Volume IV: Mathematics

Introduction to Formalization of Mathematics

Namdak Tonpa

2024 · Groupoid Infinity

IV

Зміст

Structure Preserving Theorems

Namdak Tonpa

30 квітня 2025 р.

Анотація

This article unifies algebra and geometry by characterizing algebra as the domain of homomorphisms preserving structure and geometry as the domain of inverse images of homomorphisms preserving structure. We introduce two new theorems: the Homomorphism Preservation Theorem (HPT) for Algebraic Categories and the Inverse Image Preservation Theorem (IIPT) for Geometric Categories. These build on foundational results like the First Isomorphism Theorem, Continuity Theorem, Pullback Theorem, Stone Duality, Gelfand Duality, and Adjoint Functor Theorem. Aimed at advanced graduate students, this exposition uses category theory to illuminate the algebraic-geometric duality.

1 Structure Preservation Theorems in Algebra and Geometry

Algebra and geometry, foundational to pure mathematics, differ in focus: algebra on abstract structures and their transformations, geometry on spatial properties and invariants. We propose a unifying perspective: algebra is defined by homomorphisms preserving structure, and geometry by the inverse images of homomorphisms preserving structure. This article formalizes this view through two explicit theorems—the Homomorphism Preservation Theorem (HPT) for Algebraic Categories and the Inverse Image Preservation Theorem (IIPT) for Geometric Categories—building on established results. Assuming familiarity with category theory, algebraic topology, and commutative algebra, we provide a framework for graduate students to explore these fields' interplay.

1.1 Homomorphisms in Algebra

Definition 1. Let $\mathcal{C}$ be a category, and let A, B be objects in $\mathcal{C}$. A *homomorphism* $\phi : A \to B$ is a morphism in $\mathcal{C}$ that preserves the structure defined by the category's operations and relations.

In algebraic categories (e.g., **Grp**, **Ring**, $\mathbf{Mod}_R$), homomorphisms preserve operations like group multiplication or module scalar multiplication.

Example 1. In **Grp**, a group homomorphism $\phi : G \to H$ satisfies $\phi(g_1 g_2) = \phi(g_1)\phi(g_2)$ for all $g_1, g_2 \in G$, preserving the group operation.

Theorem 1 (First Isomorphism Theorem)**.** Let $\phi : G \to H$ be a group homomorphism with kernel $K = \ker(\phi)$. Then $G/K \cong \operatorname{im}(\phi)$.

Theorem 2 (Universal Property of Free Objects)**.** In an algebraic category (e.g., **Grp**, **Ring**), for a free object $F(X)$ on a set X, any map $f : X \to A$ (where A is an object) extends uniquely to a homomorphism $\phi : F(X) \to A$.

We now introduce a theorem encapsulating the algebraic perspective.

Theorem 3 (Homomorphism Preservation Theorem for Algebraic Categories)**.** Let $\mathcal{C}$ be an algebraic category (e.g., **Grp**, **Ring**, $\mathbf{Mod}_R$) with a forgetful functor $U : \mathcal{C} \to \mathbf{Set}$. For any surjective homomorphism $\phi : A \to B$ in $\mathcal{C}$ with kernel K (a normal subobject), there exists an isomorphism $\psi : A/K \to B$ such that $\psi \circ \pi = \phi$, where $\pi : A \to A/K$ is the canonical projection. Moreover, any object A can be generated by a free object $F(X)$ via a surjective homomorphism whose structure is preserved by ϕ.

Доведення. The first part follows from the First Isomorphism Theorem [1]: for a surjective homomorphism $\phi : A \to B$ with kernel K, the quotient $A/K \cong B$ via the isomorphism $\psi : aK \mapsto \phi(a)$. The second part follows from the Universal Property of Free Objects [2]: for any object A, there exists a set X and a free object $F(X)$ with a surjective homomorphism $\eta : F(X) \to A$, and any homomorphism $\phi : A \to B$ extends the structure-preserving maps from $F(X)$. ☐

Remark 1. The HPT formalizes that homomorphisms in algebraic categories preserve structure forward, inducing isomorphisms on quotients and respecting generators, unifying the First Isomorphism Theorem and Universal Property. The name avoids confusion with the Structure-Identity Principle in category theory [2].

1.2 Homomorphisms in Geometry

Geometry emphasizes spaces where structure is preserved under inverse images of homomorphisms, as in **Top** or **Sch**.

Definition 2. Let $\phi : X \to Y$ be a morphism in a category $\mathcal{C}$. The *inverse image* of a subobject $S \subseteq Y$ (if it exists) is the subobject $\phi^{-1}(S) \subseteq X$ defined via the pullback of $S \hookrightarrow Y$ along ϕ.

Example 2. In **Top**, a continuous map $\phi : X \to Y$ ensures that $\phi^{-1}(V) \subseteq X$ is open for every open set $V \subseteq Y$.

Theorem 4 (Continuity in Topology)**.** A function $\phi : X \to Y$ between topological spaces is continuous if and only if for every open set $V \subseteq Y$, the inverse image $\phi^{-1}(V)$ is open in X.

Theorem 5 (Pullback Theorem in Sheaf Theory)**.** For a morphism $\phi : X \to Y$ in a category with sheaves (e.g., **Top**, **Sch**), the inverse image functor $\phi^{-1} : \mathrm{Sh}(Y) \to \mathrm{Sh}(X)$ is exact, preserving the structure of sheaves.

We now define a theorem for geometric categories.

Theorem 6 (Inverse Image Preservation Theorem for Geometric Categories)**.** Let $\mathcal{C}$ be a geometric category (e.g., **Top**, **Sch**) with pullbacks. For any morphism $\phi : X \to Y$ in $\mathcal{C}$, the inverse image functor $\phi^{-1} : \mathrm{Sub}(Y) \to \mathrm{Sub}(X)$ preserves the lattice structure of subobjects. If $\mathcal{C}$ admits sheaves, $\phi^{-1} : \mathrm{Sh}(Y) \to \mathrm{Sh}(X)$ is exact and preserves sheaf isomorphisms, ensuring that the geometric structure of Y is reflected in X.

Доведення. In **Top**, the Continuity Theorem [4] ensures that $\phi : X \to Y$ is continuous if and only if $\phi^{-1}(V)$ is open for every open set $V \subseteq Y$, so ϕ^{-1} preserves the lattice of open sets. In categories with sheaves (e.g., **Top**, **Sch**), the Pullback Theorem [5] guarantees that $\phi^{-1} : \mathrm{Sh}(Y) \to \mathrm{Sh}(X)$ is exact, preserving sheaf structures. For schemes, ϕ^{-1} maps prime ideals to prime ideals [3], preserving geometric properties. Since ϕ^{-1} is functorial and preserves monomorphisms, it maintains isomorphisms of subobjects or sheaves. □

Remark 2. The IIPT captures the geometric essence of inverse images preserving structure, unifying the Continuity Theorem and Pullback Theorem. The name distinguishes it from the Structure-Identity Principle [2].

Example 3. For a morphism of schemes $\phi : X \to Y$, the inverse image of a prime ideal under the induced map on stalks is prime, preserving geometric structure [3].

1.3 Categorical Unification

Category theory bridges algebra and geometry through dualities, where the HPT and IIPT interplay.

Theorem 7 (Stone Duality)**.** The category of Boolean algebras, **BoolAlg**, is dually equivalent to the category of Stone spaces, **Stone**, via the spectrum functor.

Theorem 8 (Gelfand Duality)**.** The category of commutative C^*-algebras is dually equivalent to the category of compact Hausdorff spaces via the spectrum functor.

Theorem 9 (Adjoint Functor Theorem)**.** In a complete category, a functor has a left adjoint if it preserves limits, and a right adjoint if it preserves colimits.

Remark 3. Stone and Gelfand Dualities [6, 7] connect algebraic homomorphisms (HPT) to geometric inverse images (IIPT). The Adjoint Functor Theorem [2] underpins dualities like Spec, where algebraic and geometric structures are preserved [3].

Example 4. The Spec functor maps a ring homomorphism $\phi : R \to S$ to a morphism $\mathbf{Spec}S \to \mathbf{Spec}R$, with inverse images of prime ideals preserving geometric structure.

1.4 Applications and Implications

The HPT and IIPT, supported by prior results, impact advanced research:

- **Algebraic Topology**: The HPT governs homology maps, while the IIPT defines covering spaces.
- **Algebraic Geometry**: The IIPT underpins étale cohomology via inverse images, while the HPT applies to ring homomorphisms.
- **Category Theory**: Stone, Gelfand, and Adjoint Functor Theorems reveal algebra-geometry correspondences.

Corollary 1. In any category with pullbacks, $\phi^{-1} : \mathrm{Sub}(Y) \to \mathrm{Sub}(X)$ preserves subobject lattices, as per the IIPT.

1.5 Conclusion

The Homomorphism Preservation Theorem and Inverse Image Preservation Theorem formalize that algebra preserves structure via homomorphisms and geometry via inverse images. Building on the First Isomorphism Theorem, Continuity Theorem, Pullback Theorem, and dualities, these theorems unify pure mathematics. Graduate students are encouraged to apply this framework to algebraic topology, algebraic geometry, and category theory, deepening their research.

Література

[1] Lang, S., *Algebra*, Graduate Texts in Mathematics, Springer, 2002.

[2] Mac Lane, S., *Categories for the Working Mathematician*, Graduate Texts in Mathematics, Springer, 1998.

[3] Hartshorne, R., *Algebraic Geometry*, Graduate Texts in Mathematics, Springer, 1977.

[4] Munkres, J. R., *Topology*, 2nd ed., Prentice Hall, 2000.

[5] Kashiwara, M., Schapira, P., *Sheaves on Manifolds*, Springer, 1990.

[6] Johnstone, P. T., *Stone Spaces*, Cambridge University Press, 1982.

[7] Takesaki, M., *Theory of Operator Algebras I*, Springer, 2002.

Issue XXVII: Formal Topos on Category of Sets

Maxim Sokhatsky

[1] National Technical University of Ukraine
Igor Sikorsky Kyiv Polytechnical Institute
30 квітня 2025 р.

Анотація

The purpose of this work is to clarify all topos definitions using type theory. Not much efforts was done to give all the examples, but one example, a topos on category of sets, is constructively presented at the finale.

As this cricial example definition is used in presheaf definition, the construction of category of sets is a mandatory excercise for any topos library. We propose here cubicaltt[1] version of elementary topos on category of sets for demonstration of categorical semantics (from logic perspective) of the fundamental notion of set theory in mathematics.

Other disputed foundations for set theory could be taken as: ZFC, NBG, ETCS. We will disctinct syntetically: i) category theory; ii) set theory in univalent foundations; iii) topos theory, grothendieck topos, elementary topos. For formulation of definitions and theorems only Martin-Löf Type Theory is requested. The proofs involve cubical type checker primitives.

Keywords: Homotopy Type Theory, Topos Theory

[1]Cubical Type Theory, http://github.com/mortberg/cubicaltt

2 Introduction to Topos Theory

One can admit two topos theory lineages. One lineage takes its roots from published by Jean Leray in 1945 initial work on sheaves and spectral sequences. Later this lineage was developed by Henri Paul Cartan, André Weil. The peak of lineage was settled with works by Jean-Pierre Serre, Alexander Grothendieck, and Roger Godement.

Second remarkable lineage take its root from William Lawvere and Myles Tierney. The main contribution is the reformulation of Grothendieck topology by using subobject classifier.

2.1 Category Theory

First of all very simple category theory up to pullbacks is provided. We give here all definitions only to keep the context valid.

Definition 3. (Category Signature). The signature of category is a $\sum_{A:U} A \to A \to U$ where U could be any universe. The pr_1 projection is called Ob and pr_2 projection is called $\mathrm{Hom}(a, b)$, where $a, b : \mathrm{Ob}$.

```
cat: U = (A: U) * (A -> A -> U)
```

Definition 4. (Precategory). More formal, precategory C consists of the following. (i) A type Ob_C, whose elements are called objects; (ii) for each $a, b : \mathrm{Ob}_C$, a set $\mathrm{Hom}_C(a, b)$, whose elements are called arrows or morphisms. (iii) For each $a : \mathrm{Ob}_C$, a morphism $1_a : \mathrm{Hom}_C(a, a)$, called the identity morphism. (iv) For each $a, b, c : \mathrm{Ob}_C$, a function $\mathrm{Hom}_C(b, c) \to \mathrm{Hom}_C(a, b) \to \mathrm{Hom}_C(a, c)$ called composition, and denoted $g \circ f$. (v) For each $a, b : \mathrm{Ob}_C$ and $f : \mathrm{Hom}_C(a, b)$, $f = 1_b \circ f$ and $f = f \circ 1_a$. (vi) For each $a, b, c, d : A$ and $f : \mathrm{Hom}_C(a, b)$, $g : \mathrm{Hom}_C(b, c)$, $h : \mathrm{Hom}_C(c, d)$, $h \circ (g \circ f) = (h \circ g) \circ f$.

```
isPrecategory (C: cat): U
  = (id: (x: C.1) -> C.2 x x)
  * (c: (x y z: C.1) -> C.2 x y -> C.2 y z -> C.2 x z)
  * (homSet: (x y: C.1) -> isSet (C.2 x y))
  * (left: (x y: C.1) -> (f: C.2 x y) ->
    Path (C.2 x y) (c x x y (id x) f) f)
  * (right: (x y: C.1) -> (f: C.2 x y) ->
    Path (C.2 x y) (c x y y f (id y)) f)
  * ( (x y z w: C.1) -> (f: C.2 x y) ->
    (g: C.2 y z) -> (h: C.2 z w) ->
    Path (C.2 x w) (c x z w (c x y z f g) h)
                   (c x y w f (c y z w g h)))

carrier (C: precategory) : U
hom     (C: precategory) (a b: carrier C) : U
compose (C: precategory) (x y z: carrier C)
        (f: hom C x y) (g: hom C y z) : hom C x z
```

Definition 5. (Categorical Pullback). The pullback of the cospan $A \xrightarrow{f} C \xleftarrow{g} B$ is a object $A \times_C B$ with morphisms $pb_1 : \times_C \to A$, $pb_2 : \times_C \to B$, such that diagram commutes:

$$\begin{tikzcd}
A \times_C B \arrow[r, "pb_2"] \arrow[d, "f"'] \arrow[dr] & B \arrow[d, "pb_1"] \\
A \arrow[r, "g"] & C
\end{tikzcd}$$

Pullback $(\times_C, pb_1, pb_2)$ must be universal, means for any (D, q_1, q_2) for which diagram also commutes there must exists a unique $u : D \to \times_C$, such that $pb_1 \circ u = q_1$ and $pb_2 \circ q_2$.

```
homTo (C: precategory) (X: carrier C): U
  = (Y: carrier C) * hom C Y X
cospan (C: precategory): U
  = (X: carrier C) * (_: homTo C X) * homTo C X
cospanCone (C: precategory) (D: cospan C): U
  = (W: carrier C) * hasCospanCone C D W
cospanConeHom (C: precategory) (D: cospan C)
    (E1 E2: cospanCone C D) : U
  = (h: hom C E1.1 E2.1) * isCospanConeHom C D E1 E2 h
isPullback (C: precategory) (D: cospan C) (E: cospanCone C D) : U
  = (h: cospanCone C D) -> isContr (cospanConeHom C D h E)
hasPullback (C: precategory) (D: cospan C) : U
  = (E: cospanCone C D) * isPullback C D E
```

Definition 6. (Category Functor). Let A and B be precategories. A functor $F : A \to B$ consists of: (i) A function $F_{Ob} : Ob_h A \to Ob_B$; (ii) for each $a, b : Ob_A$, a function $F_{Hom} : Hom_A(a, b) \to Hom_B(F_{Ob}(a), F_{Ob}(b))$; (iii) for each $a : Ob_A$, $F_{Ob}(1_a) = 1_{F_{Ob}}(a)$; (iv) for $a, b, c : Ob_A$ and $f : Hom_A(a, b)$ and $g : Hom_A(b, c)$, $F(g \circ f) = F_{Hom}(g) \circ F_{Hom}(f)$.

```
catfunctor (A B: precategory): U
  = (ob: carrier A -> carrier B)
  * (mor: (x y:carrier A)->hom A x y->hom B(ob x)(ob y))
  * (id: (x: carrier A) -> Path (hom B (ob x) (ob x))
    (mor x x (path A x)) (path B (ob x)))
  * ((x y z: carrier A) -> (f: hom A x y) -> (g: hom A y z) ->
    Path (hom B (ob x) (ob z)) (mor x z (compose A x y z f g))
         (compose B (ob x) (ob y) (ob z) (mor x y f) (mor y z g)))
```

Definition 7. (Terminal Object). Is such object Ob_C, that

$$\prod_{x,y:\mathrm{Ob}_C} \mathrm{isContr}(\mathrm{Hom}_C(y,x)).$$

```
isTerminal (C: precategory) (y: carrier C): U
  = (x: carrier C) -> isContr (hom C x y)
terminal (C: precategory): U
  = (y: carrier C) * isTerminal C y
```

2.2 Set Theory

Here is given the ∞-groupoid model of sets.

Definition 8. (Mere proposition, PROP). A type P is a mere proposition if for all $x, y : P$ we have $x = y$:

$$\mathrm{isProp}(P) = \prod_{x,y:P} (x = y).$$

Definition 9. (0-type). A type A is a 0-type is for all $x, y : A$ and $p, q : x =_A y$ we have $p = q$.

Definition 10. (1-type). A type A is a 1-type if for all $x, y : A$ and $p, q : x =_A y$ and $r, s : p =_{=_A} q$, we have $r = s$.

Definition 11. (A set of elements, SET). A type A is a SET if for all $x, y : A$ and $p, q : x = y$, we have $p = q$:

$$\mathrm{isSet}(A) = \prod_{x,y:A} \prod_{p,q:x=y} (p = q).$$

Definition 12. `data N = Z  | S (n: N)`

```
n_grpd (A: U) (n: N): U = (a b: A) -> rec A a b n where
  rec (A: U) (a b: A) : (k: N) -> U
    = split { Z -> Path A a b ; S n -> n_grpd (Path A a b) n }

isContr (A: U): U = (x: A) * ((y: A) -> Path A x y)
isProp  (A: U): U = n_grpd A Z
isSet   (A: U): U = n_grpd A (S Z)
PROP    : U = (X:U) * isProp X
SET     : U = (X:U) * isSet X
```

Definition 13. (Π-Contractability). If fiber is set thene path space between any sections is contractible.

```
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
```

Definition 14. (Σ-Contractability). If fiber is set then Σ is set.

```
setSig (A:U) (B: A -> U) (base: isSet A)
       (fiber: (x:A) -> isSet (B x)) : isSet (Sigma A B)
```

Definition 15. (Unit type, 1). The unit 1 is a type with one element.

```
data unit = tt
unitRec (C: U) (x: C): unit -> C = split tt -> x
unitInd (C: unit -> U) (x: C tt): (z:unit) -> C z
  = split tt -> x
```

Theorem 10. (Category of Sets, **Set**). Sets forms a Category. All compositional theorems proved by using reflection rule of internal language. The proof that Hom forms a set is taken through Π-contractability.

```
Set: precategory = ((Ob,Hom),id,c,HomSet,L,R,Q) where
  Ob: U = SET
  Hom (A B: Ob): U = A.1 -> B.1
  id (A: Ob): Hom A A = idfun A.1
  c (A B C: Ob) (f: Hom A B) (g: Hom B C): Hom A C
    = o A.1 B.1 C.1 g f
  HomSet (A B: Ob): isSet (Hom A B) = setFun A.1 B.1 B.2
  L (A B:Ob) (f:Hom A B): Path (Hom A B)(c A A B (id A)f)f
    = refl (Hom A B) f
  R (A B:Ob) (f:Hom A B): Path (Hom A B)(c A B B f(id B))f
    = refl (Hom A B) f
  Q (A B C D: Ob) (f:Hom A B) (g:Hom B C) (h:Hom C D)
    : Path (Hom A D) (c A C D (c A B C f g) h)
                     (c A B D f (c B C D g h))
    = refl (Hom A D) (c A B D f (c B C D g h))
```

3 Topos Theory

Topos theory extends category theory with notion of topological structure but reformulated in a categorical way as a category of sheaves on a site or as one that has cartesian closure and subobject classifier. We give here two definitions.

3.1 Topological Structure

Definition 16. (Topology). The topological structure on A (or topology) is a subset $S \in$ A with following properties: i) any finite union of subsets of S is belong to S; ii) any finite intersection of subsets of S is belong to S. Subets of S are called open sets of family S.

```
Structure topology (A : Type) := {
          open :> (A -> Prop) -> Prop;
          empty_open: open (empty _);
          full_open:  open (full _);
          inter_open: forall u,
                      open u -> forall v, open v
                             -> open (inter A u v) ;
          union_open: forall s, (subset _ s open)
                             -> open (union A s) }.
```

For fully functional general topology theorems and Zorn lemma you can refer to the Coq library [2]topology by Daniel Schepler.

3.2 Grothendieck Topos

Grothendieck Topology is a calculus of coverings which generalizes the algebra of open covers of a topological space, and can exist on much more general categories. There are three variants of Grothendieck topology definition: i) sieves; ii) coverage; iii) covering families. A category have one of these three is called a Grothendieck site.

Examples: Zariski, flat, étale, Nisnevich topologies.

A sheaf is a presheaf (functor from opposite category to category of sets) which satisties patching conditions arising from Grothendieck topology, and applying the associated sheaf functor to preashef forces compliance with these conditions.

The notion of Grothendieck topos is a geometric flavour of topos theory, where topos is defined as category of sheaves on a Grothendieck site with geometric moriphisms as adjoint pairs of functors between topoi, that satisfy exactness properties. [**?**]

As this flavour of topos theory uses category of sets as a prerequisite, the formal construction of set topos is cricual in doing sheaf topos theory.

Definition 17. (Sieves). Sieves are a family of subfunctors

$$R \subset Hom_C(\_, U), U \in \mathrm{C},$$

such that following axioms hold: i) (base change) If $R \subset Hom_C(\_, U)$ is covering and $\phi : V \to U$ is a morphism of C, then the subfuntor

$$\phi^{-1}(R) = \{\gamma : W \to V \| \phi \cdot \gamma \in R\}$$

is covering for V; ii) (local character) Suppose that $R, R' \subset Hom_C(\_, U)$ are subfunctors and R is covering. If $\phi^{-1}(R')$ is covering for all $\phi : V \to U$ in R, then R' is covering; iii) $Hom_C(\_, U)$ is covering for all $U \in \mathrm{C}$.

[2]https://github.com/verimath/topology

Definition 18. (Coverage). A coverage is a function assigning to each Ob_C the family of morphisms $\{f_i : U_i \to U\}_{i\in I}$ called covering families, such that for any $g : V \to U$ exist a covering family $\{h : V_j \to V\}_{j\in J}$ such that each composite $h_j \circ g$ factors some f_i:

$$\begin{tikzcd} V_j \arrow[r, "k"] \arrow[d, "h"] & U_i \arrow[d, "f_i"] \\ V \arrow[r, "g"] & U \end{tikzcd}$$

```
Co (C: precategory) (cod: carrier C) : U
  = (dom: carrier C)
  * (hom C dom cod)

Delta (C: precategory) (d: carrier C) : U
  = (index: U)
  * (index -> Co C d)

Coverage (C: precategory): U
  = (cod: carrier C)
  * (fam: Delta C cod)
  * (coverings: carrier C -> Delta C cod -> U)
  * (coverings cod fam)
```

Definition 19. (Grothendieck Topology). Suppose category C has all pullbacks. Since C is small, a pretopology on C consists of families of sets of morphisms

$$\{\phi_\alpha : U_\alpha \to U\}, U \in \mathrm{C},$$

called covering families, such that following axioms hold: i) suppose that $\phi_\alpha : U_\alpha \to U$ is a covering family and that $\psi : V \to U$ is a morphism of C. Then the collection $V \times_U U_\alpha \to V$ is a cvering family for V. ii) If $\{\phi_\alpha : U_\alpha \to U\}$ is covering, and $\{\gamma_{\alpha,\beta} : W_{\alpha,\beta} \to U_\alpha\}$ is covering for all α, then the family of composites

$$W_{\alpha,\beta} \xrightarrow{\gamma_{\alpha,\beta}} U_\alpha \xrightarrow{\phi_\alpha} U$$

is covering; iii) The family $\{1 : U \to U\}$ is covering for all $U \in \mathrm{C}$.

Definition 20. (Site). Site is a category having either a coverage, grothendieck topology, or sieves.

```
site (C: precategory): U
  = (C: precategory) * Coverage C
```

Definition 21. (Presheaf). Presheaf of a category C is a functor from opposite category to category of sets: $\mathrm{C}^{op} \to \mathrm{Set}$.

```
presheaf (C: precategory): U
  = catfunctor (opCat C) Set
```

Definition 22. (Presheaf Category, **PSh**). Presheaf category **PSh** for a site C is category were objects are presheaves and morphisms are natural transformations of presheaf functors.

Definition 23. (Sheaf). Sheaf is a presheaf on a site. In other words a presheaf $F : \mathrm{C}^{op} \to \mathbf{Set}$ such that the cannonical map of inverse limit

$$F(U) \to \lim_{V \to U \in R}^{\leftarrow} F(V)$$

is an isomorphism for each covering sieve $R \subset Hom_C(\_, U)$. Equivalently, all induced functions

$$Hom_C(Hom_C(\_, U), F) \to Hom_C(R, F)$$

should be bejections.

```
sheaf (C: precategory): U
  = (S: site C)
  * presheaf S.1
```

Definition 24. (Sheaf Category, **Sh**). Sheaf category **Sh** is a category where objects are sheaves and morphisms are natural transformation of sheves. Sheaf category is a full subcategory of category of presheaves **PSh**.

Definition 25. (Grothendieck Topos). Topos is the category of sheaves $\mathbf{Sh}(C, J)$ on a site C with topology J.

Theorem 11. (Giraud). A category C is a Grothiendieck topos iff it has following properties: i) has all finite limits; ii) has small disjoint coproducts stable under pullbacks; iii) any epimorphism is coequalizer; iv) any equivalence relation $R \to E$ is a kernel pair and has a quotient; v) any coequalizer $R \to E \to Q$ is stably exact; vi) there is a set of objects that generates C.

Definition 26. (Geometric Morphism). Suppose that C and D are Grothendieck sites. A geometric morphism

$$f : \mathbf{Sh}(C) \to \mathbf{Sh}(D)$$

consist of functors $f_* : \mathbf{Sh}(C) \to \mathbf{Sh}(D)$ and $f^* : \mathbf{Sh}(D) \to \mathbf{Sh}(C)$ such that f^* is left adjoint to f_* and f^* preserves finite limits. The left adjoint f^* is called the inverse image functor, while f_* is called the direct image. The inverse image functor f^* is left and right exact in the sense that it preserves all finite colimits and limits, respectively.

Definition 27. (Cohesive Topos). A topos E is a cohesive topos over a base topos S, if there is a geometric morphism $(p^*, p_*) : E \to S$, such that: i) exists adjunction $p^! \vdash p_*$ and $p^! \dashv p_*$; ii) p^* and $p^!$ are full faithful; iii) $p_!$ preserves finite products.

This quadruple defines adjoint triple:

$$\int \dashv \flat \dashv \sharp$$

3.3 Elementary Topos

Giraud theorem was a synonymical topos definition involved only topos properties but not a site properties. That was step forward on predicative definition. The other step was made by Lawvere and Tierney, by removing explicit dependance on categorical model of set theory (as category of set is used in definition of presheaf). This information was hidden into subobject classifier which was well defined through categorical pullback and property of being cartesian closed (having lambda calculus as internal language).

Elementary topos doesn't involve 2-categorical modeling, so we can construct set topos without using functors and natural transformations (what we need in geometrical topos theory flavour). This flavour of topos theory more suited for logic needs rather that geometry, as its set properties are hidden under the predicative predicative pullback definition of subobject classifier rather that functorial notation of presheaf functor. So we can simplify proofs at the homotopy levels, not to lift everything to 2-categorical model.

Definition 28. (Monomorphism). An morphism $f : Y \to Z$ is a monic or mono if for any object X and every pair of parralel morphisms $g_1, g_2 : X \to Y$ the

$$f \circ g_1 = f \circ g_2 \to g_1 = g_2.$$

More abstractly, f is mono if for any X the $\mathrm{Hom}(X, \_)$ takes it to an injective function between hom sets $\mathrm{Hom}(X, Y) \to \mathrm{Hom}(X, Z)$.

```
mono (P: precategory) (Y Z: carrier P) (f: hom P Y Z): U
   = (X: carrier P) (g1 g2: hom P X Y)
  -> Path (hom P X Z) (compose P X Y Z g1 f)
                      (compose P X Y Z g2 f)
  -> Path (hom P X Y) g1 g2
```

Definition 29. (Subobject Classifier[?]). In category C with finite limits, a subobject classifier is a monomorphism $true : 1 \to \Omega$ out of terminal object 1, such that for any mono $U \to X$ there is a unique morphism $\chi_U : X \to \Omega$ and pullback diagram:

$$\begin{tikzcd} U \arrow[r, "k"] \arrow[d] & 1 \arrow[d, "true"] \\ X\Omega \arrow[r, "\chi_U"] & \Omega \end{tikzcd}$$

```
subobjectClassifier (C: precategory): U
  = (omega: carrier C)
  * (end: terminal C)
  * (trueHom: hom C end.1 omega)
  * (chi: (V X: carrier C) (j: hom C V X) -> hom C X omega)
  * (square: (V X: carrier C) (j: hom C V X) -> mono C V X j
    -> hasPullback C (omega,(end.1,trueHom),(X,chi V X j)))
  * ((V X: carrier C) (j: hom C V X) (k: hom C X omega)
    -> mono C V X j
    -> hasPullback C (omega,(end.1,trueHom),(X,k))
    -> Path (hom C X omega) (chi V X j) k)
```

Theorem 12. (Category of Sets has Subobject Classifier).

Definition 30. (Cartesian Closed Categories). The category C is called cartesian closed if exists all: i) terminals; ii) products; iii) exponentials. Note that this definition lacks beta and eta rules which could be found in embedding **MLTT**.

```
isCCC (C: precategory): U
  = (Exp:    (A B: carrier C) -> carrier C)
  * (Prod:   (A B: carrier C) -> carrier C)
  * (Apply:  (A B: carrier C) -> hom C (Prod (Exp A B) A) B)
  * (P1:     (A B: carrier C) -> hom C (Prod A B) A)
  * (P2:     (A B: carrier C) -> hom C (Prod A B) B)
  * (Term:   terminal C)
  * unit
```

Theorem 13. (Category of Sets is cartesian closed). As you can see from exp and pro we internalize Π and Σ types as SET instances, the isSet predicates are provided with contractability. Exitense of terminals is proved by propPi. The same technique you can find in **MLTT** embedding.

```
cartesianClosure : isCCC Set
  = (expo,prod,appli,proj1,proj2,term,tt) where
    exp (A B: SET): SET = (A.1    -> B.1, setFun A.1 B.1 B.2)
    pro (A B: SET): SET = (prod A.1 B.1, setSig A.1 (\(_ : A.1)
                          -> B.1) A.2 (\(_ : A.1) -> B.2))
    expo:  (A B: SET) -> SET = \(A B: SET) -> exp A B
    prod:  (A B: SET) -> SET = \(A B: SET) -> pro A B
    appli: (A B: SET) -> hom Set (pro (exp A B) A) B
        = \(A B: SET) -> \(x:(pro(exp A B)A).1)-> x.1 x.2
    proj1: (A B: SET) -> hom Set (pro A B) A
        = \(A B: SET) (x: (pro A B).1) -> x.1
    proj2: (A B: SET) -> hom Set (pro A B) B
        = \(A B: SET) (x: (pro A B).1) -> x.2
    unitContr (x: SET) (f: x.1 -> unit) : isContr (x.1 -> unit)
      = (f, \(z: x.1 -> unit) -> propPi x.1 (\(_:x.1)->unit)
           (\(x:x.1) -> propUnit) f z)
    term: terminal Set = ((unit,setUnit),
           \(x: SET) -> unitContr x (\(z: x.1) -> tt))
```

Note that rules of cartesian closure forms a type theoretical langage called lambda calculus.

Definition 31. (Elementary Topos). Topos is a precategory which is cartesian closed and has subobject classifier.

```
Topos (cat: precategory) : U
  = (cartesianClosure: isCCC cat)
  * subobjectClassifier cat
```

Theorem 14. (Topos Definitions). Any Grothendieck topos is an elementary topos too. The proof is sligthly based on results of Giraud theorem.

Theorem 15. (Category of Sets forms a Topos). There is a cartesian closure and subobject classifier for a categoty of sets.

```
internal : Topos Set
         = (cartesianClosure , hasSubobject)
```

Theorem 16. (Freyd). Main theorem of topos theory[?]. For any topos C and any $b : \mathrm{Ob}_C$ relative category $C \downarrow b$ is also a topos. And for any arrow $f : a \to b$ inverse image functor $f^* : C \downarrow b \to c \downarrow a$ has left adjoint $\sum_f$ and right adjoin $\prod_f$.

Conclusion

We gave here constructive definition of topology as finite unions and intersections of open subsets. Then make this definition categorically compatible by introducing Grothendieck topology in three different forms: sieves, coverage, and covering families. Then we defined an elementary topos and introduce category of sets, and proved that **Set** is cartesian closed, has object classifier and thus a topos.

This intro could be considered as a formal introduction to topos theory (at least of the level of first chapter) and you may evolve this library to your needs or ask to help porting or developing your application of topos theory to a particular formal construction.

Issue XXIX: Heterogeneous Equality

Maksym Sokhatskyi [1]

[1] National Technical University of Ukraine
Igor Sikorsky Kyiv Polytechnical Institute
30 квітня 2025 р.

Анотація

This paper represents the very small part of the developed base library for homotopical prover based on Cubical Type Theory (CTT) announced in 2017. We demonstrate the usage of this library by showing how to build a constructive proof of heterogeneous equality, the simple and elegant formulation of the equality problem, that was impossible to achieve in pure Martin-Löf Type Theory (MLTT). The machinery used in this article unveils the internal aspect of path equalities and isomorphism, used e.g. for proving univalence axiom, that became possible only in CTT. As an example of complex proof that was impossible to construct in earlier theories we took isomorphism between Nat and Fix Maybe datatypes and built a constructive proof of equality between elements of these datatypes. This approach could be extended to any complex isomorphic data types.

Keywords: Equivalence, Isomorphism, Homotopy, Heterogeneous Equality, Cubical Type Theory, Martin-Löf Type Theory

4 Heterogeneous Equalities

After formulating Type Theory to model quantifiers using Pi and $Sigma$ types in 1972 [?] Per Martin-Löf added Equ equality types in 1984 [?]. Later Equ types were extended to non-trivial structural higher equalities (∞-groupoid model) as was shown by Martin Hofmann and Thomas Streicher in 1996 [?]. However formal constructing of Equ type eliminators was made possible only after introducing Cubical Type Theory in 2017 [?]. CTT extends MLTT with interval $I = [0, 1]$ and its de Morgan algebra: $0, 1, r, -r, min(r, s), max(r, s)$ allowing constructive proofs of earlier models based on groupoid interpretation.

The problem and tools used. In this paper, we want to present the constructive formulation of proof that two values of different types are equal using constructive heterogeneous equality in Cubical Type Checker [?] [1]. In the end, we will use path isomorphism for that purposes [?].

Author's contribution. During the story of comparing two zeros, we will show the minimal set of primitives needed for performing this task in the cubical type checker. Most of them were impossible to derive in pure MLTT. We show these primitives in dependency order while constructing our proof. This article covers different topics in type theory, which is the main motivation to show how powerful the notion of equality is: 1) Contractability and n-Groupoids; 2) Constructive J; 3) Functional Extensionality; 4) Fibers and Equivalence; 5) Isomorphism; 6) Heterogeneous equality.

4.1 Problem Statement

As a formal description of the research includes all cubical programs as research object, type theory ingenral and MLTT and CTT inparticular as research subject, direct proof construction as logical method and encoded cubical base library and examples as research results.

Research object. The homotopy type theory base libraries in Agda, Cubical, Lean and Coq. While modern Agda has the cubical mode, Coq lacks of the computational semantics of path primitives while has HoTT library. The real programming language is not enough to develop the software and theorems, the language shoud be shipped with base library. In this article we unvail the practical implementation of base library for cubical typecheckers.

Research subject. We will analyze the base library through the needs of particular features, like basic theorems, heterogeneous path equalities, univalence, basic HITs like truncations, run-time versions of the list, nat, and stream datatypes. We use Martin-Löf Type Theory as a subject and its extention CTT. The main motivation is to have small and concise base library for cubicaltt type checker than can be used in more complex proofs. As an example of library usage we will show the general sketch of the constructive proofs of heterogeneous equality in CTT, concentrating only on homotopical features of CTT.

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

Research results. Research result is presented as source code repository that can be used by cubicaltt [2] language and contains the minimal base library used in this article. These primitives form a valuable part of base library, so this arcticle could be considered as an brief introduction to several modules: **proto_path**, **proto_equiv**, **pi**, **sigma**, **mltt**, **path**, **iso**. But the library has even more modules, that exceed the scope of this article so you may refer to source code repository[3]. Brief list of base library modules is given in Conclusion.

Research methods. The formal definition of MLTT theory and constructive implementation of its instance that supplied with minimal but comprehensive base library that can be used for verifying homotopical and run-time models. The type theory itself is a modeling language, logical framework and research method. The MLTT is a particular type theory with $Universes$, Π, Σ and Equ types. This is common denominator for a series of provers based on MLTT such as Coq, Agda, Cubicaltt. In the article we will use Cubical language.

5 MLTT Type Theory

MLTT is considered as contemporary common math modeling language for different parts of mathematics. Thanks to Vladimir Voevodsky this was extended to Homotopy Theory using MLTT-based Coq proof assistant [4]. Also he formulated the univalence principle $Iso(A,B) = (A = B)$ [?], however constructive proof that isomorphism equals to equality and equivalences is possible only in Cubical Type Theory [?] (Agda and Cubical type checkers).

In this section we will briefly discribe the basic notion of MLTT, then will give a formal description of MLTT and the informal primitives of CTT. Only after that will start the proof of heterogeneous equality ended up with proofterm. MLTT consist of Π, Σ and Equ types living in a Hierarchy of universes $U_i : U_{i+1}$. We will also give an extended equality $HeteroEqu$ which is needed for our proof.

5.1 Syntax Notes

Types are the analogues of sets in ZFC, or objects in topos theory, or spaces in analisys. Types contains elements, or points, or inhabitans and it's denoted $a : A$ and there is definitional equality which usually built into type checker and compare normal forms.

$$a : A \qquad \text{(terms and types)}$$

$$x = [y : A] \qquad \text{(definitional equality)}$$

MLTT type theory with Pi and $Sigma$ types was formulated using natural deduction inference rules as a language. The inference rules in that language will be translated to cubicaltt in our article.

[2]http://github.com/mortberg/cubicaltt
[3]http://github.com/groupoid/infinity
[4]http://github.com/UniMath

$$\frac{(A:U_i)\ (B:A\to U_j)}{(x:A)\to B(x):U_{max(i,j)}} \qquad \text{(natural deduction)}$$

Equvalent definition in cubicaltt (which is inconstend $U:U$ but this won't affect correctness of our proof). Here we consider Π and Pi synonimically identical.

$$Pi\ (A:U)\ (B:A\to U):U=(x:A)\to B(x) \qquad \text{(cubicaltt)}$$

In article we will use the latter notation, the cubical syntax. The function name in cubical syntax is an inference rule name, everything from name to semicolon is context conditions, and after semicolon is a new consruction derived from context conditions. From semicolon to equality sign we have type and after equ sign we have the term of that type. If the types are treated as spaces then terms are points in these spaces.

According to MLTT each type has 4 sorts of inference rules: Formation, Introduction, Eliminators and Computational rules. Formation rules are formal definition of spaces while introduction rules are methods how to create points in these spaces. Introduction rules increase term size, while eliminators reduce term size. Computational rules always formulated as equations that represents reduction rules, or operational semantics.

5.2 Pi types

Pi types represent spaces of dependent functions. With *Pi* type we have one lambda constructor and one application eliminator. When B is not dependent on $x:A$ the *Pi* is just a non-dependent total function $A\to B$. *Pi* has one lambda function constructor, and its eliminator, the application [**?**, **?**, **?**, **?**, **?**, **?**].

$$Pi(A,B)=\prod_{x:A}B(x):U, \quad \lambda x.b:\prod_{x:A}B(x)$$

$$\prod_{f:\prod_{x:A}B(x)}\prod_{a:A}fa:B(a)$$

Here we formulate the math formula of *Pi* and its eliminators in cubical syntax as Pi. Note that backslash "\" in cubical syntax means λ function from math notation and has compatible lambda signature.

```
Pi (A:U) (B:A->U) : U = (x:A)->B(x)
lambda (A:U) (B:A->U) (a:A) (b:B(a)): A->B(a) = \ (x:A)->b
app (A:U) (B:A->U) (a:A) (f:A->B(a)): B(a) = f(a)
```

5.3 Sigma types

Sigma types represents a dependent cartesian products. With sigma type we have pair constructor and two eliminators, its first and second projections. When B is not dependent on $x:A$ the *Sigma* is just a non-dependent product $A\times B$.

Sigma has one pair constructor and two eliminators, its projections [?, ?, ?, ?, ?, ?].

$$Sigma(A,B) = \sum_{x:A} B(x) : U, \quad (a,b) : \sum_{x:A} B(x)$$

$$\pi_1 : \prod_{f:\sum_{x:A} B(x)} A\ , \quad \pi_2 : \prod_{f:\sum_{x:A} B(x)} B(\pi_1(f))$$

As *Pi* and *Sigma* are dual the *Sigma* type could be formulated in terms of *Pi* type using Church encoding, thus *Sigma* is optional. The type systems which contains only *Pi* types called Pure or PTS. Here we rewrite the math formula of *Sigma* and its eliminators in cubical syntax as Sigma:

```
Sigma (A:U) (B:A->U): U = (x:A) * B(x)
pair (A:U) (B:A->U) (a: A) (b: B(a)): Sigma A B = (a,b)
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
```

5.4 Equ types

For modeling propositional equality later in 1984 was introduced *Equ* type. [?] However unlike *Pi* and *Sigma* the eliminator J of *Equ* type is not derivable in MLTT [?, ?, ?].

$$Equ(x,y) = \prod_{x,y:A} x =_A y : U, \quad reflect : \prod_{a:A} a =_A a$$

$$D : \prod_{x,y:A}^{A:U_i} x =_A y \to U_{i+1}, \quad J : \prod_{C:D} \prod_{x:A} C(x,x,reflect(x)) \to \prod_{y:A} \prod_{p:x=_A y} C(x,y,p)$$

Eliminator of Equality has complex form and underivable in MLTT. Here we can see the formulation of *Equ* in cubical syntax as Equ:

```
Equ      (A: U) (x y: A): U = undefined
reflect  (A: U) (a: A): Equ A a a = undefined
D        (A: U) : U = (x y: A) -> Equ A x y -> U
J        (A: U) (x y: A) (C: D A) (d: C x x (reflect A x))
                       (p: Equ A x y): C x y p = undefined
```

Starting from MLTT until cubicaltt there was no computational semantics for J rules and in Agda and Coq it was formulated using inductive data types wrapper around built-in primitives (J) in the core:

```
data Equality (A:U) (x y:A) = refl_ (_: Equ A x z)
reflection (A:U) (a:A): Equality A a a = refl_ (reflect A a)
```

Heterogeneous equality is needed for computational rule of *Equ* type. And also this is crucial to our main task, constructive comparison of two values of different types. We leave the definition blank until introdure cubical primitives, here is just MLTT signature of HeteroEqu which is undervable in MLTT.

```
HeteroEqu  (A B:U)(a:A)(b:B)(P:Equ U A B):U = undefined
```

E.g. we can define Setoid specification [?] as not-MLTT basis for equality types. These signatures are also underivable in MLTT.

$$symm : \prod_{a,b:A} \prod_{p:a=_A b} b =_A a, \quad transitivity : \prod_{a,b,c:A} \prod_{p:a=_A b} \prod_{q:b=_A c} a =_A c$$

```
sym   (A:U)(a b:A)(p:Equ A a b): Equ A b a = undefined
transitivity  (A:U)(a b c:A)(p: Equ A a b)(q: Equ A b c):
              Equ A a c = undefined
```

6 Preliminaries

6.1 Cubical Type Theory Primitives and Syntax

The path equality is modeled as an interval [0,1] with its de Morgan algebra 0, 1, r, min(r,s), max(r,s). According to underlying theory it has lambdas, application, composition and gluening of [0,1] interval and Min and Max functions over interval arguments. This is enought to formulate and prove path isomorphism and heterogeneous equality.

Heterogeneous Path. The HeteroPath formation rule defines a heterogenous path between elements of two types A and B for which Path exists A = B.

Abstraction over [0,1]. Path lambda abstraction is a function which is defined on $[0,1]$: $f : [0,1] \rightarrow A$. Path lambda abstraction is an element of Path type.

Min, Max and Invert. In the body of lambda abstraction besides path application de Morgan operation also could be used: $i \wedge j$, $i \vee j$, i, $-i$ and constants 0 and 1.

Application of path to element of [0,1]. When path lambda is defined, the path term in the body of the function could be reduced using lambda parameter applied to path term.

Path composition. The composition operation states that being extensible is preserved along paths: if a path is extensible at 0, then it is extensible at 1.

Path gluening. The path gluening is an operation that allows to build path from equivalences. CTT distinguishes types gluening, value gluening and ungluening.

Here we give LALR specification of BNF notation of Cubicat Syntax as implemented in our github repository [5]. It has only 5 keywords: **data**, **split**, **where**, **module**, and **import**.

```
def := data id tele =sum + id tele : exp =exp +
       id tele : exp where def
exp := cotele*exp + cotele→exp + exp→exp + (exp) + app + id +
```

[5]http://github.com/groupoid/infinity/

```
        (exp,exp) + \  cotele→exp + split cobrs + exp.1 + exp.2
  0  := #empty          imp    := [ import id ]
brs  := 0 + cobrs       tele   := 0 + cotele
app  := exp exp         cotele := ( exp : exp ) tele
 id  := [ #nat ]        sum    := 0 + id tele + id tele | sum
ids  := [ id ]          br     := ids → exp
cod  := def dec         mod    := module id where imp def
dec  := 0 + codec       cobrs  := | br brs
```

6.2 Contractability and Higher Equalities

A type A is contractible, or a singleton, if there is $a : A$, called the center of contraction, such that $a = x$ for all $x : A$: A type A is proposition if any x,y: A are equals. A type is a Set if all equalities in A form a prop. It is defined as recursive definition.

$$isContr = \sum_{a:A}\prod_{x:A} a =_A x,\ \ isProp(A) = \prod_{x,y:A} x =_A y,\ \ isSet = \prod_{x,y:A} isProp\,(x =_A y),$$

$$isGroupoid = \prod_{x,y:A} isSet\,(x =_A y),\ \ PROP = \sum_{X:U} isProp(X),\ \ SET = \sum_{X:U} isSet(X), ...$$

The following types are inhabited: isSet PROP, isGroupoid SET. All these functions are defined in **path** module. As you can see from definition there is a recurrent pattern which we encode in cubical syntax as follows:

```
data N =Z | S (n: N)
n_grpd (A: U) (n: N): U = (a b: A) -> ((rec A a b) n) where
  rec (A: U) (a b: A): (k: N) -> U = split
    Z -> Path A a b
    S n -> n_grpd (Path A a b) n

isContr (A: U): U = (x:A) * ((y: A) -> Equ A x y)
isProp       (A: U): U = n_grpd A Z
isSet        (A: U): U = n_grpd A (S Z)
isGroupoid   (A: U): U = n_grpd A (S (S Z))
PROP       : U = (X:U) * isProp X
SET        : U = (X:U) * isSet  X
GRPOUPOID  : U = (X:U) * isGroupoid X
```

6.3 Constructive J

The very basic ground of type checker is heterogeneous equality $PathP$ and contructive implementation of reflection rule as lambda over interval $[0, 1]$ that return constant value a on all domain.

```
Path (A:U)(a b:A):U = PathP (<i>A) a b
HeteroEqu (A B:U)(a:A)(b:B)(P:Equ U A B):U = Path P a b
refl (A:U)(a:A):Path A a a = <i> a
sym (A:U)(a b:A) (p: Path A a b): Path A b a = <i> p @ -i
transitivity (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 []
```

$$trans : \prod_{p:A=_U B}^{A,B:U} \prod_{a:A} B, \quad singleton : \prod_{x:A}^{A:U} \sum_{y:A} x =_A y$$

$$subst : \prod_{a,b:A}^{A:U,B:A\to U} \prod_{p:a=_A b} \prod_{e:B(a)} B(b), \quad congruence : \prod_{f:A\to B}^{A,B:U} \prod_{a,b:A} \prod_{p:a=_A b} f(a) =_B f(b)$$

Transport tranfers the element of type to another by given path equality of the types. Substitution is like transport but for dependent functions values: by given dependent function and path equality of points in the function domain we can replace the value of dependent function in one point with value in the second point. Congruence states that for a given function and for any two points in the function domain, that are connected, we can state that function values in that points are equal.

```
singl (A:U) (a:A): U = (x: A) * Path A a x
trans (A B:U) (p: Path U A B) (a: A): B = comp p a []
congruence (A B: U) (f:A->B) (a b: A)
           (p: Path A a b): Path B (f a) (f b)
           = <i> f (p @ i)

subst (A:U) (P:A->U) (a b: A)
      (p: Path A a b) (e: P a): P b
      = trans (P a) (P b) (congruence A U P a b p) e

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

Then we can derive J using *contrSingl* and *subst* as defined in HoTT[**?**]:

```
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 (x,refl A x)
          (y,p) (contrSingl A x y p) d where
        T (z:singl A x):U = C (z.1) (z.2)
```

These function are defined in **proto_path** module, and all of them except singleton definition are underivable in MLTT.

6.4 Functional Extensionality

Function extensionality is another example of underivable theorems in MLTT, it states if two functions with the same type and they always equals for any point from domain, we can prove that these function are equal. $funExt$ as functional property is placed in **pi** module.

$$funExt : \prod_{[f,g:(x:A)\to B(x)]}^{A:U,B:A\to U} \prod_{[x:A,p:A\to f(x)=_{B(x)}g(x)]} f =_{A\to B(x)} g$$

```
funExt (A: U) (B: A->U)
       (f g: (x:A)->B(x))
       (p: (x:A)->Path (B x) (f x) (g x)):
       Path ((y:A)->B y) f g=<i>\(a:A)->(p a)@i
```

6.5 Fibers and Equivalence

The fiber of a map $f : A \rightarrow B$ over a point $y : B$ is family over x of Sigma pair containing the point x and proof that $f(x) =_B y$.

$$fiber : \prod_{f:A->B}^{A,B:U} \prod_{x:A,y:B} \sum f(x) =_B y, \quad isEquiv : \prod_{f:A->B}^{A,B:U} \prod_{y:B} isContr(fiber(f,y))$$

$$equiv : \sum_{f:A->B}^{A,B:U} isEquiv(f) \quad pathToEquiv : \prod_{p:X=_U Y}^{X,Y:U} equiv_U(X,Y)$$
.

Contractability of fibers called isEquiv predicate. The Sigma pair of a function and that predicate called equivalence, or equiv. Now we can prove that singletons are contractible and write a conversion function $X =_U Y \rightarrow equiv(X,Y)$.

```
fiber    (A B:U)(f:A->B)(y:B):U = (x:A) * Path B y (f x)
isEquiv  (A B:U)(f:A->B):U = (y:B) -> isContr (fiber A B f y)
equiv    (A B:U):U = (f:A->B) * isEquiv A B f

singletonIsContractible (A:U) (a:A): isContr (singl A a)
    = ((a,refl A a), \ (z:(x:A) * Path A a x) ->
    contrSingl A a z.1 z.2)

pathToEquiv (A X: U) (p: Path U X A): equiv X A
    = subst U (equiv A) A X p (idEquiv A)
```

equiv type is compatible with cubicaltt typechecker and it instance can be passed as parameter for Glue operation. So all *equiv* functions and properties is placed in separate **equiv** module.

6.6 Isomorphism

The general idea to build path between values of different type is first to build isomorphism between types, defined as decode and encode functions (f and g), such that $f \circ g = id_A, g \circ f = id_B$.

$$Iso(A,B) = \sum_{[f:A\rightarrow B]} \sum_{[g:B\rightarrow A]} \left(\prod_{x:A} [g(f(x)) =_A x] \times \prod_{y:B} [f(g(y) =_B y] \right)$$

$$isoToEquiv(A,B) : Iso(A,B) \rightarrow Equiv(A,B)$$

$$isoToPath(A,B) : Iso(A,B) \rightarrow A =_U B$$

lemIso proof is a bit longread, you may refer to Github repository[6]. The by proof of *isoToEquiv* using *lemIso* we define *isoToPath* as Glue of A and B types, providing $equiv(A, B)$. Glue operation first appear in proving transprt values of different type across their path equalities which are being constructed using encode and decode functions that represent isomorphism. Also Glue operation appears in constructive implementation of Univalence axiom[**?**].

```
lemIso  (A B:U) (f: A->B) (g:B->A)
        (s: (y:B) -> Path B (f(g(y)))y)
        (t: (x:A) -> Path A (g(f(x)))x) (y:B) (x0 x1:A)
        (p0: Path B y (f(x0))) (p1: Path B y (f(x1))):
        Path (fiber A B f y) (x0,p0) (x1,p1) = undefined

isoToEquiv (A B: U) (f:A->B) (g:B->A)
        (s: (y:B) -> Path B (f(g(y))) y)
        (t: (x:A) -> Path A (g(f(x))) x): isEquiv A B f =
  \(y:B) -> ((g y,<i>s y@-i),\ (z:fiber A B f y) ->
    lemIso A B f g s t y (g y) z.1 (<i>s y@-i) z.2)

isoToPath (A B:U) (f:A->B)(g:B->A)
        (s: (y:B) -> Path B (f(g(y)))y)
        (t: (x:A) -> Path A (g(f(x)))x): Path U A B =
        <i> Glue B [(i=0)->(A,f,isoToEquiv A B f g s t),
                    (i=1)->(B,idfun B,idIsEquiv B) ]
```

Isomorphism definitions are placed in three different modules due to dependency optimisation: **iso**, **iso_pi**, **iso_sigma**. Latter two contains main theorems about paths in Pi and Sigma spaces.

7 The Formal Specification of the Problem

7.1 Class of Theorems. Constructive proofs of heterogeneous equalities

Speaking of core features of CTT that were unavailable in MLTT is a notion of heterogeneous equality that was several attempts to construct heterogeneous equalities: such as John-Major Equality [7] by Connor McBride (which is included in Coq base library). As an example of library usage we will show the general sketch of the constructive proofs of heterogeneous equality in CTT, concentrating only on homotopical features of CTT.

Let us have two types A and B. And we have some theorems proved for A and functions $f : A \to B$ and $g : B \to A$ such that $f \circ g = id_A$ and $g \circ f = id_B$. Then we can prove $Iso(A, B) \to Equ(A, B)$. The result values would be proof that elements of A and B are equal — $HeteroEqu$. We will go from the primitives to the final proof. As an example we took Nat and Fix Maybe datatype and will prove $Iso(Nat, Fix(Maybe))$. And then we prove the $HeteroEqu(Nat, Fix(Maybe))$.

[6] http://github.com/groupoid/infinity/tree/master/priv/iso.ctt
[7] https://homotopytypetheory.org/2012/11/21/on-heterogeneous-equality/

7.2 Problem Example. Nat = Fix Maybe

Now we can prove $Iso(Nat, Fix(Maybe))$ and $Nat =_U Fix(Maybe)$. First we need to introduce datatypes $Nat, Fix, Maybe$ and then write encode and decode functions to build up an isomorphism. Then by using conversion from Iso to $Path$ we get the heterogeneous equality of values in Nat and $Fix(Maybe)$. We can build transport between any isomorphic data types by providing ecode and decode functions.

```
data fix (F:U->U) = Fix (point: F (fix F))
data nat          = zero     | suc  (n: nat)
data maybe (A:U)  = nothing  | just (a: A)

natToMaybe: nat -> fix maybe = split
     zero -> Fix nothing
     suc n -> Fix (just (natToMaybe n))

maybeToNat: fix maybe -> nat = split
     Fix m -> split nothing -> zero
                    just f -> suc (maybeToNat f)

natMaybeIso: (a: nat) ->
     Path nat (maybeToNat (natToMaybe a)) a = split
          zero -> <i> zero
          suc n -> <i> suc (natMaybeIso n @ i)

maybeNatIso: (a : fix maybe) ->
     Path (fix maybe) (natToMaybe (maybeToNat a)) a = split
          Fix m -> split nothing -> <i> Fix nothing
                         just f -> <i> Fix (just (maybeNatIso f @ i))

maybenat: Path U (fix maybe) nat
     = isoToPath (fix maybe) nat
                 maybeToNat natToMaybe
                 natMaybeIso maybeNatIso
```

The result term of equaluty between two zeros of Nat and Fix Maybe is given by isomorphism.

```
> HeteroEqu (fix maybe) nat (Fix nothing) zero maybenat

EVAL: PathP (<!0> Glue nat [ (!0 = 0) -> (fix (\(A : U) ->
maybe), (maybeToNat,(\(y : B) -> ((g y,<i> (s y) @ -i),
\(z : fiber A B f y) -> lemIso A B f g s t y (g y) z.1
(<i> (s y) @ -i) z.2)) (A = (fix (\(A : U) -> maybe)),
B = nat, f = maybeToNat, g = natToMaybe, s = natMaybeIso,
t = maybeNatIso))), (!0 = 1) -> (nat,((\(a : A) -> a)
(A = nat),(\(a : A) -> ((a,refl A a),\(z : fiber A A
(idfun A) a) -> contrSingl A a z.1 z.2)) (A = nat))) ])
(Fix nothing) zero
```

We admit that normalized (expanded) term has the size of one printed page. Inside it contains the encode and decode functions and identity proofs about their composition. So we can reconstruct everything up to homotopical primitives or replace the isomorphic encoding with arbitrary code.

8 Conclusion

At the moment only two provers that support CTT exists, this is Agda [?] and Cubical [?]. We developed a base library for cubical type checkers and described the approach of how to deal with heterogeneous equalities by the example of proving $Nat =_U Fix(Maybe)$.

Homotopical core in the prover is essential for proving math theorems in geometry, topology, category theory, homotopy theory. But it also useful for proving with fewer efforts even simple theorems like commutativity of Nat. By pattern matching on the edges to can specify continuous (homotopical) transformations of types and values across paths.

We propose a general-purpose base library for modeling math systems using univalent foundations and cubical type checker.

The amount of code needed for $Nat =_U Fix(Maybe)$ proof is around 400 LOC in modules.

The further development of base library implies: 1) extending run-time facilities; 2) making it useful for building long-running systems and processes; 3) implement the inductive-recursive model for inductive types (development of lambek module). The main aim is to bring homotopical univalent foundations for run-time systems and models. Our base library could be used as a first-class mathematical modeling tool or as a sandbox for developing run-time systems and proving its properties, followed with code extraction to pure type systems and/or run-time interpreters.

Issue XXVIII: Categories with Families

Максим Сохацький [1]

[1] Національний технічний університет України
Київський політехнічний інститут імені Ігоря Сікорського

30 квітня 2025 р.

Анотація

Категорійна семантика залежної теорії типів.

Ключові слова: теорія категорій, категорії з сім'ями, залежна теорія типів

9 Категорії з сім'ями

Тут подано короткий неформальний опис категорійної семантики залежної теорії типів, запропонований Пітером Диб'єром. Категоріальна абстрактна машина Диб'єра на Haskell описана тут[1].

9.1 Основні визначення

Definition 32 (Fam)**.** Категорія Fam — це категорія сімей множин, де об'єкти є залежними функціональними просторами $(x : A) \to B(x)$, а морфізми з доменом $\Pi(A, B)$ і кодоменом $\Pi(A', B')$ — це пари функцій $\langle f : A \to A', g(x : A) : B(x) \to B'(f(x))\rangle$.

Definition 33 (Π-похідність)**.** Для контексту Γ і типу A позначимо $\Gamma \vdash A = (\gamma : \Gamma) \to A(\gamma)$.

Definition 34 (Σ-охоплення)**.** Для контексту Γ і типу A маємо $\Gamma; A = (\gamma : \Gamma) * A(\gamma)$. Охоплення не є асоціативним:

$$\Gamma; A; B \neq \Gamma; B; A$$

Definition 35 (Контекст)**.** Категорія контекстів C — це категорія, де об'єкти є контекстами, а морфізми — підстановками. Термінальний об'єкт $\Gamma = 0$ у C називається порожнім контекстом. Операція охоплення контексту $\Gamma; A = (x : \Gamma) * A(x)$ має елімінатори: $p : \Gamma; A \vdash \Gamma$, $q : \Gamma; A \vdash A(p)$, що задовольняють універсальну властивість: для будь-якого $\Delta : ob(C)$, морфізму $\gamma : \Delta \to \Gamma$ і терму $a : \Delta \to A$ існує єдиний морфізм $\theta = \langle \gamma, a\rangle : \Delta \to \Gamma; A$, такий що $p \circ \theta = \gamma$ і $q(\theta) = a$. Твердження: підстановка є асоціативною:

$$\gamma(\gamma(\Gamma, x, a), y, b) = \gamma(\gamma(\Gamma, y, b), x, a)$$

Definition 36 (CwF-об'єкт)**.** CwF-об'єкт — це пара $\Sigma(C, C \to Fam)$, де C — категорія контекстів з об'єктами-контекстами та морфізмами-підстановками, а $T : C \to Fam$ — функтор, який відображає контекст Γ у C на сім'ю множин термів $\Gamma \vdash A$, а підстановку $\gamma : \Delta \to \Gamma$ — на пару функцій, що виконують підстановку γ у термах і типах відповідно.

Definition 37 (CwF-морфізм)**.** Нехай $(C, T) : ob(C)$, де $T : C \to Fam$. CwF-морфізм $m : (C, T) \to (C', T')$ — це пара $\langle F : C \to C', \sigma : T \to T'(F)\rangle$, де F — функтор, а σ — натуральна трансформація.

Definition 38 (Категорія типів)**.** Для CwF з об'єктами (C, T) і морфізмами $(C, T) \to (C', T')$, для заданого контексту $\Gamma \in Ob(C)$ можна побудувати категорію $Type(\Gamma)$ — категорію типів у контексті Γ, де об'єкти — множина типів у контексті, а морфізми — функції $f : \Gamma; A \to B(p)$.

[1] `https://www.cse.chalmers.se/~peterd/papers/Ise2008.pdf`

9.2 Семантика залежної теорії типів

Definition 39 (Терми та типи)**.** У CwF для контексту Γ терми $\Gamma \vdash a : A$ є елементами множини $A(\gamma)$, де $\gamma : \Gamma$. Типи $\Gamma \vdash A$ є об'єктами в $Type(\Gamma)$, а підстановка $\gamma : \Delta \to \Gamma$ діє на типи та терми через функтор T.

Theorem 17 (Композиція підстановок)**.** Підстановки в категорії контекстів C є асоціативними та мають одиницю (ідентичну підстановку). Формально, для $\gamma : \Delta \to \Gamma$, $\delta : \Theta \to \Delta$ і $\epsilon : \Gamma \to \Lambda$ виконується:

$$(\gamma \circ \delta) \circ \epsilon = \gamma \circ (\delta \circ \epsilon), \quad id_\Gamma \circ \gamma = \gamma, \quad \gamma \circ id_\Delta = \gamma.$$

Доведення. Асоціативність випливає з універсальної властивості охоплення контексту (Визначення 1.4). Для будь-яких γ, δ, ϵ композиція морфізмів у C відповідає послідовному застосуванню підстановок, що зберігає структуру контекстів. Ідентична підстановка id_Γ діє як нейтральний елемент, оскільки $p \circ id_\Gamma = id_\Gamma$ і $q(id_\Gamma) = q$. □

Definition 40 (Залежні типи)**.** Залежний тип у контексті Γ — це відображення $\Gamma \to Fam$, де для кожного $\gamma : \Gamma$ задається множина $A(\gamma)$. У категорії $Type(\Gamma)$ залежні типи є об'єктами, а морфізми між A і B — це функції $f : \Gamma; A \to B(p)$, що зберігають структуру підстановок.

Theorem 18 (Універсальна властивість залежних типів)**.** Для будь-якого контексту Γ, типу A і терму $a : \Gamma \vdash A$ існує унікальний морфізм $\theta : \Gamma \to \Gamma; A$, який задовольняє $p \circ \theta = id_\Gamma$ і $q(\theta) = a$. Це забезпечує коректність залежної типізації в CwF.

Доведення. За Визначенням 1.4, універсальна властивість охоплення контексту гарантує існування $\theta = \langle id_\Gamma, a \rangle$. Унікальність випливає з того, що будь-який інший морфізм θ' з тими ж властивостями ($p \circ \theta' = id_\Gamma$, $q(\theta') = a$) збігається з θ через єдиність композиції в C. □

9.3 Формалізація в Anders

Для формалізації CwF у Agda чи Lean необхідно визначити категорію C як запис із полями для об'єктів, морфізмів, композиції та ідентичності, а також функтор $T : C \to Fam$. Нижче наведено псевдокод для Anders[2]:

```
def algebra : U_1 := Σ
    -- a semicategory of contexts and substitutions:
    (Con: U)
    (Sub: Con → Con → U)
    (◊: Π (Γ Θ Δ : Con), Sub Θ Δ → Sub Γ Θ → Sub Γ Δ)
    (◊-assoc: Π (Γ Θ Δ Φ : Con) (σ: Sub Γ Θ) (δ: Sub Θ Δ)
        (ν: Sub Δ Φ), PathP (<_>Sub Γ Φ) (◊ Γ Δ Φ ν (◊ Γ Θ Δ δ σ))
                                          (◊ Γ Θ Φ (◊ Θ Δ Φ ν δ) σ))
    -- identity morphisms as identity substitutions:
    (id: Π (Γ : Con), Sub Γ Γ)
    (id-left: Π (Θ Δ : Con) (δ : Sub Θ Δ),
              Path (Sub Θ Δ) δ (◊ Θ Δ Δ (id Δ) δ))
    (id-right: Π (Θ Δ : Con) (δ : Sub Θ Δ),
               Path (Sub Θ Δ) δ (◊ Θ Θ Δ δ (id Θ)))
    -- a terminal oject as empty context:
    (•: Con)
    (ε: Π (Γ : Con), Sub Γ •)
    (•-η: Π (Γ: Con) (δ: Sub Γ •), Path (Sub Γ •) (ε Γ) δ)
    (Ty: Con → U)
    (_|_|^T: Π (Γ Δ : Con), Ty Δ → Sub Γ Δ → Ty Γ)
    (|id|^T: Π (Δ : Con) (A : Ty Δ), Path (Ty Δ) (_|_|^T Δ Δ A (id Δ)) A)
    (|◊|^T: Π (Γ Δ Φ: Con) (A : Ty Φ) (σ : Sub Γ Δ) (δ : Sub Δ Φ),
        PathP (<_>Ty Γ) (_|_|^T Γ Φ A (◊ Γ Δ Φ δ σ))
                        (_|_|^T Γ Δ (_|_|^T Δ Φ A δ) σ))
    -- a (covariant) presheaf on the category of elements as terms:
    (Tm: Π (Γ : Con), Ty Γ → U)
    (_|_|^t: Π (Γ Δ : Con) (A : Ty Δ) (B : Tm Δ A)
              (σ: Sub Γ Δ), Tm Γ (_|_|^T Γ Δ A σ))
    (|id|^t: Π (Δ : Con) (A : Ty Δ) (t: Tm Δ A),
            PathP (<i> Tm Δ (|id|^T Δ A @ i))
                  (_|_|^t Δ Δ A t (id Δ)) t)
    (|◊|^t: Π (Γ Δ Φ: Con) (A : Ty Φ) (t: Tm Φ A)
             (σ : Sub Γ Δ) (δ : Sub Δ Φ),
             PathP (<i> Tm Γ (|◊|^T Γ Δ Φ A σ δ @ i))
                   (_|_|^t Γ Φ A t (◊ Γ Δ Φ δ σ))
          (_|_|^t Γ Δ (_|_|^T Δ Φ A δ) (_|_|^t Δ Φ A t δ) σ))
```

Ця структура дозволяє реалізувати Визначення 1.1–1.11, а Теореми 1.10 і 1.12 доводяться через перевірку асоціативності та універсальних властивостей.

[2]https://anders.groupoid.space/lib/mathematics/categories/meta/kraus.anders

9.4 Висновки

Категорії з сім'ями (CwF) є потужним інструментом для моделювання залежної теорії типів. Вони забезпечують чітку семантику для контекстів, підстановок і залежних типів, що полегшує аналіз і формалізацію.