Issue I: Type Theory

Максим Сохацький [1]

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

Анотація

Martin-Löf Type Theory (MLTT), introduced by Per Martin-Löf in 1972, is a cornerstone of constructive mathematics, providing a foundation for formalizing mathematical proofs and programming languages. Its 1975 variant, MLTT-75, incorporates dependent types (Π, Σ) and identity types, with the J eliminator as a key construct for reasoning about equality. Historically, internalizing MLTT-75 in a type checker while constructively proving the J eliminator has been challenging due to limitations in pure functional systems. This article presents a canonical formalization of MLTT-75 and its complete internalization in **Per**, a minimal type theory language equipped with cubical type primitives. Using cubical type theory, we constructively prove all MLTT-75 inference rules, including the J eliminator, and demonstrate **Per**'s suitability as a robust type checker. We also provide logical, categorical, and homotopical interpretations of MLTT-75 to contextualize its significance. This work advances the mechanization of constructive mathematics and offers a blueprint for future type-theoretic explorations.
Keywords: Martin-Löf Type Theory, Cubical Type Theory.

Зміст

Introduction

For decades, type theorists have sought to fully internalize Martin-Löf Type Theory (MLTT) within a type checker, a task akin to building a self-verifying blueprint for mathematics. Introduced by Per Martin-Löf in 1972 [3] and refined in 1975 [4], MLTT-75 is a foundational system that combines dependent types (Π for universal quantification, Σ for existential quantification) with identity types, enabling rigorous reasoning about equality. Central to MLTT-75 is the J eliminator, a rule that governs how identity proofs are used, but its constructive derivation has long eluded pure functional type checkers due to the complexity of equality types.

This article addresses this challenge by presenting a canonical formalization of MLTT-75 and its complete internalization in **Per**, a novel type theory language designed for constructive proofs. Leveraging cubical type theory [17], **Per** incorporates Path types and universe polymorphism to faithfully embed MLTT-75's rules, achieving a constructive proof of the J eliminator for the first time. This internalization serves as an ultimate test of a type checker's robustness, verifying its ability to fuse introduction and elimination rules through beta and eta equalities.

To make MLTT-75 accessible, we provide intuitive interpretations of its types: logical (as quantifiers), categorical (as functors), and homotopical (as spaces). These perspectives highlight MLTT's role as a bridge between mathematics and computation. Our work builds on Martin-Löf's vision of constructive mathematics, offering a minimal yet powerful framework for mechanized reasoning. We aim to inspire researchers and practitioners to explore type theory's potential in formalizing mathematics and designing reliable software.

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), introduced by Per Martin-Löf in 1972 [3] and refined in 1975 [4], is a foundational system for constructive mathematics, blending logical rigor with computational expressiveness. Its 1975 variant, MLTT-75, centers on dependent types (Π, Σ) and identity types (Id), which underpin its ability to formalize mathematical reasoning and type checking. This section explores four interpretations of MLTT-75—logical, categorical, homotopical, and set-theoretical—to illuminate its versatility and contextualize its internalization in the **Per** language. These perspectives reveal MLTT-75 as a unifying framework bridging logic, category theory, homotopy theory, and set theory, with each interpretation highlighting distinct aspects of its types and rules.

In MLTT, types are defined by five classes of rules: (1) *formation*, specifying the type's signature; (2) *introduction*, defining constructors for its elements; (3) *elimination*, providing a dependent induction principle; (4) *computation* (beta-equality), governing reduction; and (5) *uniqueness* (eta-equality), ensuring canonical forms, though the latter is absent for identity types in homotopical settings. For MLTT-75, we focus on Π (dependent function types), Σ (dependent pair types), and Id (identity types), with the latter replaced by Path types in cubical type theory to enable constructive proofs, such as the J eliminator, in **Per**.

The identity type, introduced in MLTT-75 [4], is particularly significant, enabling reasoning about equality constructively. Unlike MLTT-72, which included only Π and Σ types, MLTT-75's Id types originally enforced uniqueness of identity proofs (UIP) via an eta-rule. However, modern homotopical interpretations, pioneered by Hofmann and Streicher [7], refute UIP, adopting Path

types that model equality as paths in a space, aligning with cubical type theory's constructive framework. This shift is crucial for **Per**, as Path types facilitate the internalization of MLTT-75's rules.

Type checkers operate within contexts, binding variables to indexed universes, built-in types, or user-defined types via de Bruijn indices or names. These contexts enable queries about type derivability and code extraction, forming the core of **Per**'s type checker. By encoding MLTT-75's syntax and rules, **Per** supports multiple interpretations, each offering unique insights into its structure and applications.

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
$\mathbf{0}$	$\bot$ 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

The logical interpretation casts MLTT-75 as a system for intuitionistic higher-order logic, where types correspond to propositions and terms to proofs, embodying the Curry-Howard correspondence. In this view, a type A represents a proposition, and a term $a : A$ is a proof of A. The Π-type, $\prod_{x:A} B(x)$, encodes universal quantification ($\forall x : A, B(x)$), while the Σ-type, $\sum_{x:A} B(x)$, represents existential quantification ($\exists x : A, B(x)$). The identity type, $\mathrm{Id}_A(a, b)$, captures propositional equality ($a =_A b$), with the J eliminator providing a constructive means to reason about equalities.

Each type's five rules (formation, introduction, elimination, computation, and uniqueness, except for Id in cubical settings) mirror the structure of logical inference rules. For instance, the introduction rule for Π constructs a lambda term (proof of a universal statement), while its elimination rule applies the term

to an argument (using the universal statement).

MLTT-75 is not standalone framework for constructive mathematics but rather the extended foundational core on top of MLTT-72. Adding **0** (Empty), **1** (Unit), **2** (Bool) types allows resulting type system to internalize intuitionistic propositional logic (IPL), via Gödel's double-negation translation, classical logic can be encoded within IPL [13]. In **Per**, this logical framework underpins the type checker's ability to verify MLTT-75's rules, ensuring constructive consistency.

1.2 Categorical Interpretation

The categorical interpretation models MLTT-75 within category theory, where types are objects, terms are morphisms, and type constructions are functors. This perspective, formalized by Cartmell and Seely [16], views MLTT-75 with **0**, **1**, **2** types as a locally cartesian closed category (LCCC). Here, Π-types correspond to dependent products (right adjoints to base change functors), and Σ-types to dependent sums (left adjoints). The identity type, Id_A, is modeled as a path space object, reflecting equality as a morphism.

For example, given a morphism $f : A \to B$ in a category, the Π_f functor maps a dependent type over B to one over A, generalizing function spaces, while Σ_f constructs the total space of a fibration. In **Per**, this interpretation informs the type checker's handling of dependent types, with cubical primitives enabling precise categorical semantics for Path types. Topos-theoretical models, such as presheaves, further enrich this interpretation by treating fibrations as functors, aligning with MLTT-75's expressive power [11].

1.3 Homotopical Interpretation

The homotopical interpretation, a breakthrough in modern type theory, views MLTT-75's types as spaces and terms as points, with identity types as paths. Introduced by Hofmann and Streicher's groupoid model [7], this perspective refutes the uniqueness of identity proofs (UIP) in classical MLTT-75, replacing Id with Path types that model equality as continuous paths in a space. In cubical type theory, Path types are functions from an interval $[0, 1]$ to a type, enabling constructive proofs of MLTT-75's rules, including the J eliminator, in **Per**.

Here, Π-types represent spaces of sections, Σ-types denote total spaces of fibrations, and Path types form path spaces (A^I). This interpretation connects MLTT-75 to homotopy theory, where types are ∞-groupoids, and fibrations (dependent types) are studied geometrically. For instance, a Π-type can be seen as a trivial fiber bundle, with its introduction rule constructing a section [1]. In **Per**, cubical primitives like connections and compositions support this interpretation, making MLTT-75's internalization homotopically robust.

1.4 Set-Theoretical Interpretation

The set-theoretical interpretation models MLTT-75's types as sets and terms as elements, aligning with classical first-order logic. In this view, a type A is a

set, and a term $a : A$ is an element. The Π-type represents a set of functions, Σ-type a disjoint union of sets, and $\mathrm{Id}_A(a, b)$ an equality relation. However, this interpretation is limited, as it cannot capture higher equalities (e.g., paths between paths) or inductive types directly, due to its 0-truncated nature [1].

In homotopical terms, sets are modeled as 0-types (types with no non-trivial higher paths), but MLTT-75's identity types introduce higher structure, making the set-theoretical view less expressive. In **Per**, the set-theoretical interpretation serves as a baseline, with cubical Path types extending it to handle MLTT-75's full complexity. This interpretation clarifies the boundaries of classical logic within MLTT-75, emphasizing the need for homotopical or categorical models for complete internalization.

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 ∀ 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)
```

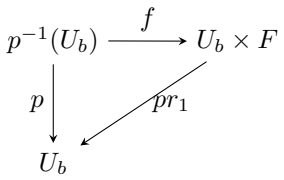
Definition 6. (Section). A section of morphism $f : A \to B$ in some category is the morphism $g : B \to A$ such that $f \circ g : B \xrightarrow{g} A \xrightarrow{f} B$ equals the identity morphism on B.

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 [1]CCHM cubicaltt model with connections. For [2]ABCFHL yacctt model with variables please refer to ytt file. You may also want

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

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

to read [3]BCH, [4]AFH. There is a [5]PO paper about CCHM axiomatic in a topos.

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
```

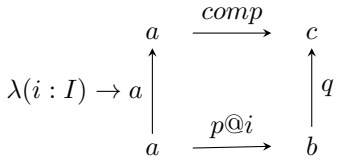
Definition 16. (Path Reflexivity). Returns an element of reflexivity path space for a given value of the type. The inhabitant of that path space is the lambda on the homotopy interval $[0, 1]$ that returns a constant value a. Written in syntax as |<i>a| which equals to $\lambda\ (i : I) \to a$.

```
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.




```
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).

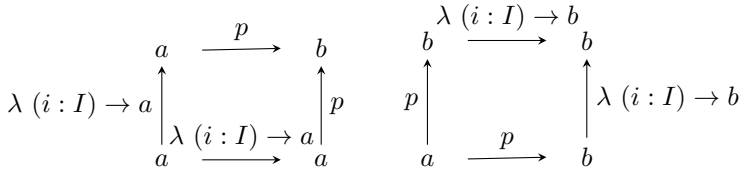
```
inv (A: U) (a b: A) (p: Path A a b)
  : Path A b a = <i> p @ -i
```

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

[4]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

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

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




```
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) $\to$ f (p @ i)$|$ for both cases.

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

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

Theorem 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). 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^{∞}).
SAR = $\{\mathrm{U}_{i\in\mathrm{N}}, \mathrm{U_i} : \mathrm{U_{j;i<j;i,j\in N}}, \mathrm{U_i} \to \mathrm{U_j} : \mathrm{U}_{\lambda(\mathrm{i,j});\mathrm{i,j}\in\mathrm{N}}\}$. 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 [6] 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

[6] 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