CASS 2020
Coq Andes Summer School

Homotopy Type Theory

Nicolas Tabareau

HoTT is about equality

Do we need a primitive notion of equality ?

HoTT is about equality

For booleans, as we have seen, equality can be defined as a fixpoint.

```
Definition eq_bool (b b' : 𝔹) : ℙ :=
  match b , b' with
    true , true ⇒ ⊤
  | false, false ⇒ ⊤
  | _ , _ ⇒ ⊥
  end.
```

HoTT is about equality

With this definition, it is possible to show
Leibniz principle of indiscernibility of premisses.

```
Definition transport_bool :
  ∀ (P : 𝔹 → Type) (x y : 𝔹), eq_bool x y → P x → P y.
Proof.
  move ⇒ P x. elim: x ⇒ [|] ⇒ y; by case: y.
Defined.
```

HoTT is about equality

For natural numbers, in the same way, equality can be defined as a fixpoint.

```
Fixpoint eq_nat (n m : ℕ) : ℙ :=
  match n , m with
    0 , 0 ⇒ ⊤
  | S n, S m ⇒ eq_nat n m
  | _ , _ ⇒ ⊥
  end.
```

HoTT is about equality

Again, with this definition, it is possible to show Leibniz principle of indiscernibility of premisses.

```
Definition transport_nat :
  ∀ (P : ℕ → Type) (x y : ℕ), eq_nat x y → P x → P y.
```

HoTT is about equality

For functions, assuming that the codomain has a notion of equality, we can define equality between functions as point wise equality.

```
Definition eq_fun A B (eqB : ∀ (b b' : B), ℙ) :
  (A → B) → (A → B) → ℙ := λ f g ⇒ ∀ x:A, eqB (f x) (g x).
```

HoTT is about equality

HoTT is about equality

Problem:

We can not prove Leibniz principle on functions.

HoTT is about equality

We need some stronger principle in the theory, which assumes the existence of an equality type for every type, with elimination principle.

```
Definition transport {A : Type} (P : A → Type) {x y : A}
           (p : x = y) (u : P x) : P y
```

(This corresponds to the rewrite tactic)

HoTT is about equality

Actually, such an equality can be defined using the following inductive type in Coq.

```
Inductive I (A : Type) (x : A) : A → Type :=
  refl : I A x x.

Notation "x = y" := (I _ x y): type_scope.
```

HoTT is about equality

With the corresponding elimination principle (more general than transport):

```
Definition subst (A:Type) (t:A) (P : ∀ y:A, t = y → Type)
           (u : A) (p : t = u) (v:P t (refl t)) : P u p :=
  match p with
  | refl _ ⇒ v : P t (refl t)
  end.
```

HoTT is about equality

Coming back to eq_nat, as we have seen, it is possible to show that it coincides with equality:

```
Lemma eq_nat_eq (n m : ℕ) : eq_nat n m ↔ n = m.
```

Only works for inductive types.

HoTT is about equality

HoTT is about equality

Problem:

This equality is missing many interesting principles

Functional Extensionality

$$\forall A\ B\ (f\ \ g : \forall a : A, B\ \ a), (\forall a, f\ \ a = g\ \ a) \to f = g$$

Uniqueness of Identity Proof

$$\forall A\ (x\ y : A)\ (e\ e' : x = y), e = e'$$

Univalence

$$\forall (A\ B : \texttt{Type}), A \simeq B \to A = B$$

Issue

Univalence + UIP = 

Issue

All those principles cannot be valid altogether !

Univalence + UIP = 

Issue

UIP implies axiom K, which says that the only proof of x = x is refl.

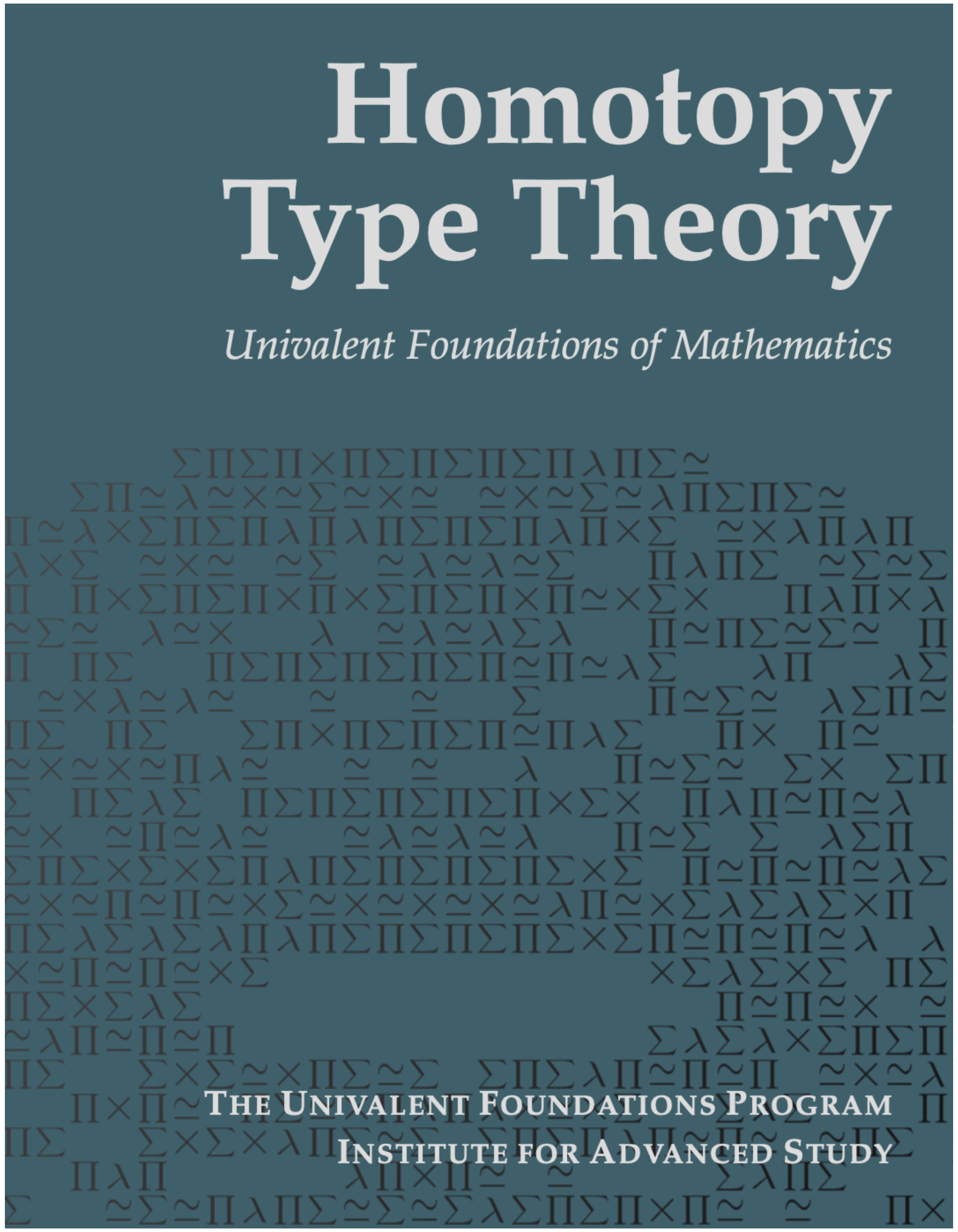
However, there are two automorphisms on booleans (id and neg).

By univalence, they correspond to equalities,
By UIP neg is equal to the identity

=> true = false 

HoTT to the rescue

We need a more conceptual point of view on equality !



Type and Set Theory

Types	Sets
A	set
$a : A$	element
$B(x)$	family of sets
$b(x) : B(x)$	family of elements
$\mathbf{0}, \mathbf{1}$	$\emptyset, \{\emptyset\}$
$A + B$	disjoint union
$A \times B$	set of pairs
$A \to B$	set of functions
$\sum_{(x:A)} B(x)$	disjoint sum
$\prod_{(x:A)} B(x)$	product
Id_A	$\{\,(x, x) \mid x \in A\,\}$

Type and Set Theory

Types	Sets
A	set
$a : A$	element
$B(x)$	family of sets
$b(x) : B(x)$	family of elements
$\mathbf{0}, \mathbf{1}$	$\emptyset, \{\emptyset\}$
$A + B$	disjoint union
$A \times B$	set of pairs
$A \to B$	set of functions
$\sum_{(x:A)} B(x)$	disjoint sum
$\prod_{(x:A)} B(x)$	product
Id_A	$\{ (x, x) \mid x \in A \}$

Type and Homotopy Theory

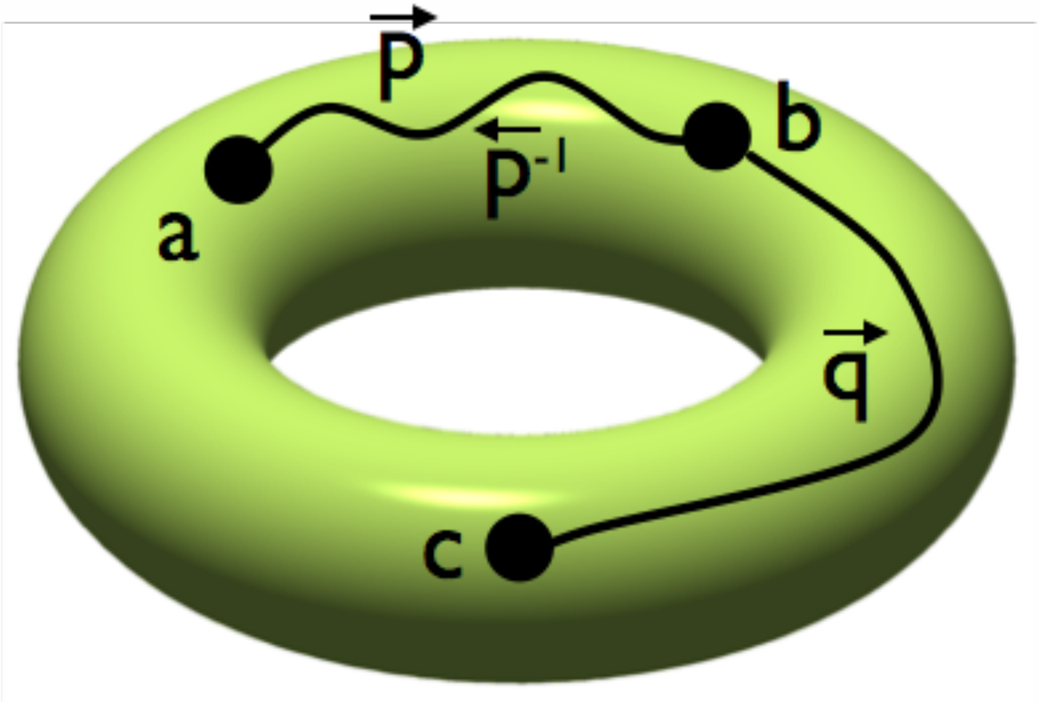
Types	Homotopy
A	space
$a : A$	point
$B(x)$	fibration
$b(x) : B(x)$	section
$\mathbf{0}, \mathbf{1}$	$\emptyset, *$
$A + B$	coproduct
$A \times B$	product space
$A \to B$	function space
$\sum_{(x:A)} B(x)$	total space
$\prod_{(x:A)} B(x)$	space of sections
Id_A	path space A^I

Type and Homotopy Theory

Types	Homotopy
A	space
$a : A$	point
$B(x)$	fibration
$b(x) : B(x)$	section
$\mathbf{0}, \mathbf{1}$	$\emptyset, *$
$A + B$	coproduct
$A \times B$	product space
$A \to B$	function space
$\sum_{(x:A)} B(x)$	total space
$\prod_{(x:A)} B(x)$	space of sections
$\boxed{\mathsf{Id}_A}$	$\boxed{\text{path space } A^I}$

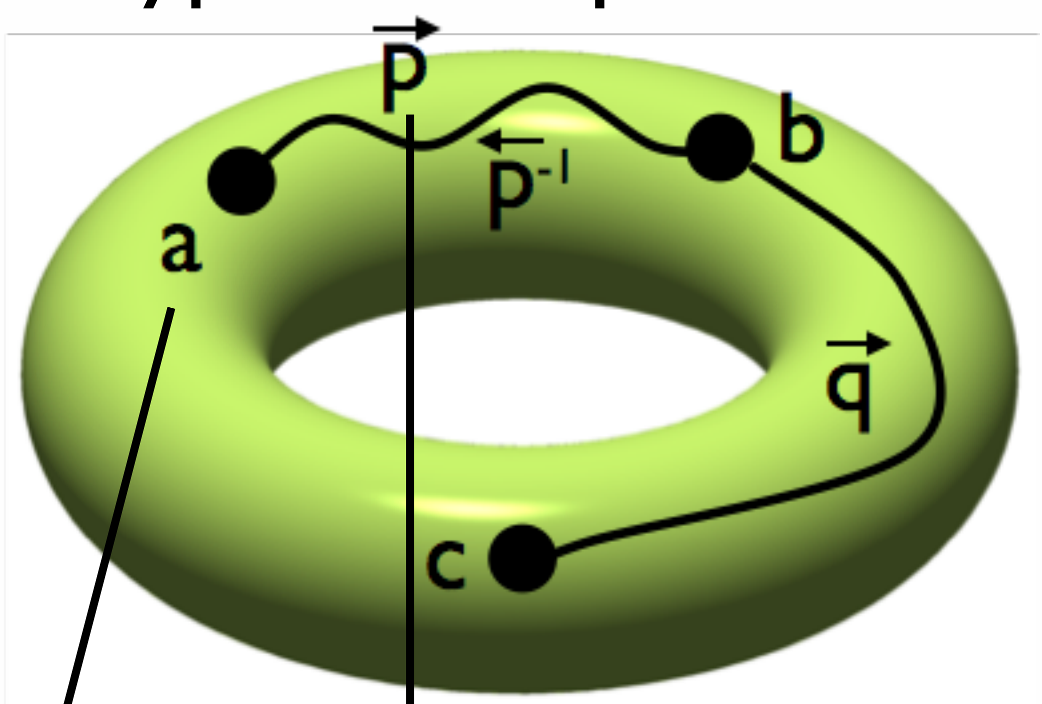
∞-groupoids and equality

type T is a space



∞-groupoids and equality

type T is a space

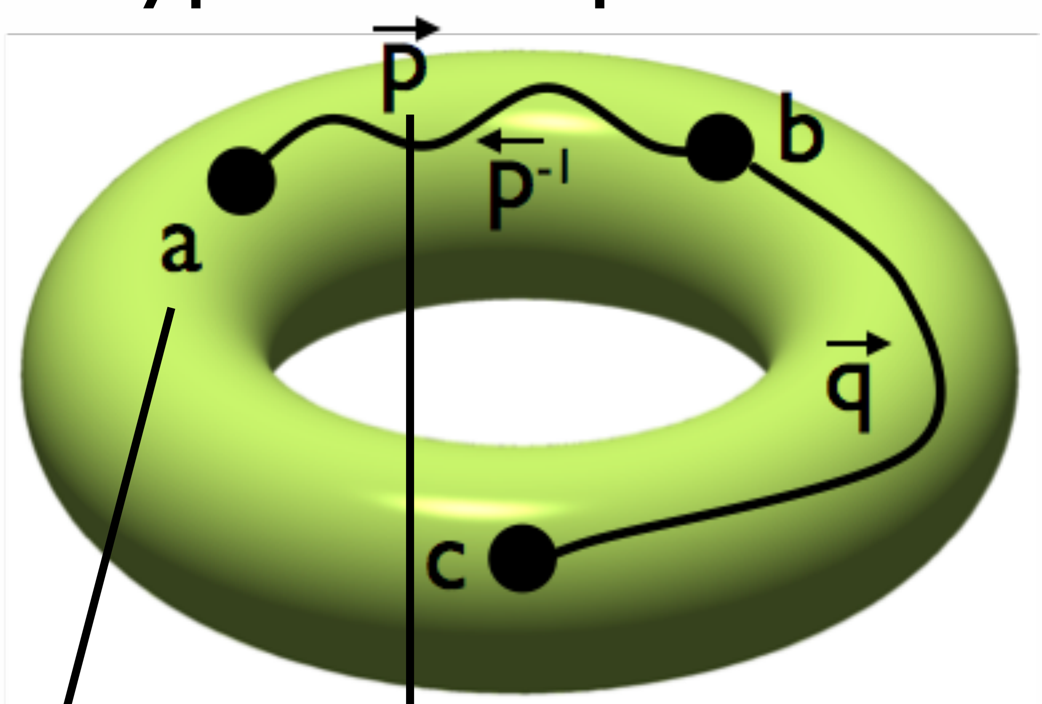


programs
a:T are
points

proofs of equality
p : a = b
are paths

∞-groupoids and equality

type T is a space



programs
a:T are
points

proofs of equality
p : a = b
are paths

Path operations:

id	$: a =_T a$
p^{-1}	$: b =_T a$
$q \circ p$	$: a =_T c$

Homotopies:

left-id	$: id \circ p =_{a=b} p$
right-id	$: p \circ id =_{a=b} p$
assoc	$: r \circ (q \circ p) =_{a=d} (r \circ q) \circ p$

∞-groupoids and equality

All those laws can be proven using
the dependent elimination of equality

∞-groupoids and equality

It has been shown formally that every type forms an ∞-groupoid.

Conversely, it is believed that HoTT is the right language to described ∞-groupoids.

=> Synthetic Homotopy Theory

A Hierarchy of Types

A Hierarchy of Types

One of the main contribution of V.V. in type theory is the notion of levels of homotopy of types.

A Hierarchy of Types

Types are classified by the complexity of their equality/identity type.

Simplest (singleton) types are called contractible:

$$\mathsf{isContr}(A) :\equiv \sum_{(a:A)} \prod_{(x:A)} (a = x).$$

A Hierarchy of Types

Types are classified by the complexity of their equality/identity type.

Proposition have a contractible equality:

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

A Hierarchy of Types

Types are classified by the complexity of their equality/identity type.

Then, n-Types are defined inductively:

Define the predicate is-n-type : $\mathcal{U} \to \mathcal{U}$ for $n \geq -2$ by recursion as follows:

$$\text{is-}n\text{-type}(X) :\equiv \begin{cases} \text{isContr}(X) & \text{if } n = -2, \\ \prod_{(x,y:X)} \text{is-}n'\text{-type}(x =_X y) & \text{if } n = n' + 1. \end{cases}$$

A Hierarchy of Types

This defines the following hierarchy:

Level of Type	Homotopy Type Theory
(-2)-Type	unit / contactible type
(-1)-Type	h-propositions
0-Type	h-sets
1-Type	h-groupoids
…	…
Type	∞-groupoids

A Hierarchy of Types

So with this point of view, UIP is not a property of the system but a property of a type: IsHset.

A Hierarchy of Types

Equivalences

```
Class IsEquiv {A : Type} {B : Type} (f : A → B) := BuildIsEquiv {
  e_inv :> B → A ;
  e_sect : ∀ x, e_inv (f x) = x;
  e_retr : ∀ y, f (e_inv y) = y;
  e_adj : ∀ x : A, e_retr (f x) = ap f (e_sect x);
}.
```

Equivalences

Univalence

In the Coq/HoTT library, univalence is stated as an axiom, but there has been a huge effort to make it a property of type theory.

Cubical Type Theory (and Cubical Agda).

Equivalences

An important fact is that being an equivalence is a mere proposition

Equivalences

An important fact is that being an equivalence is a mere proposition

Isomorphisms and Equivalences

Every isomorphism can be turned
into an equivalence by changing slightly the section

This is a well known fact in homotopy theory.

Isomorphisms and Equivalences

Every isomorphism can be turned
into an equivalence by changing slightly the section

This is a well known fact in homotopy theory.

Equivalences

The notion of equivalences behaves well with respect to type constructors and homotopy levels

Equivalences

The notion of equivalences behaves well with respect to type constructors and homotopy levels

Hedberg Theorem

Coming back on UIP and HSet, we can actually prove that every type with a decidable equality is an HSet.

We will see to quite different proofs:

- one using HoTT reasoning
- one using hardcore pattern matching.

Hedberg Theorem

Coming back on UIP and HSet, we can actually prove that every type with a decidable equality is an HSet.

We will see to quite different proofs:

- one using HoTT reasoning
- one using hardcore pattern matching.

Higher Inductive Types

HoTT is not only about equivalences and univalence.

An other very important ingredient to do synthetic homotopy theory is the introduction of HITs.

Higher Inductive Types

We will see them in Coq/HoTT, where they can be axiomatized.

But the system in which they live more naturally is cubical type theory.

Higher Inductive Types

Examples:

- Interval
- Propositional Truncation
- Circle (more generally n-Sphere)
- Cylinder

Higher Inductive Types

Examples:

- Interval
- Propositional Truncation
- Circle (more generally n-Sphere)
- Cylinder

Conclusion

Homotopy Type Theory is an important for both

- Understanding equality in type theory
- Synthetic Homotopy Theory