

Introduction to Homotopy Type Theory

Niels van der Weide

Some History of Type Theory

- ▶ **1975:** Martin-Löf introduced **intuitionistic type theory** as a system for “formalizing” intuitionistic mathematics, as done, for instance, by Bishop
- ▶ **1982:** Martin-Löf described the relation between type theory and computer programming

Some History of Type Theory

- ▶ **1975**: Martin-Löf introduced **intuitionistic type theory** as a system for “formalizing” intuitionistic mathematics, as done, for instance, by Bishop
- ▶ **1982**: Martin-Löf described the relation between type theory and computer programming
- ▶ **1988**: Coquand and Huet described the **calculus of constructions**, which is similar to Martin-Löf type theory
- ▶ **1989**: the **Coq** (now **Rocq**) theorem prover, based on the calculus of constructions, got released
- ▶ **2013**: the **Lean** theorem prover, also based on the calculus of constructions, got released

Some History of Homotopy Type Theory

- ▶ **1989**: Hofmann and Streicher describe the groupoid model of type theory. This model was rather special: in this model, **equality is proof relevant**.
- ▶ **2009**: Awodey and Warren give a model of type theory using Quillen model categories. Their model extends Hofmann and Streicher's model to higher dimensions.
- ▶ **2009**: Voevodsky describes the **univalence axiom**, and shows that this axiom holds in the **simplicial sets model**. This model is the start of **homotopy type theory**

What is Homotopy Type Theory?

Homotopy type theory is a foundation for mathematics and computer science that incorporates ideas from homotopy theory into type theory

What is Homotopy Type Theory?

Homotopy type theory is a foundation for mathematics and computer science that incorporates ideas from homotopy theory into type theory

Key features of homotopy type theory

- ▶ Types are interpreted as **spaces**, and equality is interpreted as **paths** in a space. **Homotopies** are equalities of equalities.
- ▶ **The univalence axiom**: equality of types corresponds to isomorphism. We can prove similar principles for algebraic structures, like groups, rings, and so on.
- ▶ **Higher inductive types**: we can describe types inductively by specifying their points, paths, homotopies

Homotopy Type Theory and Mathematics

Homotopy type theory provides a nice setting to study

- ▶ **Homotopy theory and algebraic topology.** Homotopy type theory provides an abstract setting to study spaces with more generality (i.e., arbitrary ∞ -toposes). Statements from homotopy theory can be translated into type theory using that types represent spaces and equality represent paths. Examples: Blakers-Massey and Serre Finiteness.

Homotopy Type Theory and Mathematics

Homotopy type theory provides a nice setting to study

- ▶ **Homotopy theory and algebraic topology.** Homotopy type theory provides an abstract setting to study spaces with more generality (i.e., arbitrary ∞ -toposes). Statements from homotopy theory can be translated into type theory using that types represent spaces and equality represent paths. Examples: Blakers-Massey and Serre Finiteness.
- ▶ **Category Theory.** The univalence axiom is particularly useful for category theory, since it gives us many examples of categories where objects are identified up to isomorphism.

This Talk

Martin-Löf Type Theory

The Identity Type

Homotopy Type Theory

The Univalence Axiom and Higher Inductive Type

Conclusion

Table of Contents

Martin-Löf Type Theory

The Identity Type

Homotopy Type Theory

The Univalence Axiom and Higher Inductive Type

Conclusion

Dependent Types

Set theory	type theory
sets	types
elements	terms

Dependent Types

Set theory	type theory
sets	types
elements	terms

Type theory allows **dependent types**

↪ types indexed by some type

Dependent Types

Set theory	type theory
sets	types
elements	terms

Type theory allows **dependent types**

↪ types indexed by some type

Think of: families of sets

Dependent Types from the Viewpoint of Set Theory

$$\left. \begin{array}{cccc} Y_{x_0} & Y_{x_1} & Y_{x_2} & \dots \\ X = \{x_0, x_1, x_2, \dots\} \end{array} \right\} \{Y_x\}_{x \in X}$$

Some Examples of Dependent Types

$$\mathbb{R}^0 \quad \mathbb{R}^1 \quad \mathbb{R}^2 \quad \mathbb{R}^3$$

$$\mathbb{N} = \{0, 1, 2, 3, \dots\}$$

Recall: First-Order Logic

First-order logic

$$\Delta \vdash_{\Gamma} \varphi$$

Recall: First-Order Logic

First-order logic

$\Delta \vdash \varphi$
|
variables

Recall: First-Order Logic

First-order logic

$\Delta \vdash \varphi$
hypotheses

Recall: First-Order Logic

First-order logic

$\Delta \vdash \varphi$
conclusion

First-Order Logic versus Type Theory

Type theory comes with **dependent types**

First-order logic

$\Gamma \vdash \varphi$ form

φ is a formula with
variables from Γ

Type theory

$\Gamma \vdash A$ type

A is a type **depending**
on the variables in Γ

First-Order Logic versus Type Theory

Type theory comes with **terms**

First-order logic

$$\Delta \vdash \varphi$$

Type theory

$$\Gamma \vdash t : A$$

First-Order Logic versus Type Theory

Type theory comes with **terms**

First-order logic

$$\Delta \vdash \varphi$$

Type theory

$$\Gamma \vdash t : A$$

variables and hypotheses

First-Order Logic versus Type Theory

Type theory comes with **terms**

First-order logic

$$\Delta \vdash \varphi$$

Using the variables in Γ ,
 φ follows from Δ

Type theory

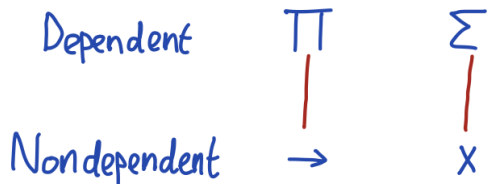
$$\Gamma \vdash t : A$$

Assuming Γ the
term t has type A

Key Examples of Dependent Types

- ▶ Π -types, which generalizes **function types**
- ▶ Σ -types, which generalizes **product types**
- ▶ The identity type, also known as propositional equality

Π -types and Σ -types



Π -types

$$\left. \begin{array}{cccc} Y_{x_0} & Y_{x_1} & Y_{x_2} & \dots \end{array} \right\} \{Y_x\}_{x \in X}$$
$$X = \{x_0, x_1, x_2, \dots\}$$

Π -type

$$\{f: X \rightarrow \bigcup_{x \in X} Y_x \mid \forall x. f_x \in Y_x\}$$

Introduction Rule for function types

First-order logic

$$\frac{\Delta, \varphi \vdash_{\Gamma} \psi}{\Delta \vdash_{\Gamma} \varphi \rightarrow \psi}$$

Type theory

$$\frac{\Gamma, x:A \vdash M:B}{\Gamma \vdash \lambda x. M:A \rightarrow B}$$

Introduction Rule for Π -types

First-order logic

$$\frac{\Delta \vdash_{x:A, \Gamma} \varphi}{\Delta \vdash_{\Gamma} \forall x:A. \varphi}$$

Type theory

$$\frac{\Gamma, x:A \vdash M : B[x]}{\Gamma \vdash \lambda x. M : \Pi(x:A). B[x]}$$

Elimination Rule for function types

First-order logic

$$\frac{\Delta \vdash_{\Gamma} \varphi \rightarrow \psi \quad \Delta \vdash_{\Gamma} \varphi}{\Delta \vdash_{\Gamma} \psi}$$

Type theory

$$\frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash x : A}{\Gamma \vdash f x : B}$$

Elimination Rule for Π -types

First-order logic

$$\frac{\Delta \vdash_{\Gamma} \forall x:A. \varphi \quad \Gamma \vdash t:A}{\Delta \vdash_{\Gamma} \varphi[x \mapsto t]}$$

Type theory

$$\frac{\Gamma \vdash f:\Pi(x:A), B[x] \quad \Gamma \vdash t:A}{\Gamma \vdash f\,t : B[x \mapsto t]}$$

Σ -types

$$\left. \begin{array}{cccc} Y_{x_0} & Y_{x_1} & Y_{x_2} & \dots \end{array} \right\} \{Y_x\}_{x \in X}$$
$$X = \{x_0, x_1, x_2, \dots\}$$

Σ -type

$$\{(x, y) \mid x \in X, y \in Y_x\}$$

Elimination Rule for product types

First-order logic

$$\frac{\Delta \vdash_{\Gamma} \varphi \wedge \psi}{\Delta \vdash_{\Gamma} \varphi}$$

Type theory

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi, t : A}$$

Elimination Rule for Σ -types

Product

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi, t : A}$$

Σ -type

$$\frac{\Gamma \vdash t : \Sigma(x:A).B[x]}{\Gamma \vdash \pi, t : A}$$

Elimination Rule for product types

First-order logic

$$\frac{\Delta \vdash_{\Gamma} \varphi \wedge \psi}{\Delta \vdash_{\Gamma} \psi}$$

Type theory

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi_2 t : B}$$

Elimination Rule for Σ -types

Product

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi_2 t : B}$$

Σ -type

$$\frac{\Gamma \vdash t : \Sigma(x:A), B[x]}{\Gamma \vdash \pi_2 t : B[x \mapsto \pi_1 t]}$$

Introduction Rule for product types

First-order logic

$$\frac{\Delta \vdash \varphi \quad \Delta \vdash \psi}{\Delta \vdash \varphi \wedge \psi}$$

Type theory

$$\frac{\Gamma \vdash t_1 : A \quad \Gamma \vdash t_2 : B}{\Gamma \vdash \langle t_1, t_2 \rangle : A \times B}$$

Introduction Rule for Σ -types

Product

$$\frac{\Gamma \vdash t_1 : A \quad \Gamma \vdash t_2 : B}{\Gamma \vdash \langle t_1, t_2 \rangle : A \times B}$$

Σ -type

$$\frac{\Gamma \vdash t_1 : A \quad \Gamma \vdash t_2 : B[x \mapsto \pi, t]}{\Gamma \vdash \langle t_1, t_2 \rangle : \Sigma(x:A), B[x]}$$

Table of Contents

Martin-Löf Type Theory

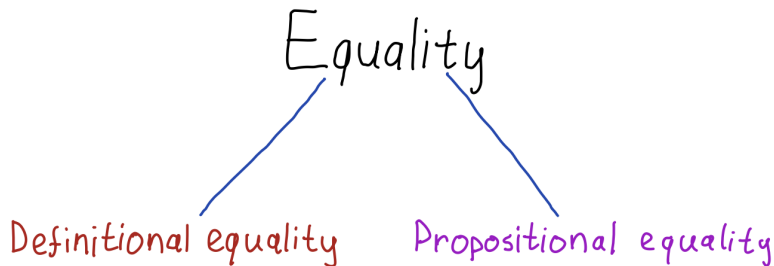
The Identity Type

Homotopy Type Theory

The Univalence Axiom and Higher Inductive Type

Conclusion

Equality in Type Theory



Definitional and Propositional Equality

Definitional

Propositional

decided via
computation

proven using
derivations

equivalence
relation on
terms and types

a type

Some Definitional Equalities

$$(\lambda x. M) t \equiv M[x \mapsto t]$$

$$\pi_1 \langle t_1, t_2 \rangle \equiv t_1$$

$$\pi_2 \langle t_1, t_2 \rangle \equiv t_2$$

Note: we have a separate judgement for definitional equality

The Identity Type

- ▶ Given a type A and $x : A$ and $y : A$, we have a **type** $x = y$. We call this type the **identity type** and sometimes **propositional equality**
- ▶ Terms of type $x = y$ represent proofs that x and y are equal
- ▶ Note: we have types like $\prod (x : A), x = x$, but we cannot use definitional equality as a type

Rules for the Identity Type

$$\frac{\Gamma \vdash x:A}{\Gamma \vdash \text{refl}_x : x=x}$$

Rules for the Identity Type

$$\frac{\Gamma, x:A, y:A, p:x=y \vdash B[x,y,p] \text{ type} \quad \Gamma, x:A \vdash b: B[x, x, \text{refl}_x] \quad \Gamma \vdash p: x=y}{\Gamma \vdash \text{ind}(b,p): B[x,y,p]}$$

This says: equality is the **least reflexive relation**

Rules for the Identity Type

Given

- $\Gamma, x:A, y:A, p:x=y \vdash B[x, y, p] \text{ type}$
- $\Gamma, x:A \vdash b: B[x, x, \text{refl}_x]$
- $\Gamma \vdash p: x=y$

we get

$$\Gamma \vdash \text{ind}(b, p): B[x, y, p]$$

This says: equality is the **least reflexive relation**

Note the Minimality

In first-order logic, we postulate

- ▶ Equality is reflexive, symmetric, and transitive
- ▶ Equality is substitutive: equals can be substituted for each other. In particular, functions preserve equality.

In type theory, **there are no postulated rules for symmetry, transitivity, and substitution. Such rules are derived.**

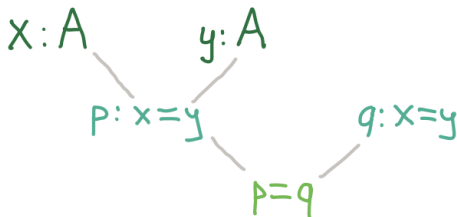
Iterating the Identity Type

Note: we can make the following types

$$\begin{array}{ccc} x:A & & y:A \\ & \searrow \quad \swarrow & \\ & x=y & \end{array}$$

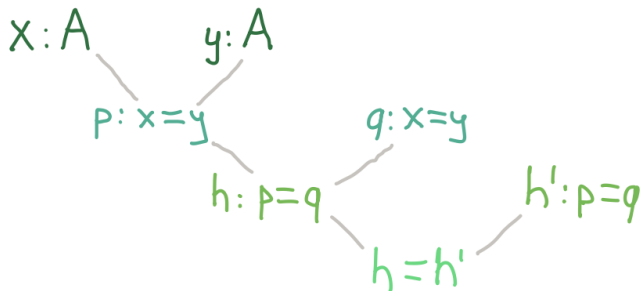
Iterating the Identity Type

Note: we can make the following types



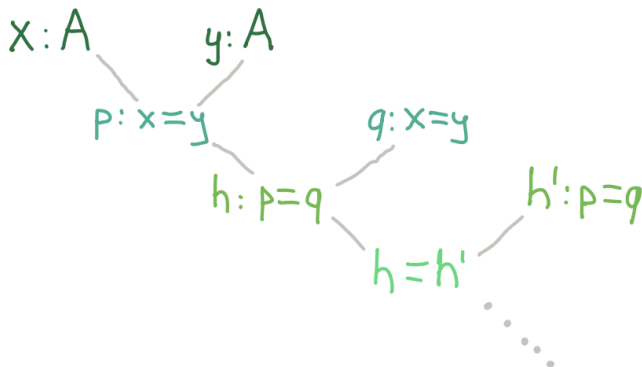
Iterating the Identity Type

Note: we can make the following types



Iterating the Identity Type

Note: we can make the following types



Derivation of Symmetry

We have

- $\Gamma, x:A, y:A, p:x=y \vdash y=x$ type
- $\Gamma, x:A \vdash \text{refl}_x : x=x$
- $\Gamma \vdash p : x=y$

so we get

$$\Gamma \vdash \text{sym}(p) : y=x$$

Derivation of Transitivity

We have

$$\bullet \Gamma, x:A, y:A, p:x=y \vdash \Pi z:A, y=z \longrightarrow x=z \text{ type}$$

$$\bullet \Gamma, x:A \vdash \lambda z p. p : \Pi z:A, x=z \longrightarrow x=z$$

$$\bullet \Gamma \vdash p : x=y$$

so we get

$$\Gamma \vdash \text{trans}(p) : \Pi z:A, y=z \longrightarrow x=z$$

Preservation of Equality

Let $f: A \rightarrow B$

We have

- $\Gamma, x:A, y:A, p:x=y \vdash fx=fy$ type
- $\Gamma, x:A \vdash \text{refl}_x : fx = fx$
- $\Gamma \vdash p:x=y$

so we get

$$\Gamma \vdash \text{op } f \ p : fx = fy$$

Derivation of Substitution

Let $\Gamma, x:A \vdash B[x]$ type

We have

- $\Gamma, x:A, y:A, p:x=y \vdash B[x \mapsto x] \rightarrow B[x \mapsto y]$ type
- $\Gamma, x:A \vdash \lambda z. z : B[x \mapsto x] \rightarrow B[x \mapsto x]$
- $\Gamma \vdash p : x=y$

so we get

$$\Gamma \vdash \text{transport}^B p : B[x \mapsto x] \rightarrow B[x \mapsto y]$$

Table of Contents

Martin-Löf Type Theory

The Identity Type

Homotopy Type Theory

The Univalence Axiom and Higher Inductive Type

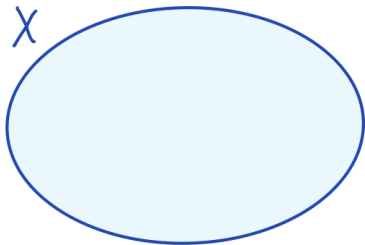
Conclusion

Interpreting the Identity Type

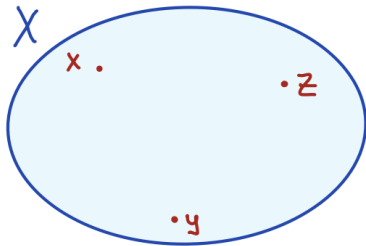
There are multiple ways to view the identity type in mathematics

Set theoretic	Topological
$x=y$ means that x and y are exactly the same	$x=y$ means that we have a continuous transformation from x to y , i.e. a path
there is at most one proof of $x=y$	there might be different proofs of $x=y$

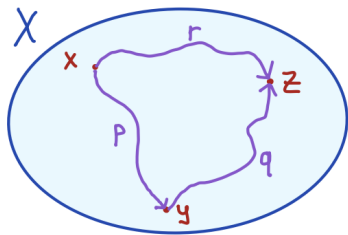
Types as Spaces



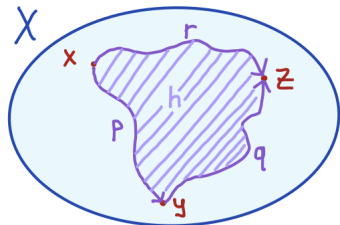
Terms as Points



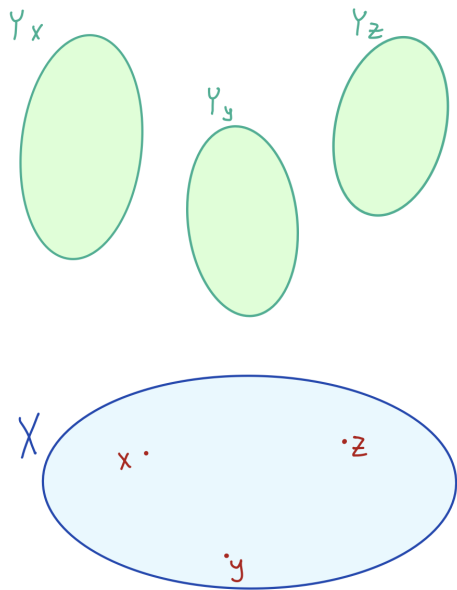
Equality as Paths



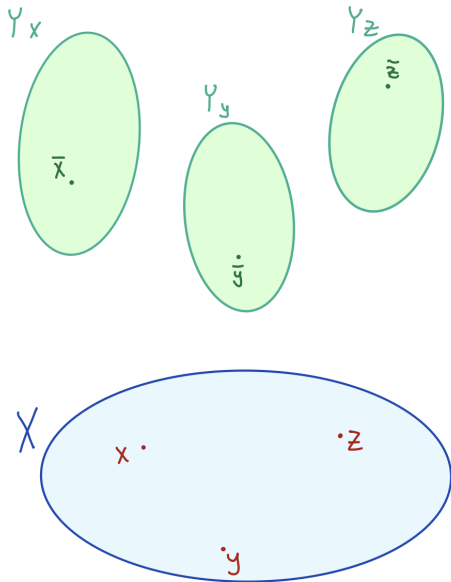
Higher Equalities as Homotopies



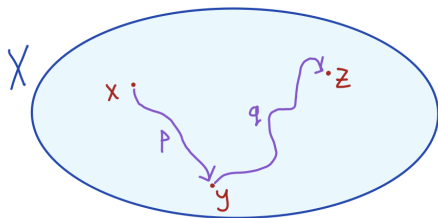
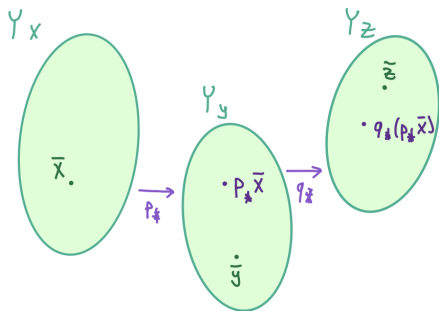
Dependent Types as Fibrations



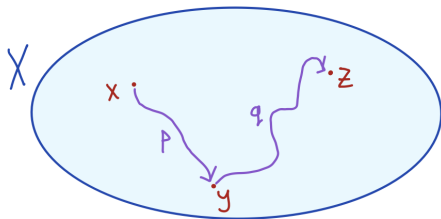
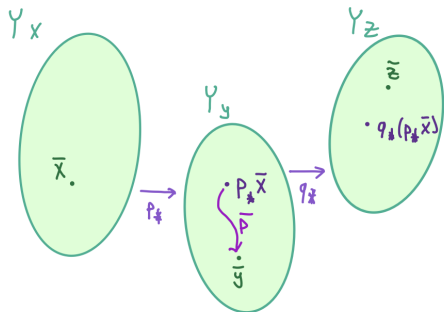
Dependent Types as Fibrations



Dependent Types as Fibrations



Dependent Types as Fibrations



Dependent Types as Fibrations

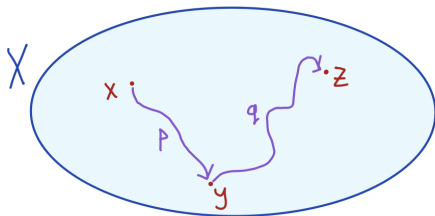
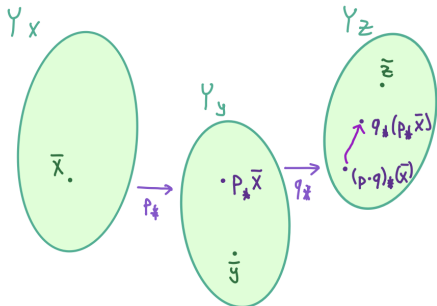


Table of Contents

Martin-Löf Type Theory

The Identity Type

Homotopy Type Theory

The Univalence Axiom and Higher Inductive Type

Conclusion

The Univalence Axiom

- ▶ Earlier in this talk, we discussed the identity type, and we discussed its type theoretic rules
- ▶ **Problem:** the identity type is “underspecified”
- ▶ We cannot prove that equality of functions is pointwise equality, and we do not have a nice characterisation for the equality of types

The Univalence Axiom

- ▶ Earlier in this talk, we discussed the identity type, and we discussed its type theoretic rules
- ▶ **Problem:** the identity type is “underspecified”
- ▶ We cannot prove that equality of functions is pointwise equality, and we do not have a nice characterisation for the equality of types

The **univalence axiom** fixes these defects of the identity type. It says: to give $p : A = B$ for types A and B , it suffices to give

- ▶ $f : A \rightarrow B$
- ▶ $g : B \rightarrow A$
- ▶ forall all $a : A$ an equality $q : g(f(a)) = a$
- ▶ forall all $b : B$ an equality $r : f(g(b)) = b$

Consequences of the Univalence Axiom

- ▶ Function extensionality (represent functions via their graphs)
- ▶ Two groups are equal if they are isomorphic
- ▶ Two rings are equal if they are isomorphic

We can find similar structure identity principles for a wide variety of mathematical structures.

Main benefit: we can treat isomorphic objects as equals

Higher Inductive Types

- ▶ **Inductive types**: given by constructors for their points
- ▶ **Higher inductive types**: given by constructors for their points, **paths**, **homotopies**, and so on

Higher Inductive Types

- ▶ **Inductive types:** given by constructors for their points
- ▶ **Higher inductive types:** given by constructors for their points, **paths**, **homotopies**, and so on

The elimination rule is derived from the constructors.

Lists as an Inductive Type

Inductive list ($A : \text{Type}$): $\text{Type} :=$

| nil : list A

| cons : $A \rightarrow \text{list } A \rightarrow \text{list } A$

Lists as an Inductive Type

Given

- $\Gamma, x: \text{list } A \vdash B[x] \text{ type}$

Lists as an Inductive Type

Given

- $\Gamma, x: \text{list } A \vdash B[x] \text{ type}$
-

$\text{nil} : \text{list } A$

Lists as an Inductive Type

Given

- $\Gamma, x: \text{list } A \vdash B[x] \text{ type}$
- $\Gamma \vdash n: B[x \mapsto \text{nil}]$

$\text{nil} : \text{list } A$

Lists as an Inductive Type

Given

- $\Gamma, x: \text{list } A \vdash B[x] \text{ type}$
- $\Gamma \vdash n: B[x \mapsto \text{nil}]$
-

$\text{cons}: A \rightarrow \text{list } A \rightarrow \text{list } A$

Lists as an Inductive Type

Given

- $\Gamma, x: \text{list } A \vdash B[x] \text{ type}$
- $\Gamma \vdash n: B[x \mapsto \text{nil}]$
- $\Gamma, a: A,$

$\text{cons}: A \rightarrow \text{list } A \rightarrow \text{list } A$

Lists as an Inductive Type

Given

- $\Gamma, x: \text{list } A \vdash B[x] \text{ type}$
- $\Gamma \vdash n: B[x \mapsto \text{nil}]$
- $\Gamma, a: A, l: \text{list } A$

$\text{cons}: A \rightarrow \text{list } A \rightarrow \text{list } A$

Lists as an Inductive Type

Given

- $\Gamma, x: \text{list } A \vdash B[x]$ type
- $\Gamma \vdash n: B[x \mapsto \text{nil}]$
- $\Gamma, a: A, l: \text{list } A, IH: B[x \mapsto l]$

$\text{cons}: A \rightarrow \text{list } A \rightarrow \text{list } A$

Lists as an Inductive Type

Given

- $\Gamma, x: \text{list } A \vdash B[x]$ type
- $\Gamma \vdash n: B[x \mapsto \text{nil}]$
- $\Gamma, a: A, l: \text{list } A, IH: B[x \mapsto l] \vdash c(a, l, IH): B[x \mapsto \text{cons } a\ l]$
- $\Gamma \vdash l: \text{list } A$

Lists as an Inductive Type

Given

- $\Gamma, x: \text{list } A \vdash B[x]$ type
- $\Gamma \vdash n: B[x \mapsto \text{nil}]$
- $\Gamma, a: A, l: \text{list } A, IH: B[x \mapsto l] \vdash c(a, l, IH): B[x \mapsto \text{cons } a\ l]$
- $\Gamma \vdash l: \text{list } A$

we get

$$\Gamma \vdash \text{listrec}(n, c, l): B[x \mapsto l]$$

Vectors as an Inductive Type

Inductive $\text{vec} (A:\text{Type}): \mathbb{N} \rightarrow \text{Type} :=$

| $\text{nil} : \text{vec } A \ 0$

| $\text{cons} : \prod (n:\mathbb{N}), A \rightarrow \text{vec } A \ n \rightarrow \text{vec } A \ (n+1)$

Vectors as an Inductive Type

Given

- $\Gamma, n:\mathbb{N}, x:\text{vec } A\ n \vdash B[x] \text{ type}$

Vectors as an Inductive Type

Given

- $\Gamma, n:\mathbb{N}, x:\text{vec } A_n \vdash B[x] \text{ type}$
-

$\text{nil} : \text{vec } A_0$

Vectors as an Inductive Type

Given

- $\Gamma, n:\mathbb{N}, x:\text{vec } A_n \vdash B[x] \text{ type}$
- $\Gamma \vdash m : B[\text{nil}]$

$\text{nil} : \text{vec } A_0$

Vectors as an Inductive Type

Given

- $\Gamma, n:N, x:\text{vec } A \ n \vdash B[x] \text{ type}$
- $\Gamma \vdash m : B[\text{nil}]$
-

$$\prod (n:N), A \rightarrow \text{vec } A \ n \rightarrow \text{vec } A \ (n+1)$$

Vectors as an Inductive Type

Given

- $\Gamma, x: \text{list } A \vdash B[x] \text{ type}$
- $\Gamma \vdash m : B[\text{nil}]$
- $\Gamma, n: \mathbb{N}, a: A,$

$$\Pi(n: \mathbb{N}), A \rightarrow \text{vec } A \ n \rightarrow \text{vec } A \ (n+1)$$

Vectors as an Inductive Type

Given

- $\Gamma, x: \text{list } A \vdash B[x] \text{ type}$
- $\Gamma \vdash m: B[\text{nil}]$
- $\Gamma, n: \mathbb{N}, a: A, l: \text{vec } A\ n,$

$$\Pi (n: \mathbb{N}), A \rightarrow \text{vec } A\ n \rightarrow \text{vec } A\ (n+1)$$

Vectors as an Inductive Type

Given

- $\Gamma, x: \text{list } A \vdash B[x] \text{ type}$
- $\Gamma \vdash m: B[\text{nil}]$
- $\Gamma, n: \mathbb{N}, a: A, l: \text{vec } A\ n, IH: B[l] \vdash$

$$\Pi (n: \mathbb{N}), A \rightarrow \text{vec } A\ n \rightarrow \text{vec } A\ (n+1)$$

Vectors as an Inductive Type

Given

- $\Gamma, x: \text{list } A \vdash B[x] \text{ type}$
- $\Gamma \vdash m: B[\text{nil}]$
- $\Gamma, n: \mathbb{N}, a: A, l: \text{vec } A_n, IH: B[l] \vdash c(n, a, l, IH): B[\text{cons}_n a l]$

Vectors as an Inductive Type

Given

- $\Gamma, x: \text{list } A \vdash B[x] \text{ type}$
- $\Gamma \vdash m: B[\text{nil}]$
- $\Gamma, n: \mathbb{N}, a: A, l: \text{vec } A\ n, IH: B[l] \vdash c(n, a, l, IH): B[\text{cons}_n\ a\ l]$
- $\Gamma \vdash l: \text{vec } A\ n$

Vectors as an Inductive Type

Given

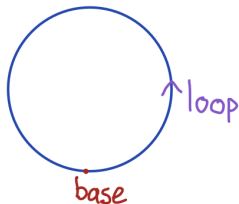
- $\Gamma, x: \text{list } A \vdash B[x] \text{ type}$
- $\Gamma \vdash m: B[\text{nil}]$
- $\Gamma, n: \mathbb{N}, a: A, l: \text{vec } A\ n, IH: B[l] \vdash c(n, a, l, IH): B[\text{cons}_n\ a\ l]$
- $\Gamma \vdash l: \text{vec } A\ n$

we get

$$\Gamma \vdash \text{vecrec}(m, c, l): B[l]$$

Example: The Circle

In homotopy type theory, we can define the circle in a rather nice way.



Example: The Circle

In essence, the circle is generated by its base point and the loop. Using higher-inductive types, we can write the circle as follows.

$HIT\ S' :=$

| $base : S'$

| $loop : base = base$

Example: The Circle

In essence, the circle is generated by its base point and the loop. Using higher-inductive types, we can write the circle as follows.

$HIT\ S' :=$

| $base : S'$

| $loop : base = base$

We look at some rules for the circle in the next two slides

Example: The Circle

$$\Gamma \vdash \text{base} : S'$$
$$\Gamma \vdash \text{loop} : \text{base} = \text{base}$$

Example: The Circle

Given

- $\Gamma, x:S' \vdash B[x]$ type
- $\Gamma \vdash b:B[x \mapsto \text{base}]$
- $\Gamma \vdash l: \text{transport}^B \text{ loop } b = b$
- $\Gamma \vdash z:S'$

We get

$$\Gamma \vdash S' \text{ ind}(b, l, z): B[x \mapsto z]$$

Example: The Circle

Given

- $\Gamma, x:S' \vdash B[x] \text{ type}$
- $\Gamma \vdash b:B[x \mapsto \text{base}]$
- $\Gamma \vdash l: \text{transport}^B \text{ loop } b = b$
- $\Gamma \vdash z:S'$

We get

$$\Gamma \vdash S' \text{ ind } (b, l, z): B[x \mapsto z]$$

Table of Contents

Martin-Löf Type Theory

The Identity Type

Homotopy Type Theory

The Univalence Axiom and Higher Inductive Type

Conclusion

Conclusion

- ▶ **Type theory** is a foundation for mathematics and computer science that has proven itself to be especially useful for proof assistants.
- ▶ Equality is particularly interesting in type theory, and it comes in two forms: **definitional and propositional equality**
- ▶ The starting point of homotopy type theory is viewing proofs of **equality as paths** in some topological space
- ▶ Key features: **univalence axiom and higher inductive types**
- ▶ Univalence: types are identified if they are isomorphic
- ▶ Higher inductive types: define a type by specifying its points, paths, homotopies, ...