 29–April 1, 1989, pp. 209–228.*, pp. 209–228, 1989. doi:10.1007/BFb0040259.

[6] P. Dybjer, "Inductive families," in *Formal aspects of computing*, pp. 440–465, 1994. doi:10.1016/S0049-237X(08)71945-1.

[7] P. Martin-Löf, "An intuitionistic theory of types: Predicative part," in *Studies in Logic and the Foundations of Mathematics*, vol. 80, pp. 73–118, 1975. doi:10.1016/S0049-237X(08)71945-1.

[8] M. Hofmann and T. Streicher, "The groupoid interpretation of type theory," in *In Venice Festschrift*, pp. 83–111, Oxford University Press, 1996.

[9] P.-L. Curien *et al.*, "Revisiting the categorical interpretation of dependent type theory," in *Theoretical Computer Science*, vol. 546, pp. 99–119, 2014. doi:10.1016/j.tcs.2014.03.003.

[10] S. Huber, "Cubical interpretations of type theory," in *Ph.D. thesis, Dept. Comp. Sci. Eng, University of Gothenburg*, 2016.

[11] M. Sokhatskyi and P. Maslianko, "The systems engineering of consistent pure language with effect type system for certified applications and higher languages," 2018. doi:10.1063/1.5045439.