

Homotopy theory in type theory

Michael Shulman

11 April 2012

Review of type theory

- Type theory consists of rules for deriving typing judgments:

$$(x_1 : A_1), (x_2 : A_2), \dots, (x_n : A_n) \vdash (b : B)$$

- The rules come in “packages” called type constructors.
- Each type constructor has four groups of rules: formation, introduction, elimination, and computation.
- Categorically: types are objects, terms are morphisms.
- Each type constructor corresponds to a categorical universal property.

Dependent eliminators

When we introduce predicates and dependent types, the eliminators of other types need to be generalized.

Example

- Suppose $(z: A + B) \vdash (P(z): \text{Type})$ is a predicate on $A + B$.
- We should be able to prove P by cases.
 - ① Prove $(x: A) \vdash (p_A: P(\text{inl}(x)))$.
 - ② Prove $(y: B) \vdash (p_B: P(\text{inr}(y)))$.
 - ③ Conclude $(z: A + B) \vdash (\text{case}(z; p_A, p_B): P(z))$.
- This looks like the “case split” eliminator for $A + B$, but the output type $P(z)$ depends on the element z that we are case-analyzing.

Therefore: we strengthen the elimination rules.

Dependent eliminators

Before

Suppose A , B , and C are types.

If $(x: A) \vdash (c_A: C)$ and $(y: B) \vdash (c_B: C)$,
then for $p: A + B$ we have $\text{case}(p, c_A, c_B): C$.

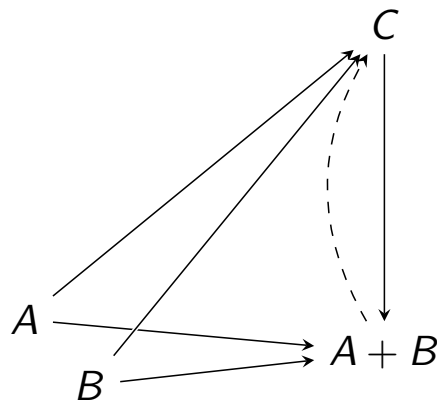
After

Suppose A and B are types, and

$$(z: A + B) \vdash (C(z): \text{Type})$$

is a dependent type.

If $(x: A) \vdash (c_A: C(\text{inl}(x)))$ and $(y: B) \vdash (c_B: C(\text{inr}(y)))$,
then for $p: A + B$ we have $\text{case}(p, c_A, c_B): C(p)$.



Dependent eliminators imply uniqueness

Theorem

Suppose $f, g: C^{A+B}$ and that

- for all $a: A$, we have $f(\text{inl}(a)) = g(\text{inl}(a))$, and
- for all $b: B$, we have $f(\text{inr}(b)) = g(\text{inr}(b))$.

Then for all $z: A + B$, we have $f(z) = g(z)$.

Proof.

Consider the dependent type

$$(z: A + B) \vdash (f(z) = g(z) : \text{Type})$$

By the dependent eliminator for $A + B$, to construct a term of this type, it suffices to construct terms

$$\begin{aligned} (a: A) &\vdash (e_A : f(\text{inl}(a)) = g(\text{inl}(a))) \\ (b: B) &\vdash (e_B : f(\text{inr}(b)) = g(\text{inr}(b))) \end{aligned}$$

□

Function extensionality

It's more difficult to give a dependent eliminator for function types. Instead, we assert **function extensionality** directly as an axiom.

$$(f, g : B^A) \vdash (\text{funext} : (\prod_{x : A} (f(x) = g(x))) \rightarrow (f = g))$$

Remarks

- Today I'll use both B^A and $A \rightarrow B$ for the function type.
- **Later:** more homotopical versions of both kinds of uniqueness.

Equality types

Equality types (or **identity types**) are a “positive type” (determined by the introduction rule):

- ① For any type A and $a : A$ and $b : A$, there is a type $(a = b)$.
- ② For any $a : A$, we have $\text{refl}_a : (a = a)$.
- ③ Suppose $C(x, y, p)$ is a type dependent on three variables $x, y : A$ and $p : (x = y)$. Suppose moreover that for any $x : A$ we have an element $d(x) : C(x, x, \text{refl}_x)$. Then for any x, y, p we have an element $J(d; x, y, p) : C(x, y, p)$.
- ④ $J(d; a, a, \text{refl}_a)$ computes to $d(a)$.

Informally, ③ says

Elimination on equality

In order to do something with an arbitrary $p : (x = y)$, it suffices to consider the case of $\text{refl}_x : (x = x)$.

Equality is symmetric

Theorem

Suppose $p: (x = y)$. Then $p^{-1}: (y = x)$.

Proof.

By elimination, we may assume that p is $\text{refl}_x: (x = x)$. But in this case, we can take p^{-1} to also be $\text{refl}_x: (x = x)$. \square

Just as in the cases of the dependent eliminator for coproducts, the desired conclusion $C(z)$ becomes $C(\text{inl}(a))$ and $C(\text{inr}(b))$, when we eliminate p the desired conclusion $(y = x)$ becomes $(x = x)$.

Equality is transitive

Theorem

Suppose $p: (x = y)$ and $q: (y = z)$. Then $p * q: (x = z)$.

Proof.

By elimination, we may assume that p is $\text{refl}_x: (x = x)$. But in this case, we have $q: (x = z)$, so we can take $p * q$ to be just q . \square

We could equally well have eliminated q , or both p and q .

Paths

We treat **types** as spaces/ ∞ -groupoids/homotopy types, and we think of terms $p: (x = y)$ as **paths** $x \rightsquigarrow y$.

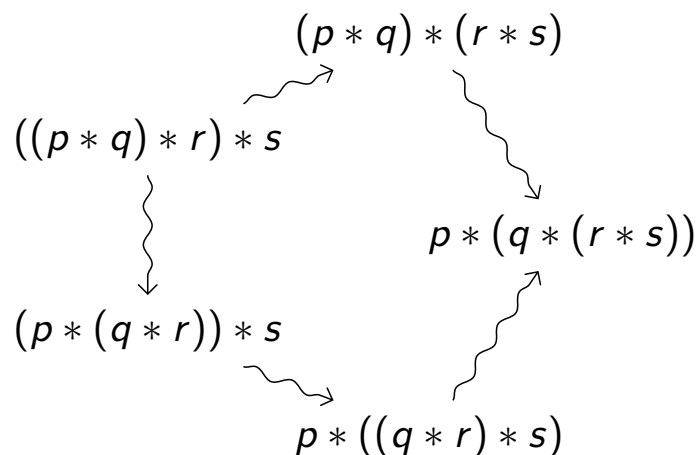
- Reflexivity becomes the **constant path** $\text{refl}_x: x \rightsquigarrow x$.
- Transitivity becomes **concatenation** $x \xrightarrow{p*q} z$ of $x \xrightarrow{p} y \xrightarrow{q} z$.
- Symmetry becomes **reversal** $y \xrightarrow{p^{-1}} x$ of $x \xrightarrow{p} y$.

But now there is more to say.

- Concatenation is **associative**: $\alpha_{p,q,r} : ((p * q) * r = p * (q * r))$.

2-paths

The “associator” $\alpha_{p,q,r}$ is coherent:

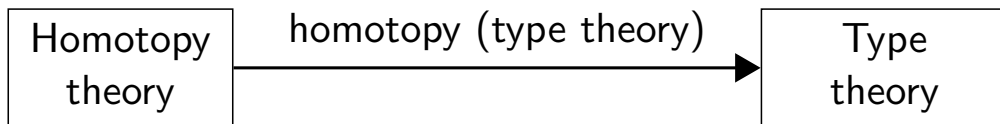


... or more precisely, there is a **path** between those two concatenations...

... which then has to be coherent...

Theorem (Lusmdaine, Garner–van den Berg)

The terms belonging to the iterated identity types of any type A form an ∞ -groupoid.



Note: Uses Batanin-Leinster ∞ -groupoids (can also be done with simplicial versions).

Mapping on paths

Given $f: A \rightarrow B$, $x, y: A$, and a path $p: (x = y)$, we have an **image path**

$$\text{map}(f, p) : (f(x) = f(y))$$

defined by eliminating on p :

- If p is refl_x , then $\text{map}(f, p) := \text{refl}_{f(x)}$.

Transporting along paths

Given $x, y: A$, $p: (x = y)$, and B dependent on A , we have the operation of **transporting along p**

$$\text{trans}(p, -) : B(x) \rightarrow B(y).$$

defined by eliminating on p :

- If p is refl_x , then $\text{trans}(p, -)$ is the identity map of $B(x)$.

Interpretation

We should view the map $B \rightarrow A$ as a **fibration**.

(In an $(\infty, 1)$ -category, we can treat any map as a fibration.)

Paths for type constructors

For any type built using a type constructor, we can characterize its paths in terms of paths in its input types.

Example (Cartesian products)

- From $p: (a_1 = a_2)$ and $q: (b_1 = b_2)$, we can build

$$(p, q) : ((a_1, b_1) = (a_2, b_2))$$

- Given $z_1, z_2: A \times B$ and $r: (z_1 = z_2)$, we have

$$\text{map}(\text{fst}, r) : (\text{fst}(z_1) = \text{fst}(z_2))$$

$$\text{map}(\text{snd}, r) : (\text{snd}(z_1) = \text{snd}(z_2))$$

Paths in dependent sums

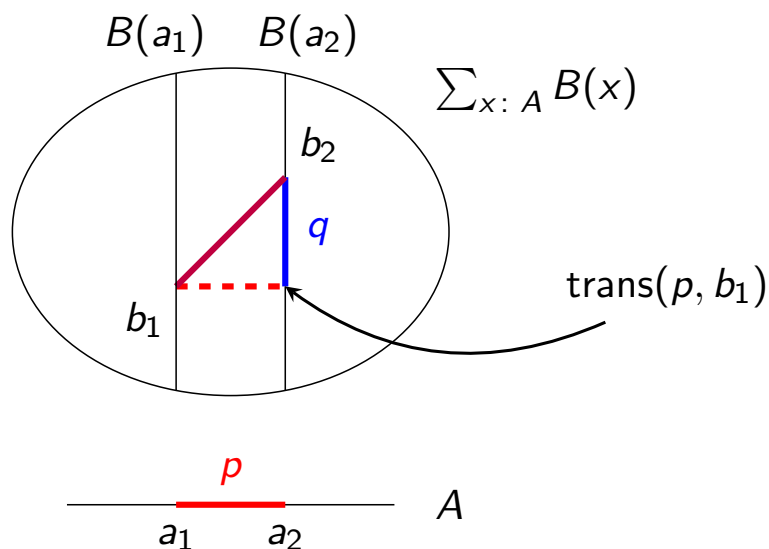
Suppose $a_1, a_2: A$ and $b_1: B(a_1)$ and $b_2: B(a_2)$. A path

$$(a_1, b_1) = (a_2, b_2)$$

in $\sum_{x:A} B(x)$ should consist of

- A path $p: (a_1 = a_2)$ in A , and...
- what?
 - The expression $(b_1 = b_2)$ is ill-formed, since b_1 and b_2 have different types.
 - Instead we can use $q: (\text{trans}(p, b_1) = b_2)$.

Paths in dependent sums



- In a fibration, we can lift the path p starting at b_1 .
- We choose one lift and call its endpoint $\text{trans}(p, b_1)$.
- Any **other** lift of p is determined by a path in the fiber $B(a_2)$.

Subsingletons in homotopy theory

Recall that **logic** is type theory restricted to subsingletons.

In homotopy type theory, we interpret “subsingleton” homotopically:

Theorem

For an object P in an $(\infty, 1)$ -category with products, TFAE:

- ① Each space $\text{Hom}(X, P)$ is empty or contractible.
- ② Any two morphisms $X \rightrightarrows P$ are homotopic.
- ③ The diagonal $P \rightarrow P \times P$ has a section.
- ④ The diagonal $P \rightarrow P \times P$ is an equivalence.

h-Propositions

Definition

A type P is a **proposition** (or **h-proposition** or **h-prop**) if we have

$$(x : P), (y : P) \vdash (p : (x = y))$$

$$\begin{array}{c} P \\ \downarrow \\ P \times P \end{array}$$

These are the “subsingletons” of homotopy type theory.

Building h-props

What ways do we have to obtain h-props?

- Most type constructors preserve h-props.
- For others ($+$ and \sum), we intend to apply “support”.
- $(x = y)$ is **not** generally an h-prop, but has a support:
 - $(x = y)$ is the **type of paths** from x to y .
 - $\text{supp}(x = y)$ is the assertion: **there exists a path** from x to y .
- For **some** types A , all equalities $(x = y)$ **are** h-props.
 - These are called **sets** or **h-sets**.
 - Certain types are always sets (e.g. \mathbb{N} , on Friday).
- But can we say anything **homotopy-theoretic** with this logic?

Internalizing h-props

How can we say **in type theory** “ A is an h-prop”?

$$\text{isProp}(A) := \text{supp} \left(\prod_{x:A} \prod_{y:A} (x = y) \right) \quad ?$$

Internalizing h-props

How can we say **in type theory** “ A is an h-prop”?

$$\text{isProp}(A) := \prod_{x:A} \prod_{y:A} (x = y) \quad !$$

This is already an h-prop!

Theorem

For any A , we can construct a term in

$$\text{isProp}(\text{isProp}(A)).$$

Internalizing h-props

Theorem

For any A , $\text{isProp}(\text{isProp}(A))$.

Proof.

- Suppose $H, K : \text{isProp}(A)$; we want $(H = K)$.
- By funext, suffices to show $H(x, y) = K(x, y)$ for all $x, y : A$.
- Now $\text{map}(K(x, -), H(x, y))$ is a path in $\sum_z (x = z)$ from $K(x, x)$ to $K(x, y)$. In particular, it contains a path

$$\text{trans}(H(x, y), K(x, x)) = K(x, y)$$

- Hence $H(x, y) * K(x, x) = K(x, y)$ (a fact).
- It suffices to prove $K(x, x) = \text{refl}_x$.
 - The above argument with H being K , and y being x , yields $K(x, x) * K(x, x) = K(x, x)$.
 - Now cancel $K(x, x)$ (i.e. concatenate with $K(x, x)^{-1}$). □

Some subtleties

- We can loosely read $\prod_{x:A} \prod_{y:A} (x = y)$ as “for all $x, y : A$, we have a path $(x = y)$ ”
- But “for all $x, y : A$, **there exists** a path $(x = y)$ ” should be read to mean

$$\prod_{x:A} \prod_{y:A} \text{supp}(x = y)$$

This asserts that “if A is nonempty, then it is **connected**.”

- In $\prod_{x:A} \prod_{y:A} (x = y)$, the assigned path $(x = y)$ must **depend continuously** on x and y . This can be confusing until you get used to this meaning of “for all”.

Some subtleties

- Type theory is a formal system.
- We can and do (and must, in practice) use informal language to speak and think about it.
- This depends on certain conventions about the formal interpretation given to informal words, which are sometimes subtly different to those used for some other formal system (like set theory).
- Fortunately, we have a computer proof assistant to type-check our proofs and guarantee that we didn't screw up!

Homotopy equivalences

Definition

A function $f: A \rightarrow B$ is a **homotopy equivalence** if there exists $g: B \rightarrow A$ and homotopies $g \circ f \sim \text{id}_A$ and $f \circ g \sim \text{id}_B$.

$$\text{isHtpyEquiv}(f) := \text{supp} \left(\sum_{g: B \rightarrow A} \left((g \circ f = \text{id}_A) \times (f \circ g = \text{id}_B) \right) \right)$$

This would not be an h-prop without `supp`. Can we avoid it?

Back to bijections

A function $f: A \rightarrow B$ between sets is a **bijection** if

- ① There exists $g: B \rightarrow A$ such that $g \circ f = \text{id}_A$ and $f \circ g = \text{id}_B$.
- ② OR: For each $b \in B$, the set $f^{-1}(b)$ is a singleton.
- ③ OR: There exists $g: B \rightarrow A$ such that $g \circ f = \text{id}_A$ and also $h: B \rightarrow A$ such that $f \circ h = \text{id}_B$.

Voevodsky equivalences

Definitions

The **homotopy fiber** of $f: A \rightarrow B$ at $b: B$ is

$$\text{hfiber}(f, b) := \sum_{x: A} (f(x) = b).$$

A type X is **contractible** if it is an inhabited h-prop:

$$\text{isContr}(X) := \text{isProp}(X) \times X$$

Definition (Voevodsky)

f is an **equivalence** if each $\text{hfiber}(f, b)$ is contractible:

$$\text{isEquiv}(f) := \prod_{b: B} \text{isContr}(\text{hfiber}(f, b))$$

This is an h-prop.

H-isomorphisms

Definition (Joyal)

$f: A \rightarrow B$ is an **h-isomorphism** if we have $g: B \rightarrow A$ and a homotopy $g \circ f \sim \text{id}_A$, and also $h: B \rightarrow A$ and a homotopy $f \circ h \sim \text{id}_B$.

$$\text{isHlso}(f) := \left(\sum_{g: B \rightarrow A} (g \circ f = \text{id}_A) \right) \times \left(\sum_{h: B \rightarrow A} (f \circ h = \text{id}_B) \right)$$

This is also an h-prop.

Adjoint equivalences

Given a homotopy equivalence, we can also ask for more coherence from $r: (g \circ f = \text{id}_A)$ and $s: (f \circ g = \text{id}_B)$.

(1a) For all $b: B$, we have $u(b): (r(g(b)) = \text{map}(g, s(b)))$.

(1b) For all $a: A$, we have $v(a): (\text{map}(f, r(a)) = s(f(a)))$.

(2a) For all $b: B$, we have $\dots v(g(b)) \dots \text{map}(g, u(b)) \dots$

(2b) For all $a: A$, we have $\dots u(f(a)) \dots \text{map}(f, v(a)) \dots$

\vdots

This gives an h-prop if we stop between any (na) and (nb) (and then the rest can be constructed).

Definition

f is an **adjoint equivalence** if we have g, r, s , and u .

$$\text{isAdjEquiv}(f) := \sum_{g: B \rightarrow A} \sum_{r: \dots} \sum_{s: \dots} (r(g(b)) = \text{map}(g, s(b)))$$

All equivalences are the same

Theorem

The following are equivalent:

- ① f is a homotopy equivalence.
- ② f is a (Voevodsky) equivalence.
- ③ f is a (Joyal) h-isomorphism.
- ④ f is an adjoint equivalence.

The last three are supp-free h-props, so we have equivalences

$$\text{isEquiv}(f) \simeq \text{isHlso}(f) \simeq \text{isAdjEquiv}(f)$$

Definition

The **type of equivalences** between $A, B: \text{Type}$ is

$$\text{Equiv}(A, B) := \sum_{f: A \rightarrow B} \text{isEquiv}(f).$$

The short five lemma

$$\begin{array}{ccccc} \text{hfiber}(f) & \longrightarrow & A & \xrightarrow{f} & B \\ & & \downarrow r & & \downarrow t \\ & & \text{hfiber}(g) & \longrightarrow & C & \xrightarrow{g} & D \end{array}$$

Theorem

- If s and t are equivalences, so is r .
- If r and t are equivalences, so is s .

This is a theorem **in type theory**: A, B, C, D are types and we have a proof term

$$(p_1 : \text{isEquiv}(s)), (p_2 : \text{isEquiv}(t)) \vdash (q : \text{isEquiv}(r))$$

The 3×3 lemma

$$\begin{array}{ccccc} & & \text{hfiber}(h) & \xrightarrow{r} & \text{hfiber}(k) \\ & & \downarrow & & \downarrow \\ \text{hfiber}(f) & \longrightarrow & A & \xrightarrow{f} & B \\ & & \downarrow h & & \downarrow k \\ \text{hfiber}(g) & \longrightarrow & C & \xrightarrow{g} & D \end{array}$$

Theorem

There is an equivalence $\text{hfiber}(r) \simeq \text{hfiber}(s)$.

(Also a theorem **in type theory**.)

Homotopical uniqueness

Theorem

For any types A, B, C , the map

$$\lambda f. (\lambda a. f(\text{inl}(a)), \lambda b. f(\text{inr}(b))) : C^{A+B} \rightarrow C^A \times C^B$$

is an equivalence (using the dependent eliminator).

The type $C^{A+B} \rightarrow C^A \times C^B$ should be more consistently (but less legibly) written:

$$(C^A \times C^B)^{C^{A+B}} \quad \text{or} \quad ((A + B) \rightarrow C) \rightarrow ((A \rightarrow C) \times (B \rightarrow C))$$

Awodey–Gambino–Sojakova have proven a much more general version of this, in the context we'll discuss on Friday.

Homotopical function extensionality

For $f, g : B^A$, there is a term

$$\text{happly} : \left((f = g) \rightarrow \prod_{a:A} (f(a) = g(a)) \right)$$

defined by identity elimination:

$$\text{happly}(\text{refl}_f) := \lambda a. \text{refl}_{f(a)}$$

Theorem (Voevodsky)

happly is an equivalence (using the naive funext).

Also works for dependent functions.

Paths in the universe

The only type whose path-types we have not determined (up to equivalence, in terms of other path-spaces) is the universe “Type”.

$$\begin{array}{ccc} B & \longrightarrow & \widetilde{\text{Type}} \\ \downarrow & \lrcorner & \downarrow \\ A & \longrightarrow & \text{Type} \end{array}$$

If Type is the “classifying space” of types, then a path in Type should be an **equivalence** of types.

The univalence axiom

For $A, B : \text{Type}$, we have

$$\text{pathToEquiv}_{A,B} : ((A = B) \rightarrow \text{Equiv}(A, B))$$

defined by identity elimination.

Note: $(A = B)$ is a path-type of “Type”.

The Univalence Axiom (Voevodsky)

For all A, B , the function $\text{pathToEquiv}_{A,B}$ is an equivalence.

$$\prod_{A : \text{Type}} \prod_{B : \text{Type}} \text{isEquiv}(\text{pathToEquiv}_{A,B})$$

In particular, every equivalence yields a path between types.

The meaning of univalence

The meaning of univalence

Given an equivalence $f: A \xrightarrow{\sim} B$, we can **identify A with B along f** .

In other words:

- When talking about A , B , and f , we “may as well assume” that B is A , and f is 1_A .
- Or: equivalent types can be treated as identical.

Proof.

Use the inverse of `pathToEquiv`, then the eliminator of equality. \square

This is something we do informally all the time in mathematics. The univalence axiom gives it a **precise** formal expression.

The uses of univalence

- ① The homotopy theory is nontrivial (Type is not an h-set).
- ② (Voevodsky) Univalence implies `funext`.
- ③ For any type F , the type

$$\sum_{A: \text{Type}} \text{supp}(A = F)$$

is the classifying space for bundles with fiber F .

- ④ Computing homotopy groups! (on Friday)
- ⑤ Many more ...