Issue I: Type Theory

Максим Сохацький [1]

[1] Національний технічний університет України
Київський політехнічний інститут імені Ігоря Сікорського
15 травня 2025 р.

Анотація

Background. The long road from pure type systems of AUTOMATH by de Bruijn to type checkers with homotopical core was made. This article touches only the formal Martin-Löf Type Theory core type system with Π and Σ types (that correspond to $\forall$ and $\exists$ quantifiers for mathematical reasoning) and identity type (MLTT-75). Expressing the MLTT embedding in a host type checker for a long time was inaccessible due to the non-derivability of the J eliminator in pure functions. This was recently made possible by cubical type theory and cubical type checker.

Objective. Select the type system as a part of conceptual model of theorem proving system that is able to derive the J eliminator and its theorems based on the latest research in cubical type systems. The goal of this article is to demonstrate the formal embedding of MLTT-75 into **Per** with constructive proofs of the complete set of inference rules including J eliminator.

Methods. As types are formulated using 5 types of rules (formation, intro, elimination, computation, uniqueness) according to MLTT we constructed aliases for the host language primitives and used the cubical type checker to prove that it has the realization of MLTT-75.

Results. This work leads to several results: 1) **Per** — a special embedded version of type theory with infinite number of universes and Path type suitable for HoTT purposes without uniqueness rule of equality type; 2) The actual embedding of MLTT with syntax implying universe polymorphism and cubical primitives in **Per**; 3) The different interpretations of types were given: set-theoretical, groupoid, homotopical;
4) Internalization could be seen as an 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.

Conclusion. We should note that this is an entrance to the internalization technique, and after formal MLTT embedding, we could go through inductive types up to embedding of CW-complexes as the indexed gluing of the higher inductive types. This means the implementation of a wide spectrum of math theories inside HoTT up to algebraic topology.

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

Зміст

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 [3, 5] 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-75. 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 HoTT [1] prover cubicaltt[1] prover was presented [17].

Problem Statement

The formal initial problem was to create a full self-contained MLTT internalization in the host typechecker, where all theorems are being checked constructively. This task involves a modern techniques in type theory, namely cubical type theories. By following most advaced theories apply this results for building minimal type checker that is able to derive J and the whole MLTT theorems constructively. This leads us to the compact MLTT core yet compatible with future possible homotopy extensions.

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

Per 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
    f1 := f1 /\ f2
    f2 := −f2 + id + 0 + 1
  form := form \/ f1 + f1 + f2
 sides := #empty + cos + side
   cos := side , side + side , cos
    id := #list #nat
   ids := #list id
   mod := module id where imps dec
  imps := #list imp
   imp := import id
   brs := #empty + cobrs
 cobrs := | br brs
    br := ids → exp +ids @ ids → exp
   tel := #empty + cotel
   dec := #empty + codec
 cotel := (exp:exp) tel
 codec := def dec
   sum := #empty + id tel + id tel | sum
   def := data id tel=sum +id tel:exp=exp
        + id tel : exp where def
   app := exp exp
   exp := cotel * exp + cotel → exp
        + exp → exp +(exp) +id
        + (exp,exp) + \cotele → exp
        + split cobrs +exp .1
        + exp .2 +⟨ ids ⟩ exp
        + exp @ form + app + comp exp sys
```

Here := (definition), + (disjoint sum), #empty, #nat, #list are parts of BNF language and |, :, *, ⟨, ⟩, (,), =, \, /, -, →, 0, 1, @, [,], **module**, **import**, **data**, **split**, **where**, **comp**, **.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 Interpretations

Martin-Löf Type Theory MLTT-80 contains Π, Σ, Id, W, 0, 1, 2 types.

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 MLTT-75 [5] while original MLTT-72 with only Π and Σ was introduced in [3]. Predicative Universe Hierarchy was added in [4]. While original MLTT-75 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 [7].

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.

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.

Табл. 1: *

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
$\mathbf{1}$	$\top$ 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	

1.1 Logical 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.

1.2 Categorical Interpretation

Categorical interpretation [11] 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 an intermediate result in future issues. In some sense we include here topos-theoretical interpretations, with presheaf model of type theory as example (in this case fibrations are constructes as functors, categorically).

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 [7].

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.

1.4 Set 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.

MLTT-80 could be reduced to Π, Σ, Path types (MLTT-75) omitting polynomial functors W modeled by F-algebras and their terminators: 0, 1, 2 types. In this issue Π, Σ, Path are given as a core of MLTT-75. The inductive types will be disscussed in the upcoming **Issue II: Inductive Types**.

2 Internalized Type Theory

2.1 Dependent Product (Π)

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 $\forall$ 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)
```

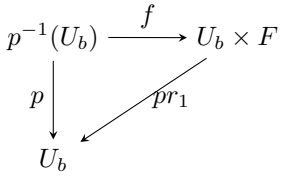
Definition 6. (Section). A section of morphism $f : A \to B$ in some category is the morphism $g : B \to A$ such that $f \circ g : B \xrightarrow{g} A \xrightarrow{f} B$ equals the identity morphism on B.

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.

2.2 Dependent Sum (Σ)

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)
```

2.3 Path (Ξ)

The Path identity type or Ξ 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.

[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`

Cubical interpretation

Cubical interpretation was first given by Simon Huber [18] and later was written first constructive type checker in the world by Anders Mörtberg [17].

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.

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

```
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 []
```

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

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

```
meet (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)

join (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)
```

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 $|\lambda$ (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 — |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, 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 20. (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 21. (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.

2.4 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 21. (Empty Context).

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

Definition 22. (Context Comprehension).

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

Definition 23. (Context Derivability).

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

2.5 Universes

Definition 24. (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 25. (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 26. (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 27. (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 28. (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 29. (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 30. (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 31. (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^∞, MLTT^∞).
$\mathrm{SAR} = \{\mathrm{U}_{i\in\mathrm{N}}, \mathrm{U_i} : \mathrm{U}_{\mathrm{j;i<j;i,j}\in\mathrm{N}}, \mathrm{U_i} \to \mathrm{U_j} : \mathrm{U}_{\lambda(\mathrm{i,j});\mathrm{i,j}\in\mathrm{N}}\}$. Where U_i is a universe of i-level or i-category in categorical interpretation. The working prototype of PTS^∞ is given in **Issue XVI: Pure Type System** [19].

2.6 MLTT-75

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-75). 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 22. (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.

```
cubicaltt — 6 second.
Arend — 1 second.
Agda (cubical) — & 2 second.
```

Conclusions

In this issue the type-theoretical model (interpretation) of MLTT was presented in cubical syntax and type checked in it. This is the first constructive proof of internalization of MLTT.

From the theoretical point of view 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.

Type theoretical cubical constructions was given for the Path types along the article for other interpretations, all of them were taken from our Groupoid Infinity [7] base library.

The objective of complete derivability of all eliminators, computational and uniquness rules is a basic objective for constructive mathematics as mathematical reasoning implies verification and mechanization. Yes cubical type system represent most compact system that make possible derivability of all theorems for core types which make this system as a first candidate for the metacircular type checker.

Also for programming purposes we may also want to investigate Fixpoint as a useful type in coinductive and modal type theories and harmful type in theoretical foundation of type systems. Elimination the possibility of uncontrolled Fixpoint is a main objective of the correct type system for reasoning without paradoxes. By this creatiria we could filter all the fixpoint implementations being condidered harmful.

Without a doubt the core type that makes type theory more like programming is the inductive type system that allows to define type families. In the following

[7] https://groupoid.space/

Табл. 2: *
Table. Core Features

Lang	Π	Σ	$\equiv$	Path	U^∞	Co/Fix	Lazy
PTS	x						
Cedile, MLTT	x	x	x				
Henk	x				x		
Per	x	x		x	x		
Lean, Agda	x		x	x	x		
NuPRL	x	x	x			x	
System-D	x	x				x	x
cubicaltt	x	x	x	x		x	

Issue II will be shown the semantics and embedding of inductive types with several types of Inductive-Recursive encodings.

Табл. 3: *
Table. Inductive Type Systems

Lang	Co/Inductive	Quot/Trunc	HITs
System-D	x		
Lean	x	x	
NuPRL	x	x	
Arend	x	x	x
Agda, Coq	x		x
cubicaltt, yacctt, RedPRL	x		x

Further research of the most pure type theory on a weak fibrations and pure Kan oprations without interval lattice structure (connections, de Morgan algebra, connection algebras) and diagonal coersions could be made on the way of building a minimal homotopy core [2].

The next language after **Henk** and **Per** will be **Anders** with homotopy type system and infinite number of universes. Along with **Joe** cartesian interpreter this evaluators form a set of languages as a part of conceptual model of theorem proving system with formalized virtual machine as extraction target.

Табл. 4: *
Table. Cubical Type Systems

Lang	Interval	Diagonal	Kan/Coe
BCH, cubical			$0 \to r$, $1 \to r$
CCHM, cubicaltt, Agda	$\vee$,$\wedge$		$0 \to 1$
Dedekind	$\vee$,$\wedge$		$0 \to 1$, $1 \to 0$
AFH/ABCFHL, yacctt		x	$r \to s$
HTS/CMS			$r \to s$, *weak*

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 Foundations volume of articles define formal programming language with geometric foundations and show how to prove properties of such constructions. The second volume of article is dedicated to cover the programming and modeling of Mathematics.

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

Issue II: Inductive Types. 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: Modalities. The constructive extensions with additional context and adjoint transports between toposes (cohesive toposes). This approach serves the needs of modal logics, differential geometry, cohomology.

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.

Issue XVI: Pure Type System. Pure Type System named after **Henk** Barendregt.

Issue XVII: Inductive Type System. Inductive Type System named after **Per** Martin-Löf.

Issue XIX: Modal Homotopy Type System. Modal Homotopy Type System named after **Anders** Mörtberg.

Література

[1] Vladimir Voevodsky et al., *Homotopy Type Theory*, in *Univalent Foundations of Mathematics*, 2013.

[2] Evan Cavallo, Anders Mörtberg, and Andrew W. Swan, *Unifying Cubical Models of Univalent Type Theory*, Preprint, 2019. `http://www.cs.cmu.edu/~amoertbe/papers/unifying.pdf`

[3] Per Martin-Löf and Giovanni Sambin, *The Theory of Types*, in *Studies in Proof Theory*, 1972.

[4] Per 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`

[5] Per Martin-Löf and Giovanni Sambin, *Intuitionistic Type Theory*, in *Studies in Proof Theory*, 1984.

[6] Thierry Coquand and Gérard Huet, *The Calculus of Constructions*, in *Information and Computation*, pp. 95–120, 1988. `doi:10.1016/0890-5401(88)90005-3`

[7] Martin Hofmann and Thomas Streicher, *The Groupoid Interpretation of Type Theory*, in *Venice Festschrift*, Oxford University Press, pp. 83–111, 1996.

[8] Claudio Hermida and Bart Jacobs, *Fibrations with Indeterminates: Contextual and Functional Completeness for Polymorphic Lambda Calculi*, in *Mathematical Structures in Computer Science*, vol. 5, pp. 501–531, 1995.

[9] Alexandre Buisse and Peter Dybjer, *The Interpretation of Intuitionistic Type Theory in Locally Cartesian Closed Categories – an Intuitionistic Perspective*, in *Electronic Notes in Theoretical Computer Science*, pp. 21–32, 2008. `doi:10.1016/j.entcs.2008.10.003`

[10] Andreas Abel, Thierry Coquand, and Peter Dybjer, *On the Algebraic Foundation of Proof Assistants for Intuitionistic Type Theory*, in *Functional and Logic Programming*, Springer, Berlin, Heidelberg, pp. 3–13, 2008.

[11] Pierre-Louis 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`

[12] Errett Bishop, *Foundations of Constructive Analysis*, 1967.

[13] Bengt Nordström, Kent Petersson, and Jan M. Smith, *Programming in Martin-Löf's Type Theory*, Oxford University Press, 1990.

[14] Matthieu Sozeau and Nicolas Tabareau, *Internalizing Intensional Type Theory*, unpublished.

[15] Martin Hofmann and Thomas Streicher, *The Groupoid Model Refutes Uniqueness of Identity Proofs*, in *Logic in Computer Science (LICS'94)*, IEEE, pp. 208–212, 1994.

[16] Bart Jacobs, *Categorical Logic and Type Theory*, vol. 141, 1999.

[17] Anders Mörtberg et al., *Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom*, arXiv:1611.02108, 2017.

[18] Simon Huber, *Cubical Interpretations of Type Theory*, Ph.D. thesis, Dept. of Computer Science and Engineering, University of Gothenburg, 2016.

[19] Maksym Sokhatskyi and Pavlo Maslianko, *The Systems Engineering of Consistent Pure Language with Effect Type System for Certified Applications and Higher Languages*, in *Proc. 4th Int. Conf. Mathematical Models and Computational Techniques in Science and Engineering*, 2018. doi:10.1063/1.5045439