Homotopy type theory

Richard Garner

Macquarie University

SAPLING 2013

Martin–Löf type theory

- Simply-typed λ-calculus;

Martin–Löf type theory

- Simply-typed λ-calculus;
- Plus inductive definitions (= abstract data types);

Martin–Löf type theory

- Simply-typed λ-calculus;
- Plus inductive definitions (= abstract data types);
- Plus "`Type : Type`";

Martin–Löf type theory

- Simply-typed λ-calculus;
- Plus inductive definitions (= abstract data types);
- Plus "`Type : Type`";
- Plus dependent functions.

Martin–Löf type theory

- Simply-typed λ-calculus;
- Plus inductive definitions (= abstract data types);
- Plus "`Type : Type`";
- Plus dependent functions.

All programs must terminate!

Inductive definitions

```
Inductive nat : Type :=
  | O : nat
  | s : nat -> nat.
```

Inductive definitions

```
Inductive nat : Type :=
  | O : nat
  | s : nat -> nat.
```

```
Inductive list (A : Type) : Type :=
  | nil : list A
  | cons : A -> list A -> list A.
```

Inductive definitions

```
Inductive nat : Type :=
  | O : nat
  | s : nat -> nat.
```

```
Inductive list (A : Type) : Type :=
  | nil : list A
  | cons : A -> list A -> list A.
```

Typings:

```
list : Type -> Type
nil : (A : Type) -> list A
cons : (A : Type) -> A -> list A -> list A
(++) : (A : Type) -> list A -> list A -> list A
```

Dependent inductive definitions

```
Inductive slist (A : Type) : nat -> Type :=
  | nil : slist A 0
  | cons : (n : nat) -> A -> slist A n -> slist A (s n).
```

Dependent inductive definitions

```
Inductive slist (A : Type) : nat -> Type :=
  | nil : slist A 0
  | cons : (n : nat) -> A -> slist A n -> slist A (s n).
```

Typings:

```
slist : Type -> nat -> Type
nil : (A : Type) -> slist A 0
cons : (A : Type) -> (n : nat) -> A -> slist A n -> slist A (s n)
hd : (A : Type) -> (n : nat) -> slist A (s n) -> A
tl : (A : Type) -> (n : nat) -> slist A (s n) -> slist A n
zip : (A B : Type) -> (n : nat) -> slist A n -> slist B n ->
    slist (A*B) n
```

Program specification

```
Inductive leq : nat -> nat -> Type :=
  | refl : (n : nat) -> leq n n
  | succ : (n m : nat) -> leq n m -> leq n (s m).

lt (n m : nat) : Type := leq (s n) m.
```

Program specification

```
Inductive leq : nat -> nat -> Type :=
  | refl : (n : nat) -> leq n n
  | succ : (n m : nat) -> leq n m -> leq n (s m).

lt (n m : nat) : Type := leq (s n) m.
```

A function of type `(n : nat) -> (m : nat) * lt n m` is an increasing function on natural numbers.

Program specification

```
Inductive leq : nat -> nat -> Type :=
  | refl : (n : nat) -> leq n n
  | succ : (n m : nat) -> leq n m -> leq n (s m).

lt (n m : nat) : Type := leq (s n) m.
```

A function of type `(n : nat) -> (m : nat) * lt n m` is an increasing function on natural numbers.

```
prime (p : nat) : Type :=
   (n : nat) -> (lt 1 n) -> (lt n p) -> (lt 0 (p % n)).
```

Program specification

```
Inductive leq : nat -> nat -> Type :=
  | refl : (n : nat) -> leq n n
  | succ : (n m : nat) -> leq n m -> leq n (s m).

lt (n m : nat) : Type := leq (s n) m.
```

A function of type `(n : nat) -> (m : nat) * lt n m` is an increasing function on natural numbers.

```
prime (p : nat) : Type :=
   (n : nat) -> (lt 1 n) -> (lt n p) -> (lt 0 (p % n)).
```

A function of type `(n : nat) -> (m : nat) * (lt n m) * (prime m)` is a proof that there are infinitely many primes!

Equality

```
Inductive leq : nat -> nat -> Type :=
  | refl : (n : nat) -> leq n n
  | succ : (n m : nat) -> leq n m -> leq n (s m).
```

Equality

```
Inductive eq : nat -> nat -> Type :=
  | refl : (n : nat) -> leq n n.
```

Equality

```
Inductive eq (A : Type) : A -> A -> Type :=
  | refl : (a : A) -> eq A a a.
```

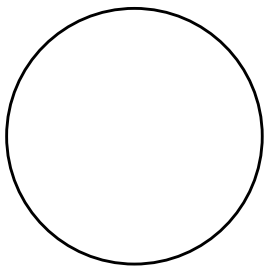
Equality

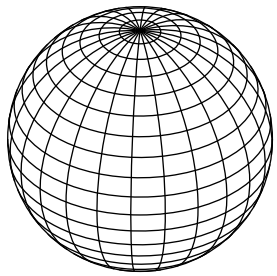
```
Inductive eq (A : Type) : A -> A -> Type :=
  | refl : (a : A) -> eq A a a.
```

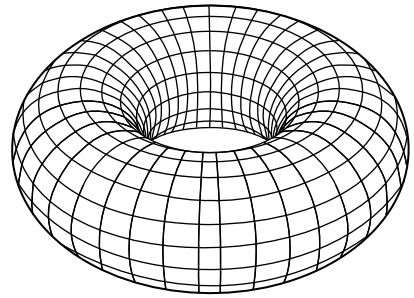
This derives the "correct" equality for most simple data types (e.g., `nat`, `bool`, `list A`, `tree A`...). But intensional at function types.

Homotopy theory

Understand geometry:



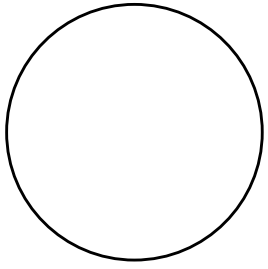


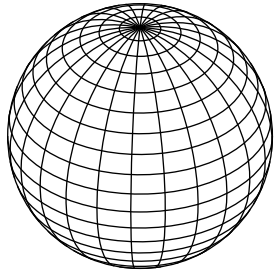


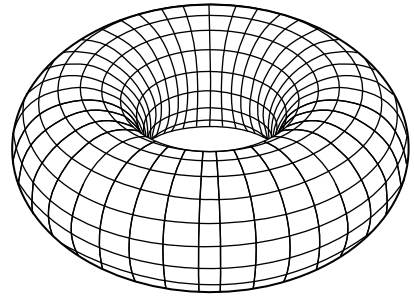
through algebraic invariants.

Homotopy type theory

Understand geometry:







through type theory!

Homotopy type theory

- Martin-Löf type theory;

Homotopy type theory

- Martin-Löf type theory;
- Plus Voevodsky's univalence axiom;

Homotopy type theory

- Martin-Löf type theory;
- Plus Voevodsky's univalence axiom;
- Plus higher inductive types;

Homotopy type theory

- Martin-Löf type theory;
- Plus Voevodsky's univalence axiom;
- Plus higher inductive types;
- Understood via the homotopical interpretation.

Univalence axiom

A function `f : A -> B` is an isomorphism if we can find:

```
g : B -> A
inv1 : (a : A) -> eq A a (g (f a))
inv2 : (b : B) -> eq B b (f (g b))
```

For example: `(+1) : Int -> Int`.

Univalence axiom

A function `f : A -> B` is an isomorphism if we can find:

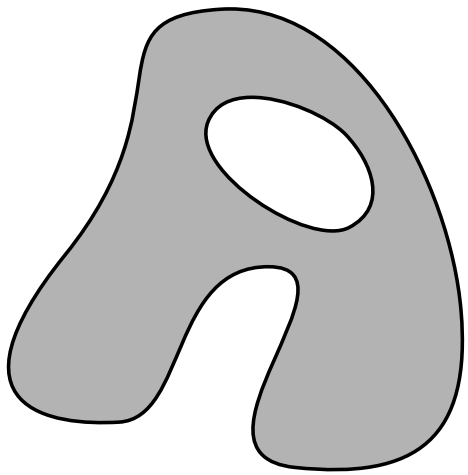
```
g : B -> A
inv1 : (a : A) -> eq A a (g (f a))
inv2 : (b : B) -> eq B b (f (g b))
```

For example: `(+1) : Int -> Int`.

Voevodsky's *univalence axiom*: `eq Type A B` is the type of isomorphisms from `A` to `B`.

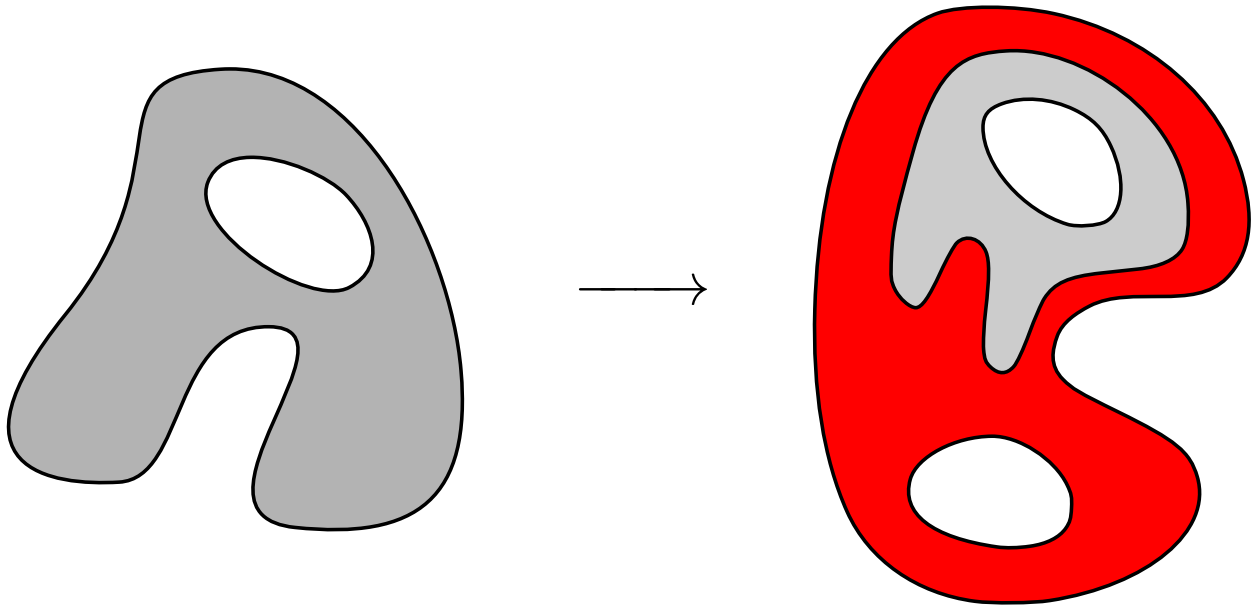
Homotopical interpretation

Types `A` as *geometric objects*



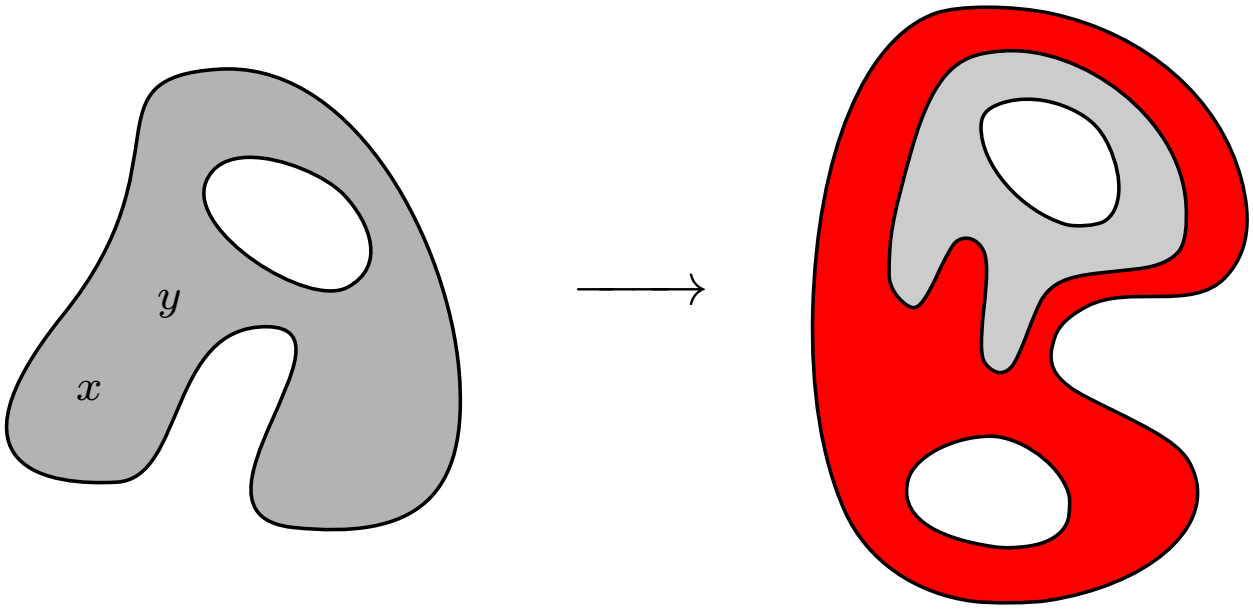
Homotopical interpretation

	Types `A`	as	*geometric objects*
Functions `f : A -> B`		as	*continuous mappings*



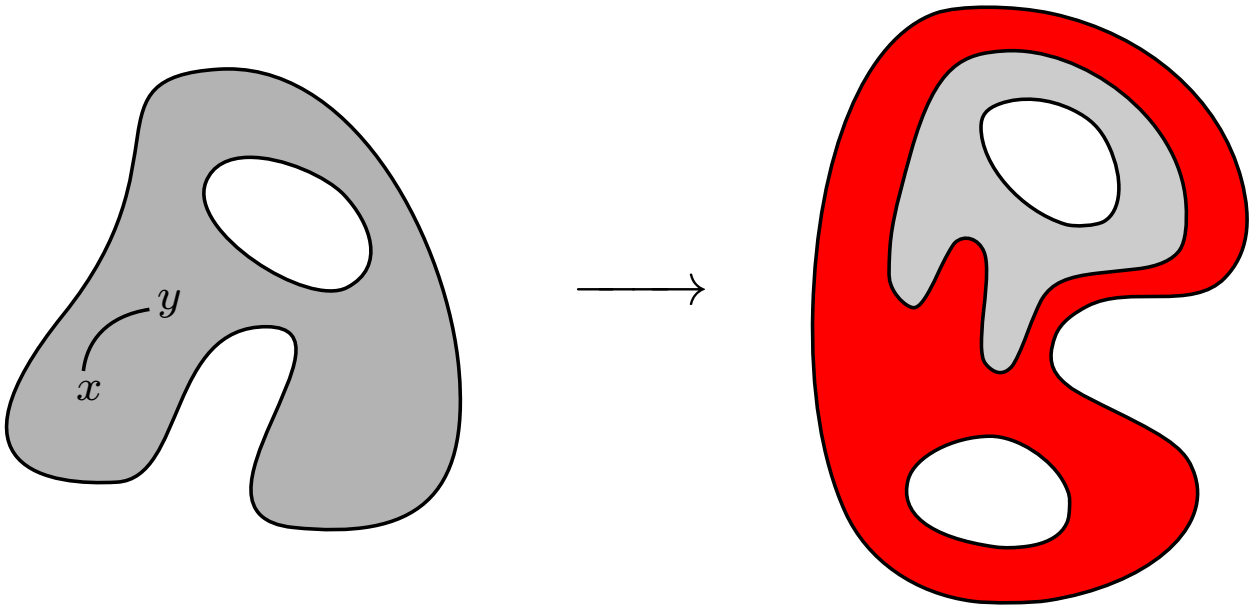
Homotopical interpretation

Types	`A`	as	*geometric objects*
Functions	`f : A -> B`	as	*continuous mappings*
Elements	`x, y : A`	as	*points*



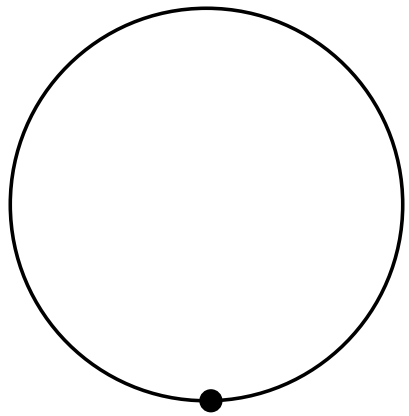
Homotopical interpretation

Types	`A`	as	*geometric objects*
Functions	`f : A -> B`	as	*continuous mappings*
Elements	`x, y : A`	as	*points*
Equality terms	`p : eq A x y`	as	*paths*



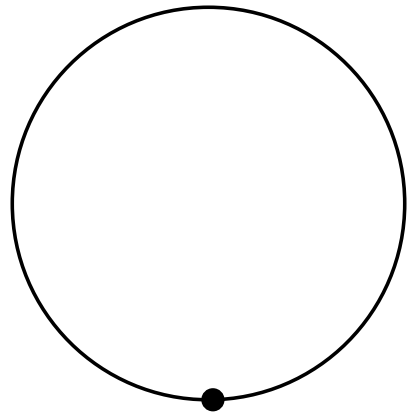
Higher inductive types 1

```
Inductive circle : Type :=
  | base : circle
  | loop : eq circle base base.
```



Higher inductive types 1

```
Inductive circle : Type :=
  | base : circle
  | loop : eq circle base base.
```

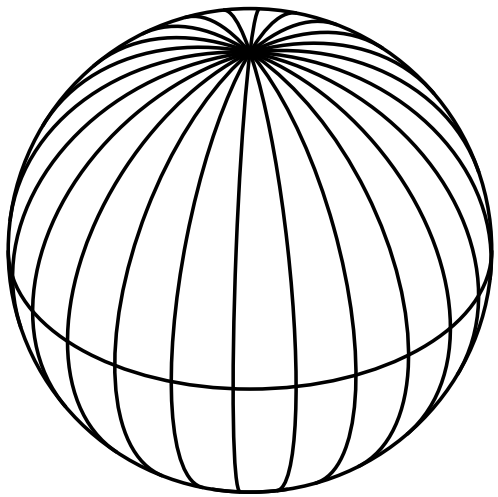


To define a function `f : circle -> A` it's enough to give

```
f base : A
f loop : eq A (f base) (f base)
```

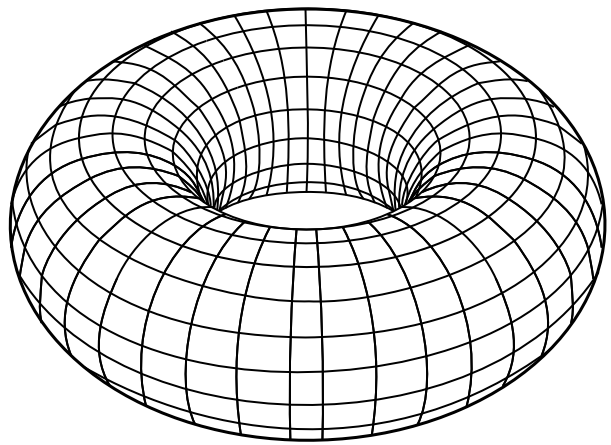
Higher inductive types 2

```
Inductive sphere : Type :=
  | N, S : sphere
  | meridian : circle -> eq sphere N S.
```



Higher inductive types 3

```
Definition torus : Type := circle * circle.
```



Application

Recall that

```
Inductive circle : Type :=
  | base : circle
  | loop : eq circle base base.
```

So by recursion can define

```
universalcover : circle -> Type
universalcover base = Int
universalcover loop = (+1) : eq Type Int Int
```

Application

Recall that

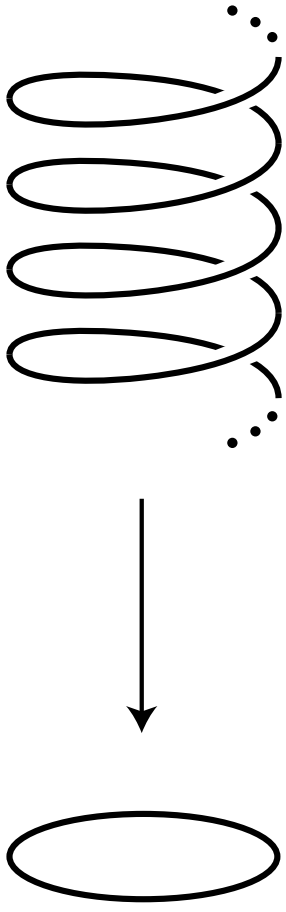
```
Inductive circle : Type :=
  | base : circle
  | loop : eq circle base base.
```

So by recursion can define

```
universalcover : circle -> Type
universalcover base = Int
universalcover loop = (+1) : eq Type Int Int
```

Makes sense because `eq Type A B` is the type of isomorphisms from `A` to `B` (the univalence axiom).

Application



$$(x : \mathtt{circle}) * \mathtt{universalcover}\ x$$

$$\mathtt{proj}_1 \downarrow$$

$$\mathtt{circle}$$

Why do we care?

- Basic constructions correspond:
  type isomorphism $\leftrightarrow$ *homotopy equivalence*.

Why do we care?

- Basic constructions correspond:
  type isomorphism $\leftrightarrow$ *homotopy equivalence*.
- Formalise mathematics, and reprove homotopy-theoretic results axiomatically:

$$\texttt{eq circle base base} \cong \texttt{Int} \quad \leftrightarrow \quad \pi_1(S^1) = \mathbb{Z}$$

Why do we care?

- Basic constructions correspond:
  type isomorphism $\leftrightarrow$ *homotopy equivalence*.
- Formalise mathematics, and reprove homotopy-theoretic results axiomatically:

$$\texttt{eq circle base base} \cong \texttt{Int} \quad \leftrightarrow \quad \pi_1(S^1) = \mathbb{Z}$$

  and more: Hopf fibration, $\pi_2(S^2)$, $\pi_3(S^2)$, Blakers–Massey theorem, Freudenthal suspension theorem, ...

Why do we care?

- Basic constructions correspond:
  type isomorphism $\leftrightarrow$ *homotopy equivalence*.
- Formalise mathematics, and reprove homotopy-theoretic results axiomatically:

$$\texttt{eq circle base base} \cong \texttt{Int} \quad \leftrightarrow \quad \pi_1(S^1) = \mathbb{Z}$$

  and more: Hopf fibration, $\pi_2(S^2)$, $\pi_3(S^2)$, Blakers–Massey theorem, Freudenthal suspension theorem, ...
- Extract algorithms from homotopy-theoretic proofs.

Why do we care?

- Basic constructions correspond:
  type isomorphism $\leftrightarrow$ *homotopy equivalence*.
- Formalise mathematics, and reprove homotopy-theoretic results axiomatically:

$$\texttt{eq circle base base} \cong \texttt{Int} \quad \leftrightarrow \quad \pi_1(S^1) = \mathbb{Z}$$

  and more: Hopf fibration, $\pi_2(S^2)$, $\pi_3(S^2)$, Blakers–Massey theorem, Freudenthal suspension theorem, ...
- Extract algorithms from homotopy-theoretic proofs.
- Results also valid in "non-classical" universes of homotopy theory.

To learn more...

http://homotopytypetheory.org/