Homotopy theory in type theory

Michael Shulman

11 April 2012

Review of type theory

- Type theory consists of rules for deriving typing judgments:

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

- 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.

Outline

Dependent eliminators

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

Example

- Suppose $(z\colon A + B) \vdash (P(z)\colon \mathrm{Type})$ is a predicate on $A + B$.
- We should be able to prove P by cases.
  1. Prove $(x\colon A) \vdash (p_A\colon P(\mathrm{inl}(x)))$.
  2. Prove $(y\colon B) \vdash (p_B\colon P(\mathrm{inr}(y)))$.
  3. Conclude $(z\colon A + B) \vdash (\mathrm{case}(z; p_A, p_B)\colon P(z))$.

Dependent eliminators

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

Example

- Suppose $(z\colon A + B) \vdash (P(z)\colon \text{Type})$ is a predicate on $A + B$.
- We should be able to prove P by cases.
  1. Prove $(x\colon A) \vdash (p_A\colon P(\text{inl}(x)))$.
  2. Prove $(y\colon B) \vdash (p_B\colon P(\text{inr}(y)))$.
  3. Conclude $(z\colon A + B) \vdash (\text{case}(z; p_A, p_B)\colon P(z))$.
- 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.

Dependent eliminators

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

Example

- Suppose $(z\colon A+B) \vdash (P(z)\colon \mathsf{Type})$ is a predicate on $A+B$.
- We should be able to prove P by cases.
  1. Prove $(x\colon A) \vdash (p_A\colon P(\mathsf{inl}(x)))$.
  2. Prove $(y\colon B) \vdash (p_B\colon P(\mathsf{inr}(y)))$.
  3. Conclude $(z\colon A+B) \vdash (\mathsf{case}(z; p_A, p_B)\colon P(z))$.
- 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 $(x\colon A) \vdash (c_A : C)$ and $(y\colon B) \vdash (c_B : C)$,
then for $p\colon A + B$ we have $\mathrm{case}(p, c_A, c_B) : C$.

Dependent eliminators

Before

Suppose A, B, and C are types.

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

After

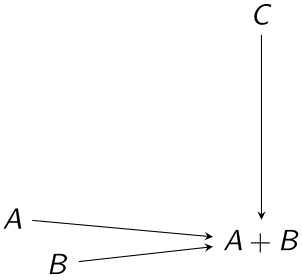
Suppose A and B are types, and

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

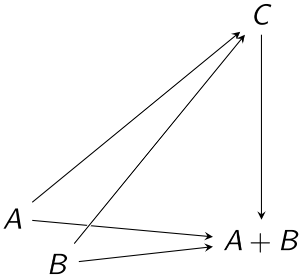
is a dependent type.

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

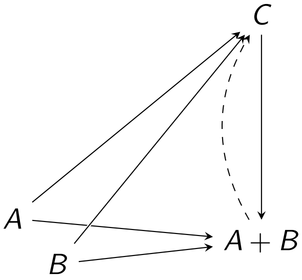
Dependent eliminators in categories



Dependent eliminators in categories



Dependent eliminators in categories



Dependent eliminators imply uniqueness

Theorem

Suppose $f, g \colon C^{A+B}$ *and that*

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

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

Dependent eliminators imply uniqueness

Theorem

Suppose $f, g \colon C^{A+B}$ *and that*

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

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

Proof.

Consider the dependent type

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

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

$$(a \colon A) \ \vdash \ (e_A : f(\mathrm{inl}(a)) = g(\mathrm{inl}(a)))$$
$$(b \colon B) \ \vdash \ (e_B : f(\mathrm{inr}(b)) = g(\mathrm{inr}(b)))$$

□

Interlude

(Coq)

Function extensionality

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

$$\Big(f, g \colon B^A\Big) \;\vdash\; \Big(\mathsf{funext} : \Big(\textstyle\prod_{x \colon A}(f(x) = g(x))\Big) \to (f = g)\Big)$$

Function extensionality

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

$$\left(f, g : B^A\right) \vdash \left(\mathsf{funext} : \left(\prod_{x : A}(f(x) = g(x))\right) \to (f = g)\right)$$

Remarks

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

Outline

Equality types

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

1. For any type A and $a\colon A$ and $b\colon A$, there is a type $(a = b)$.

Equality types

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

1. For any type A and $a\colon A$ and $b\colon A$, there is a type $(a = b)$.
2. For any $a\colon A$, we have $\mathsf{refl}_a : (a = a)$.

Equality types

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

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

Equality types

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

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

Equality types

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

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

Informally, 3 says

Elimination on equality

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

Equality is symmetric

Theorem

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

Proof.

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

Equality is symmetric

Theorem

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

Proof.

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

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

Equality is transitive

Theorem

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

Proof.

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

Equality is transitive

Theorem

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

Proof.

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

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

Interlude

(Coq)

Paths

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

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

Paths

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

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

But now there is more to say.

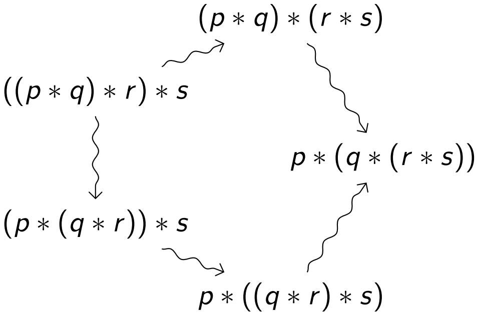
- Concatenation is associative: $\alpha_{p,q,r} : ((p * q) * r = p * (q * r))$.

Interlude

(Coq)

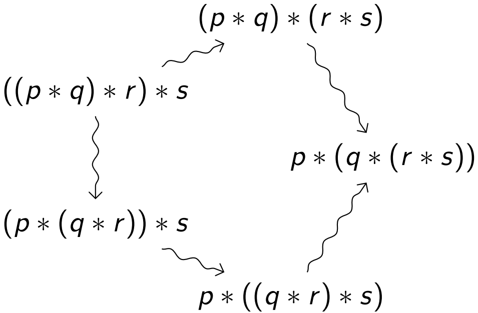
2-paths

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



2-paths

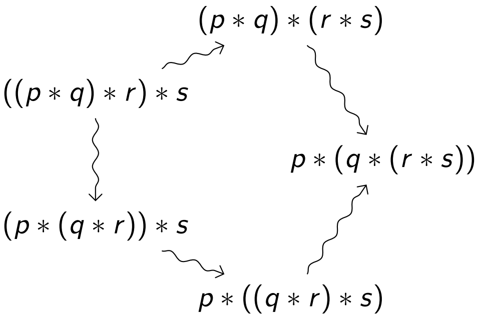
The “associator” $\alpha_{p,q,r}$ is coherent:



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

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. . .

∞-groupoids

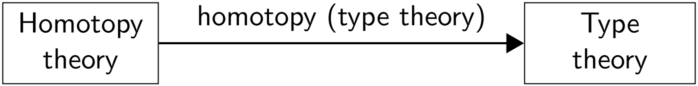
Theorem (Lusmdaine,Garner–van den Berg)

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

∞-groupoids

Theorem (Lusmdaine,Garner–van den Berg)

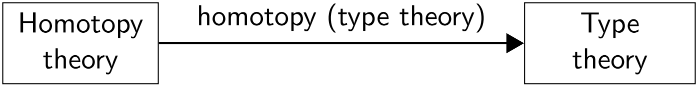
The terms belonging to the iterated identity types of any type A form an ∞-groupoid.



∞-groupoids

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 $f\colon A \to B$, $x, y\colon A$, and a path $p\colon (x = y)$, we have an image path

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

defined by eliminating on p:

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

Transporting along paths

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

$$\mathsf{trans}(p, -) : B(x) \to B(y).$$

defined by eliminating on p:

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

Transporting along paths

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

$$\mathsf{trans}(p, -) : B(x) \to B(y).$$

defined by eliminating on p:

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

Interpretation

We should view the map $B \to A$ 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 $p\colon (a_1 = a_2)$ and $q\colon (b_1 = b_2)$, we can build

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

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 $p\colon (a_1 = a_2)$ and $q\colon (b_1 = b_2)$, we can build

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

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

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

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

Paths in dependent sums

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

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

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

- A path $p \colon (a_1 = a_2)$ in A, and...
- what?

Paths in dependent sums

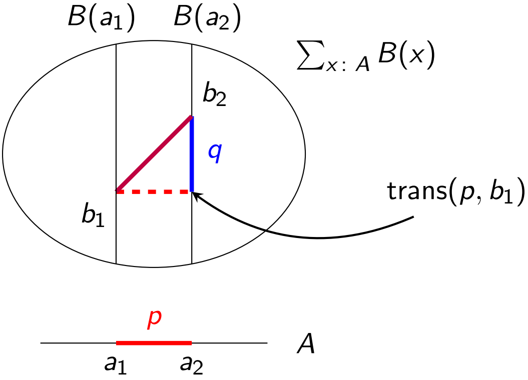
Suppose $a_1, a_2 \colon A$ and $b_1 \colon B(a_1)$ and $b_2 \colon B(a_2)$. A path

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

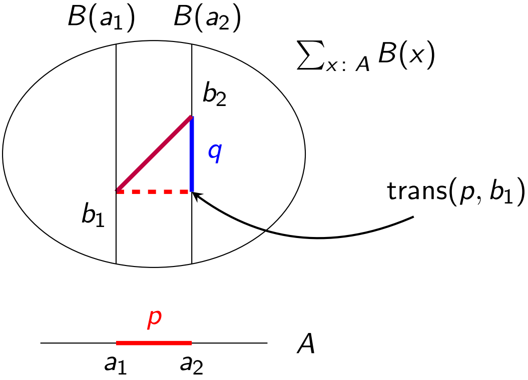
in $\sum_{x \colon A} B(x)$ should consist of

- A path $p \colon (a_1 = a_2)$ 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 $q \colon (\mathsf{trans}(p, b_1) = b_2)$.

Paths in dependent sums



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 trans(p, b_1).
- Any other lift of p is determined by a path in the fiber $B(a_2)$.

Outline

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:

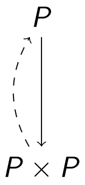
1. *Each space* $\mathrm{Hom}(X, P)$ *is empty or contractible.*
2. *Any two morphisms* $X \rightrightarrows P$ *are homotopic.*
3. *The diagonal* $P \to P \times P$ *has a section.*
4. *The diagonal* $P \to P \times P$ *is an equivalence.*

h-Propositions

Definition

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

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



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".

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.

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.
    - $\mathrm{supp}(x = y)$ 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).

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.
  - $\mathrm{supp}(x = y)$ 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"?

$$\mathrm{isProp}(A) :\equiv \mathrm{supp}\left(\prod_{x:A}\prod_{y:A}(x=y)\right) \qquad ?$$

Internalizing h-props

How can we say in type theory "A is an h-prop"?

$$\mathrm{isProp}(A) :\equiv \prod_{x:A} \prod_{y:A} (x = y) \qquad !$$

This is already an h-prop!

Theorem

For any A, we can construct a term in

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

Some subtleties

- We can loosely read $\prod_{x\colon A} \prod_{y\colon A} (x = y)$ as

  "for all $x, y\colon A$, we have a path $(x = y)$"

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

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

  This asserts that "if A is nonempty, then it is connected."

Some subtleties

- We can loosely read $\prod_{x:\,A} \prod_{y:\,A} (x = y)$ as

  "for all $x, y \colon A$, we have a path $(x = y)$"

- But "for all $x, y \colon A$, there exists a path $(x = y)$" should be read to mean

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

  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).

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!

Outline

Homotopy equivalences

Definition

A function $f\colon A \to B$ is a homotopy equivalence if there exists $g\colon B \to A$ and homotopies $g \circ f \sim \mathrm{id}_A$ and $f \circ g \sim \mathrm{id}_B$.

$$\mathsf{isHtpyEquiv}(f) := \mathsf{supp}\left(\sum_{g\colon B\to A}\Big((g \circ f = \mathrm{id}_A) \times (f \circ g = \mathrm{id}_B)\Big)\right)$$

Homotopy equivalences

Definition

A function $f\colon A \to B$ is a homotopy equivalence if there exists $g\colon B \to A$ and homotopies $g \circ f \sim \mathrm{id}_A$ and $f \circ g \sim \mathrm{id}_B$.

$$\mathrm{isHtpyEquiv}(f) :\equiv \mathrm{supp}\left(\sum_{g\colon B \to A} \Big((g \circ f = \mathrm{id}_A) \times (f \circ g = \mathrm{id}_B)\Big)\right)$$

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

Back to bijections

A function $f \colon A \to B$ between sets is a bijection if

1. There exists $g \colon B \to A$ such that $g \circ f = \mathrm{id}_A$ and $f \circ g = \mathrm{id}_B$.

Back to bijections

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

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

Back to bijections

A function $f\colon A \to B$ between sets is a bijection if

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

Voevodsky equivalences

Definitions

The homotopy fiber of $f\colon A \to B$ at $b\colon B$ is

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

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

$$\mathrm{isContr}(X) :\equiv \mathrm{isProp}(X) \times X$$

Definition (Voevodsky)

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

$$\mathrm{isEquiv}(f) :\equiv \prod_{b\colon B} \mathrm{isContr}(\mathrm{hfiber}(f, b))$$

This is an h-prop.

H-isomorphisms

Definition (Joyal)

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

$$\mathrm{isHIso}(f) := \left(\sum_{g\colon B \to A} (g \circ f = \mathrm{id}_A) \right) \times \left(\sum_{h\colon B \to A} (f \circ h = \mathrm{id}_B) \right)$$

This is also an h-prop.

Adjoint equivalences

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

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

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

Adjoint equivalences

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

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

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

(2a) For all $b\colon B$, we have $\ldots v(g(b) \ldots \mathrm{map}(g, u(b)) \ldots$

(2b) For all $a\colon A$, we have $\ldots u(f(a) \ldots \mathrm{map}(f, v(a)) \ldots$

$\vdots$

Adjoint equivalences

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

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

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

(2a) For all $b\colon B$, we have $\ldots v(g(b) \ldots \mathrm{map}(g, u(b)) \ldots$

(2b) For all $a\colon A$, we have $\ldots u(f(a) \ldots \mathrm{map}(f, v(a)) \ldots$

$\vdots$

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

Adjoint equivalences

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

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

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

(2a) For all $b\colon B$, we have $\ldots v(g(b) \ldots \mathrm{map}(g, u(b)) \ldots$

(2b) For all $a\colon A$, we have $\ldots u(f(a) \ldots \mathrm{map}(f, v(a)) \ldots$

$\vdots$

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

Definition

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

$$\mathrm{isAdjEquiv}(f) :\equiv \sum_{g\colon B\to A} \sum_{r\colon \ldots} \sum_{s\colon \ldots} \Big(r(g(b)) = \mathrm{map}(g, s(b)) \Big)$$

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

$$\mathsf{isEquiv}(f) \simeq \mathsf{isHIso}(f) \simeq \mathsf{isAdjEquiv}(f)$$

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

$$\mathsf{isEquiv}(f) \simeq \mathsf{isHIso}(f) \simeq \mathsf{isAdjEquiv}(f)$$

Definition

The type of equivalences between $A, B :$ Type is

$$\mathsf{Equiv}(A, B) :\equiv \sum_{f : A \to B} \mathsf{isEquiv}(f).$$

The short five lemma

$$
\begin{array}{ccccc}
\mathrm{hfiber}(f) & \longrightarrow & A & \xrightarrow{f} & B \\
\downarrow{\scriptstyle r} & & \downarrow{\scriptstyle s} & & \downarrow{\scriptstyle t} \\
\mathrm{hfiber}(g) & \longrightarrow & C & \xrightarrow[g]{} & D
\end{array}
$$

Theorem

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

The short five lemma

$$
\begin{array}{ccccc}
\mathrm{hfiber}(f) & \longrightarrow & A & \xrightarrow{f} & B \\
\big\downarrow r & & \big\downarrow s & & \big\downarrow t \\
\mathrm{hfiber}(g) & \longrightarrow & C & \xrightarrow[g]{} & D
\end{array}
$$

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

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

The 3×3 lemma

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

Theorem

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

(Also a theorem in type theory.)

Homotopical uniqueness

Theorem

For any types A, B, C, the map

$$\lambda f.\big(\lambda a.f(\mathsf{inl}(a)),\, \lambda b.f(\mathsf{inr}(b))\big) \;:\; C^{A+B} \to C^A \times C^B$$

is an equivalence (using the dependent eliminator).

Homotopical uniqueness

Theorem

For any types A, B, C, the map

$$\lambda f.\big(\lambda a.f(\mathsf{inl}(a)),\, \lambda b.f(\mathsf{inr}(b))\big) \;:\; C^{A+B} \to C^A \times C^B$$

is an equivalence (using the dependent eliminator).

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

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

Homotopical uniqueness

Theorem

For any types A, B, C, the map

$$\lambda f.\big(\lambda a.f(\mathrm{inl}(a)),\, \lambda b.f(\mathrm{inr}(b))\big) \;:\; C^{A+B} \to C^A \times C^B$$

is an equivalence (using the dependent eliminator).

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

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

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

Homotopical function extensionality

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

$$\mathsf{happly} \ : \ \left((f = g) \to \prod_{a \colon A} (f(a) = g(a))\right)$$

defined by identity elimination:

$$\mathsf{happly}(\mathsf{refl}_f) :\equiv \lambda a.\mathsf{refl}_{f(a)}$$

Homotopical function extensionality

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

$$\mathsf{happly} \ : \ \left((f = g) \to \prod_{a \colon A} (f(a) = g(a)) \right)$$

defined by identity elimination:

$$\mathsf{happly}(\mathsf{refl}_f) :\equiv \lambda a.\mathsf{refl}_{f(a)}$$

Theorem (Voevodsky)

happly *is an equivalence (using the naive* funext*).*

Also works for dependent functions.

Outline

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".

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”.

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

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 : Type, we have

$$\mathsf{pathToEquiv}_{A,B} \;:\; \Big((A = B) \to \mathsf{Equiv}(A, B)\Big)$$

defined by identity elimination.
Note: $(A = B)$ is a path-type of "Type".

The univalence axiom

For $A, B\colon \mathsf{Type}$, we have

$$\mathsf{pathToEquiv}_{A,B} \ : \ \Big((A = B) \to \mathsf{Equiv}(A, B)\Big)$$

defined by identity elimination.
Note: $(A = B)$ is a path-type of "Type".

The Univalence Axiom (Voevodsky)

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

$$\prod_{A\colon \mathsf{Type}} \ \prod_{B\colon \mathsf{Type}} \mathsf{isEquiv}(\mathsf{pathToEquiv}_{A,B})$$

In particular, every equivalence yields a path between types.

The meaning of univalence

The meaning of univalence

Given an equivalence $f\colon A \xrightarrow{\sim} B$, 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.

The meaning of univalence

The meaning of univalence

Given an equivalence $f \colon A \xrightarrow{\sim} B$, 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. □

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. □

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).

The uses of univalence

1. The homotopy theory is nontrivial (Type is not an h-set).
2. (Voevodsky) Univalence implies funext.

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

$$\sum_{A\colon \mathsf{Type}} \mathsf{supp}(A = F)$$

   is the classifying space for bundles with fiber F.

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

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

   is the classifying space for bundles with fiber F.
4. Computing homotopy groups! (on Friday)

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

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

   is the classifying space for bundles with fiber F.
4. Computing homotopy groups! (on Friday)
5. Many more ...

Internalizing h-props

Theorem

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

Proof.

- Suppose $H, K : \mathrm{isProp}(A)$; we want $(H = K)$.

Internalizing h-props

Theorem

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

Proof.

- Suppose $H, K : \mathsf{isProp}(A)$; we want $(H = K)$.
- By funext, suffices to show $H(x, y) = K(x, y)$ for all $x, y : A$.

Internalizing h-props

Theorem

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

Proof.

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

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

Internalizing h-props

Theorem

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

Proof.

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

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

- Hence $H(x,y) * K(x,x) = K(x,y)$ (a fact).

Internalizing h-props

Theorem

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

Proof.

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

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

- Hence $H(x, y) * K(x, x) = K(x, y)$ (a fact).
- It suffices to prove $K(x, x) = \mathsf{refl}_x$.

Internalizing h-props

Theorem

For any A, isProp(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 $K(x, x) * K(x, x) = K(x, x)$.

Internalizing h-props

Theorem

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

Proof.

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

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

- Hence $H(x, y) * K(x, x) = K(x, y)$ (a fact).
- It suffices to prove $K(x, x) = \mathrm{refl}_x$.
  - 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 $K(x, x)^{-1}$). □