

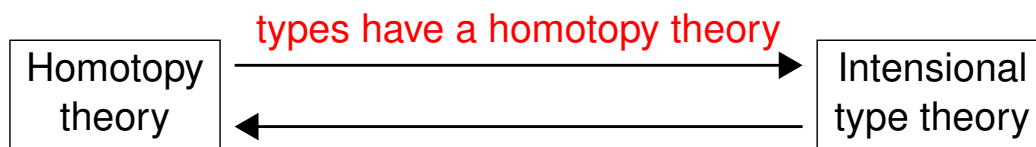
Introduction to type theory and homotopy theory

Michael Shulman

January 24, 2012

1 / 47

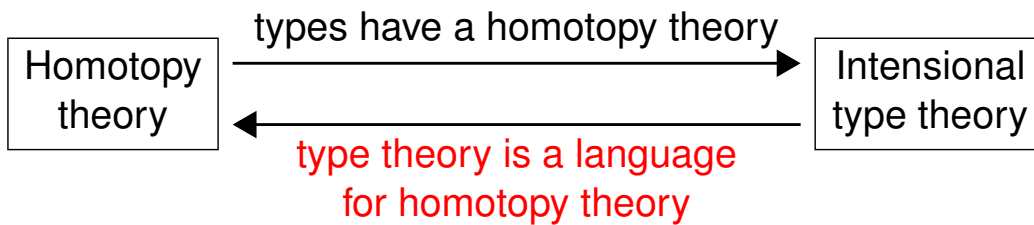
Homotopy type theory



- New perspectives on extensional vs. intensional
- Univalence: the correct identity types for a universe
- Homotopy levels: insight into proof-irrelevance
- Higher inductive types: quotients and free structures

2 / 47

Homotopy type theory



- Type theory is a formal system, like ZFC, but...
 - More computer-friendly
 - Naturally constructive
 - Can formalize homotopy theory more directly
- The same proof can apply to many homotopy theories (equivariant, parametrized, sheaves, ...)

2 / 47

Outline

- ① A bird's-eye view of type theory
- ② A bird's-eye view of homotopy theory
- ③ Path spaces and identity types
- ④ Homotopy type theory
- ⑤ Looking ahead

3 / 47

Typing judgments

Type theory consists of rules for manipulating **judgments**.
The most important judgment is a **typing judgment**:

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

The turnstile \vdash binds most loosely, followed by commas.
This should be read as:

In the context of variables x_1 of type A_1 , x_2 of type A_2 , ...,
and x_n of type A_n , the expression b has type B .

Examples

$$\begin{aligned} & \vdash 0 : \mathbb{N} \\ x : \mathbb{N}, y : \mathbb{N} & \vdash x + y : \mathbb{N} \\ f : \mathbb{R} \rightarrow \mathbb{R}, x : \mathbb{R} & \vdash f(x) : \mathbb{R} \\ f : C^\infty(\mathbb{R}, \mathbb{R}), n : \mathbb{N} & \vdash f^{(n)} : C^\infty(\mathbb{R}, \mathbb{R}) \end{aligned}$$

5/47

Dependent types

We consider types A_i, B as also expressions, of type “Type”.

Examples

$$\begin{aligned} & \vdash \mathbb{N} : \text{Type} \\ A : \text{Type}, x : A & \vdash x : A \\ A : \text{Type}, B : A \rightarrow \text{Type}, x : A & \vdash B(x) : \text{Type} \\ n : \mathbb{N} & \vdash \{k : \mathbb{N} \mid k < n\} : \text{Type} \\ f : \mathbb{R} \rightarrow \mathbb{R} & \vdash \{x : \mathbb{R} \mid f(x) = 0\} : \text{Type} \end{aligned}$$

A judgment $x : A \vdash B(x) : \text{Type}$, or a term $B : A \rightarrow \text{Type}$, is a **dependent type** (over A).

6/47

Type constructors

$$A: \text{Type}, B: \text{Type} \vdash A \times B: \text{Type}$$
$$A: \text{Type}, B: \text{Type} \vdash A \rightarrow B: \text{Type}$$
$$A: \text{Type}, B: \text{Type} \vdash A + B: \text{Type}$$
$$A: \text{Type}, B: A \rightarrow \text{Type} \vdash \prod_{x: A} B(x): \text{Type}$$
$$A: \text{Type}, B: A \rightarrow \text{Type} \vdash \sum_{x: A} B(x): \text{Type}$$

Each comes with rules for producing and using terms, e.g.

$$A: \text{Type}, B: \text{Type}, f: A \rightarrow B, x: A \vdash f(x): B$$
$$A: \text{Type}, B: \text{Type}, x: A \vdash \text{inl}(x): A + B$$
$$A: \text{Type}, B: \text{Type}, y: B \vdash \text{inr}(y): A + B$$

7/47

Typing as programming

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

The term b can be viewed as a **program**, with inputs x_1, \dots, x_n of types A_1, \dots, A_n and output of type B , that can be **computed**.

Thus:

- Type theory helps analyze programming languages.
- Type theory can be understood and verified by a computer.

8/47

Typing as proving

Sometimes, $x_1 : P_1, x_2 : P_2, \dots, x_n : P_n \vdash q : Q$ means instead

Under hypotheses P_1, P_2, \dots, P_n ,
the conclusion Q is provable (with proof q).

Examples

$f : P \rightarrow Q, x : P \vdash f(x) : Q$ ← modus ponens
 $x : P \vdash \text{inl}(x) : P \text{ or } Q$
 $y : Q \vdash \text{inr}(y) : P \text{ or } Q$
 $x : P, y : Q \vdash \text{inl}(x) : P \text{ or } Q$ ← two different (?) proofs
 $x : P, y : Q \vdash \text{inr}(y) : P \text{ or } Q$ ← of the same thing

9/47

Propositions as types

a.k.a. proofs as terms, or the Curry-Howard correspondence

| Types | \longleftrightarrow | Propositions |
|--------------------|-----------------------|------------------------|
| $A \times B$ | \longleftrightarrow | $P \text{ and } Q$ |
| $A + B$ | \longleftrightarrow | $P \text{ or } Q$ |
| $A \rightarrow B$ | \longleftrightarrow | $P \text{ implies } Q$ |
| $\prod_{x:A} B(x)$ | \longleftrightarrow | $(\forall x : A) P(x)$ |
| $\sum_{x:A} B(x)$ | \longleftrightarrow | $(\exists x : A) P(x)$ |

10/47

Predicate logic

Also, $x_1: A_1, x_2: A_2, y_1: P_1, y_2: P_2 \vdash q: Q$ can mean

In the context of variables x_1 of type A_1 and x_2 of type A_2 ,
and under hypotheses P_1 and P_2 ,
the conclusion Q is provable (with proof q).

Examples

$n: \mathbb{N}, e: \text{even}(n) \vdash s(e): \text{odd}(n + 1)$

$x: \mathbb{R}, y: \mathbb{R} \vdash \text{comm}(x, y): (x + y = y + x)$

$x: \mathbb{R}, nz: \neg(x = 0) \vdash \text{inv}(x): (\exists y: \mathbb{R})(x \cdot y = 1)$

11/47

Uses of type theory

Since type theory includes both programming and logic, it is a natural context in which to **prove things about programs**.

$x: \text{input} \vdash \text{de}(x): (\text{decrypt}(\text{encrypt}(x)) = x)$

$x: \text{user}, y: \text{moonPhase} \vdash p(x, y): \neg \text{crashes}(\text{Windows})$

We can also **develop mathematics** in type theory, which can then be formally verified by a computer.

$g: \text{Graph}, p: \text{Planar}(g) \vdash c: \text{Coloring}(g, 4)$ ✓

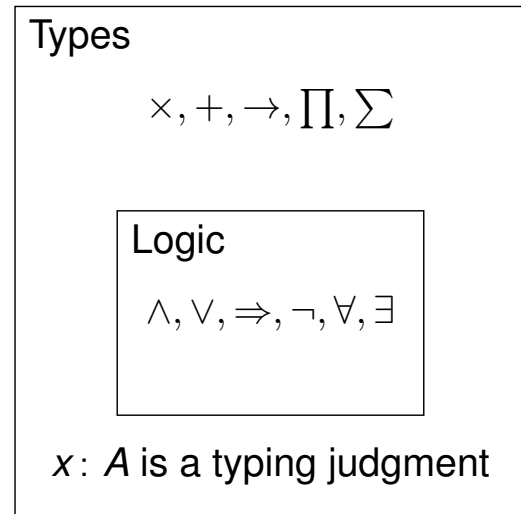
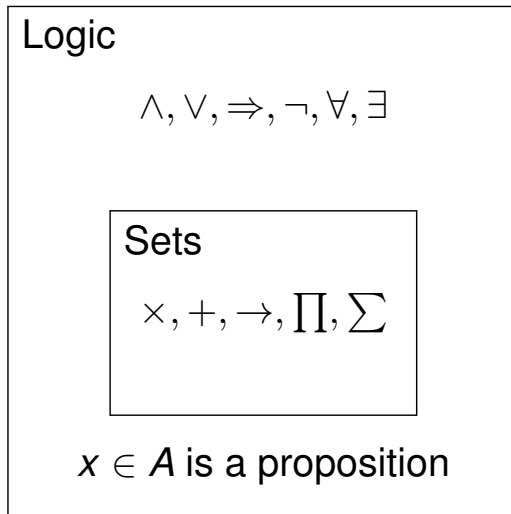
$x: \mathbb{N}, y: \mathbb{N}, z: \mathbb{N}, n: \mathbb{N}, f: (x^n + y^n = z^n) \vdash w: (n \leq 2 \text{ or } z \leq 1)$

12/47

Type-theoretic foundations

Set theory

Type theory



13/47

Type theory and category theory

Type theory can be a syntax for describing objects and morphisms in a category.

| | | |
|----------------------------------|-----------------------|---|
| $A: \text{Type}, B: \text{Type}$ | \longleftrightarrow | Objects |
| $x: A \vdash b: B$ | \longleftrightarrow | Morphism $A \rightarrow B$ |
| $x_1: A_1, x_2: A_2 \vdash b: B$ | \longleftrightarrow | Morphism $A_1 \times A_2 \rightarrow B$ |
| $A \times B: \text{Type}$ | \longleftrightarrow | Categorical product |
| $A + B: \text{Type}$ | \longleftrightarrow | Categorical coproduct |
| $A \rightarrow B: \text{Type}$ | \longleftrightarrow | Categorical exponential |
| | | \vdots |

Anything proven in type theory will hold in any such category.

14/47

Dependent types, categorically

Recall a **dependent type** is $x: A \vdash B(x): \text{Type}$ or $B: A \rightarrow \text{Type}$.
There are two ways to interpret this in a category:

- As a morphism $A \xrightarrow{B} U$, where U is a “universe” like “the set of all small sets”.
- As a morphism $p: |B| \rightarrow A$, where the type/object $B(x)$ is the “fiber” of p over x .

These are related by a pullback square:

$$\begin{array}{ccc} |B| & \longrightarrow & \tilde{U} \\ \downarrow & & \downarrow \\ A & \longrightarrow & U \end{array}$$

15/47

Type theories and category theories

| | | |
|----------------------------------|-----------------------|---------------------------|
| Simply typed λ -calculus | \longleftrightarrow | Cartesian closed category |
| Dependent type theory | \longleftrightarrow | Locally c.c. category |
| First-order predicate logic | \longleftrightarrow | Boolean/Heyting category |
| Geometric logic | \longleftrightarrow | Grothendieck topos |
| Higher-order logic | \longleftrightarrow | Elementary topos |
| ?? | \longleftrightarrow | Homotopical category |

Not too surprising

There is a type theory that goes in ??.

Surprising (to me)

That type theory was **already invented** by type theorists, long before anyone realized it had to do with homotopy!

16/47

Sets, revisited

Ignoring considerations of size, a **set** is...

- ... a collection of **elements**
- ... together with a notion of when two elements are **equal**
- ... which is **transitive, symmetric, and reflexive**.

“To define a set we prescribe... what we... must do in order to construct an element of the set, and what we must do to show that two elements are equal”

– Errett Bishop, “Foundations of Constructive Analysis”

18/47

Groupoids

A **groupoid** is...

- ... a collection of “elements” or “points”
- ... with, for all points x and y , a set $\text{hom}(x, y)$ of “isomorphisms” or “paths” from x to y
- ... which can be **composed, inverted, and have identities**

Equivalently: a category in which all morphisms are invertible.

Examples

- Elements = sets, isomorphisms = bijections
- Elements = any set X , isomorphisms = only identities
- Elements = $*$, isomorphisms = any group G

NB: Two points connected by a path are regarded as **the same**.

19/47

An ∞ -groupoid is

- ... a collection of **points**
- ... for all points x and y , a collection $\text{hom}(x, y)$ of **paths** from x to y
- ... for all paths f and g from x to y , a collection $\text{hom}(f, g)$ of **2-paths** from f to g
- ...
- with composition, inversion, identities, ...

Examples

- Any set, with only identity n -paths for $n > 0$
- Any groupoid, with only identity n -paths for $n > 1$
- Points = ∞ -groupoids, ... (more later)

20/47

The fundamental ∞ -groupoid

A **topological space** is a set together with a notion of “closeness”, “continuity”, or “deformation”.

The **fundamental ∞ -groupoid** of a topological space X has

- As points, the points of X .
- As paths, the continuous paths $[0, 1] \rightarrow X$.
- As 2-paths, continuous deformations between paths.
- ...

We denote this by $\Pi_\infty(X)$.

21/47

$\Pi_\infty(\star)$

Let \star be the one-point topological space. Then $\Pi_\infty(\star)$ has:

- One point.
- One path from that point to itself.
- One 2-path from that path to itself.
- ...

This is the same as the ∞ -groupoid arising from the set $\{\star\}$.

22/47

$\Pi_\infty(\mathbb{R})$

For \mathbb{R} the real numbers, $\Pi_\infty(\mathbb{R})$ has:

- The real numbers $x \in \mathbb{R}$ as points...
 - ...but any two of them are connected by a path, so there is essentially only one point.
- Paths between them...
 - ...but any two such paths are related by a deformation.
- ...

$\Pi_\infty(\mathbb{R})$ has **essentially** only one of all these things.

Thus it is **equivalent** to $\Pi_\infty(\star)$. The same is true for $\Pi_\infty(\mathbb{R}^n)$.

An ∞ -groupoid that is equivalent to $\Pi_\infty(\star)$ is called **contractible**.

23/47

$\Pi_\infty(S^1)$

Let $S^1 = \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 = 1\}$. Then $\Pi_\infty(S^1)$ has:

- The points $(x, y) \in S^1$ as points...
 - ... but any two of them are connected by a path, so there might as well only be one point, say $(1, 0)$.
- Lots of paths from $(1, 0)$ to itself. Two such paths α and β are inter-deformable exactly when they wind around the circle an equal number of times.
 - Thus there are essentially \mathbb{Z} -many paths from $(1, 0)$ to itself.
- For $n > 1$, every n -path is trivial.

This is (arguably) the simplest groupoid that is not a set.

24/47

$\Pi_\infty(S^2)$

Let $S^2 = \{(x, y, z) \in \mathbb{R}^3 \mid x^2 + y^2 + z^2 = 1\}$. Then $\Pi_\infty(S^2)$ has:

- Essentially only one point, say $(1, 0, 0)$.
- Essentially only the constant path from $(1, 0, 0)$ to itself.
- Essentially \mathbb{Z} -many 2-paths from the constant path to itself (how many times the deformation wraps over the surface).
- Essentially \mathbb{Z} -many 3-paths from any 2-path to itself.
- Essentially two 4-paths from any 3-path to itself.
- Essentially two 5-paths from any 4-path to itself.
- Essentially twelve 6-paths from any 5-path to itself.
- ... essentially 336 14-paths from any 13-path to itself ...
- ...

Computing these **homotopy groups of spheres** for all S^n is a big part of classical homotopy theory.

25/47

Presenting ∞ -groupoids

There are many ways to define ∞ -groupoids. All are “equivalent”, but most are technical. Instead, we can use:

Fact

Every ∞ -groupoid is equivalent to $\Pi_\infty(X)$ for some X .

Fact

For nice X and Y , we have $\Pi_\infty(X) \simeq \Pi_\infty(Y)$ if and only if X and Y are **homotopy equivalent** (next slide).

Thus it suffices to think about topological spaces up to homotopy equivalence.

26 / 47

Homotopy

A **homotopy** $f \sim g$ between continuous maps $f, g: X \rightarrow Y$ is

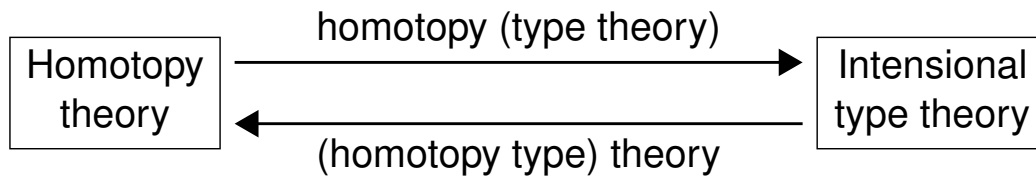
- A path from f to g in the space Y^X of continuous functions (with a suitable topology)
- **OR:** a map $H: X \times [0, 1] \rightarrow Y$ such that $H(x, 0) = f(x)$ and $H(x, 1) = g(x)$.
- **OR:** a map $\bar{H}: X \rightarrow Y^{[0,1]}$ such that $\bar{H}(x)(0) = f(x)$ and $\bar{H}(x)(1) = g(x)$.

A map $f: X \rightarrow Y$ is a **homotopy equivalence** if there exists $g: Y \rightarrow X$ and homotopies $g \circ f \sim \text{id}_X$ and $f \circ g \sim \text{id}_Y$.

27 / 47

Homotopy types

“Topological spaces up to homotopy equivalence” were studied long before “ ∞ -groupoids”. They were called **homotopy types**.



28 / 47

The ∞ -groupoid of ∞ -groupoids

For spaces X and Y , let $\text{Equiv}(X, Y) \subseteq Y^X$ be the subspace consisting of the homotopy equivalences.

Definition

The **∞ -groupoid of ∞ -groupoids** has

- As points, topological spaces
- As paths, homotopy equivalences (= points of $\text{Equiv}(X, Y)$)
- As 2-paths, paths in $\text{Equiv}(X, Y)$
- As 3-paths, 2-paths in $\text{Equiv}(X, Y)$
- ...

29 / 47

Categories with homotopies

Recall:

?? \longleftrightarrow Homotopical category

What structure on the category of topological spaces encapsulates its homotopy theory?

- The interval $[0, 1]$.
- The class of homotopy equivalences.
- For each X , there is a **cylinder** $\text{Cyl}(X) := X \times [0, 1]$.
- For each Y , there is a **path space** $\text{Paths}(Y) := Y^{[0,1]}$.

These can all work — and they work best when combined!
But $\text{Paths}(Y)$ matches the type theory best.

31 / 47

Path objects

A category has **path objects** if each object Y has a factorization of the diagonal

$$\begin{array}{ccc} Y & \longrightarrow & \text{Paths}(Y) \\ & \searrow \Delta & \downarrow \\ & & Y \times Y \end{array}$$

satisfying certain axioms.

Examples

- Topological spaces, with $\text{Paths}(Y) := Y^{[0,1]}$.
- Chain complexes, with $\text{Paths}(Y) := \underline{\text{Hom}}(\mathbb{I}, Y)$
- Any category, with $\text{Paths}(Y) := Y$ (trivial homotopy theory)

32 / 47

Equality

In logic, formulas are built from atomic formulas using connectives and quantifiers:

$$\wedge, \vee, \Rightarrow, \neg, \top, \perp, \forall, \exists$$

The most basic atomic formula is **equality** ($x = y$).

In type theory:

| | | |
|-----------------------------|-----------------------|-------------------|
| propositions | \longleftrightarrow | types |
| connectives and quantifiers | \longleftrightarrow | type constructors |
| equality | \longleftrightarrow | ?? |

33 / 47

Identity types

We may include a type constructor

$$A: \text{Type}, x: A, y: A \vdash (x = y): \text{Type}$$

This is an **equality type** or **identity type**.

There is a clean, concise, and computational way to obtain the rules for identity types, as an “inductive family” (Martin-Löf).

But this method **doesn't** imply the rule

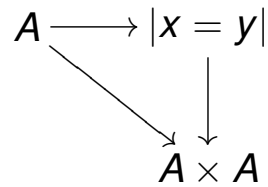
$$x: A, y: A, p: (x = y), q: (x = y) \vdash ?: (p = q)$$

If we add this additional rule, we have **extensional type theory**; otherwise it is **intensional**.

34 / 47

Identity types are path objects

The dependent type $x : A, y : A \vdash (x = y) : \text{Type}$ must be interpreted by:



Theorem (Awodey, Gambino, Garner, van den Berg, Lumsdaine, Warren)

Path objects exactly model identity types.

The trivial path objects $Y \rightarrow Y \times Y$ model extensional identity types. Others are intensional.

35 / 47

Homotopy theory in type theory

We can define internally in type theory:

- When a type is contractible.
- When a type is “set-like”, or “groupoid-like”, etc.
- When a function is a homotopy equivalence.
- The type of homotopy equivalences $\text{Equiv}(X, Y)$
- Loop spaces, homotopy groups, fibration sequences, ...

37 / 47

The univalence axiom

Recall that a **type** is a **term** of type `Type`.

$$\vdash A : \text{Type}$$

Thus, we have identity types:

$$A : \text{Type}, B : \text{Type} \vdash (A = B) : \text{Type}$$

What should this be? The standard rules don't determine it.

The univalence axiom (Voevodsky)

For types A and B , the identity type $(A = B)$ is homotopy equivalent to the type $\text{Equiv}(A, B)$ of homotopy equivalences between A and B .

38 / 47

The meaning of univalence

Univalence means that

Under the interpretation of types as ∞ -groupoids, the type `Type` corresponds to the ∞ -groupoid of ∞ -groupoids.

Some consequences of univalence:

- If there is a homotopy equivalence $f : A \rightarrow B$, then there is a path $p : (A = B)$.
- The subtype of set-like types in `Type` corresponds to the groupoid of sets.
- Dependent functions are strongly extensional (Voevodsky).

39 / 47

A simple use of univalence

In a “set-like” type, any path is deformable to the identity.

Theorem

Type is not set-like. Hence, not all types are set-like.

Proof.

- 1 Let $\text{bool} := \{\top, \perp\}$ be a two-element type.
- 2 The “flip” map $f: \text{bool} \rightarrow \text{bool}$, defined by $f(\perp) := \top$ and $f(\top) := \perp$, is a homotopy equivalence that is not homotopic to the identity.
- 3 Hence, by univalence, it gives a path $p_f: (\text{bool} = \text{bool})$ in Type that is not deformable to the identity.

□

40 / 47

Modeling univalence

Theorem (Voevodsky)

There is a model of type theory in ∞ -groupoids, for which the univalence axiom holds.

- This is nontrivial (only) because of strictness and coherence issues.
- Many other categories (called “ $(\infty, 1)$ -toposes”) contain objects like “the ∞ -groupoid of ∞ -groupoids”, but we don’t yet know how to overcome the technical issues there.

41 / 47

Inductive types

A large class of type constructors are **inductive types**.

Example

\mathbb{N} is inductively generated by $0: \mathbb{N}$ and $s: \mathbb{N} \rightarrow \mathbb{N}$.

- \Leftrightarrow Every $n: \mathbb{N}$ is generated in a unique way from 0 and s .
- \Leftrightarrow We can define functions by recursion on \mathbb{N} .
- \Leftrightarrow We can prove theorems by induction on \mathbb{N} .

Example

The disjoint union $A + B$ is inductively generated by $\text{inl}: A \rightarrow A + B$ and $\text{inr}: B \rightarrow A + B$.

- \Leftrightarrow Every $x: A + B$ arises from exactly one of inl and inr .
- \Leftrightarrow We can define functions by cases on $A + B$.
- \Leftrightarrow We can prove theorems by cases on $A + B$.

42/47

Higher inductive types

We can extend this to homotopy types (Lumsdaine–S.).

Example

S^1 is inductively generated by $b: S^1$ and a path $\ell: (b = b)$.

- \Leftrightarrow Every point, path, or higher path of S^1 is “generated uniquely” from b and ℓ .
- \Leftrightarrow We can define functions $S^1 \rightarrow A$ “recursively” or “by cases”, by giving a point $f_b: A$ and a path $f_\ell: (f_b = f_b)$.
- \Leftrightarrow We can prove theorems by “induction” on S^1 .

Example

S^2 is inductively generated by $c: S^2$ and a 2-path $\sigma: (\text{id}_c = \text{id}_c)$.

43/47

Recall S^1 is inductively generated by $b: S^1$ and $\ell: (b = b)$.

Theorem (S.)

Assuming the univalence axiom, the identity type $(b = b)$ is equivalent to \mathbb{Z} .

- Thus, paths from b to b are classified by integers, and there are no nontrivial higher paths.
- In homotopy-theoretic language, this implies $\pi_1(S^1) \cong \mathbb{Z}$.
- The proof is completely written inside of type theory, and has been fully verified by the computer proof assistant Coq.

44 / 47

Looking ahead: this quarter

- 1 Some type theory, precisely
- 2 Programming type theory in the proof assistant Coq
- 3 Some homotopy theory, precisely
- 4 Programming homotopy type theory in Coq
- 5 Categorical models of homotopy type theory
- 6 ...

46 / 47

Suggested Homework

- 1 Install Coq: <http://coq.inria.fr>
- 2 Learn a bit of functional programming.
- 3 Learn a bit of category theory.