Homotopy Type Theory

A Crash Course

Siva Somayyajula

May 1, 2018

A Review of MLTT

- Martin-Löf Type Theory (MLTT) is a programming language, but it's also a language for mathematical structures...

type	set
U	universe of sets
`void`	$\emptyset$
`unit`	singleton $\{\star\}$
$\mathbb{Z}$	integers
$A + B$	disjoint union
$A \times B$	Cartesian product
$A \to B$	function space

Table 1: Sets-as-types

A Review of MLTT

- . . . and a language for mathematical reasoning

proposition	type
$\perp$	$\texttt{void}$
$\top$	$\texttt{unit}$
$A \vee B$	$A + B$
$A \wedge B$	$A \times B$
$A \implies B$	$A \to B$
$\neg A$	$A \to \texttt{void}$
$x \in A$, predicate $P(x)$	type family $P : A \to U$
$\forall_{a \in A} P(a)$	$(a : A) \to P(a)$
$\exists_{a \in A} P(a)$	$(a : A) \times P(a)$
$a, b \in A, a = b$	$a = b$ or $Id_A(a, b)$

Table 2: Propositions-as-types

The Identity Type

The Identity Type

Term Introduction

Given $a : A$, $\mathrm{refl}_a : a = a$.

The Identity Type

Term Introduction

Given $a : A$, $\mathrm{refl}_a : a = a$.

Theorem

$1 + 1 = 2$

Proof.

By refl_2. $\square$

The Identity Type

Term Introduction

Given $a : A$, $\mathrm{refl}_a : a = a$.

Term Elimination: Paulin-Mohring's J

Given a type family $P : (a, b : A) \to a = b \to U$:

$$\mathrm{J} : P(a, a, \mathrm{refl}_a) \to (p : a = b) \to P(a, b, p)$$

Computationally, we have $\mathrm{J}(r, refl_a) \triangleq r$.

The Identity Type

Theorem (Symmetry of equality)

For $a, b : A$ and $p : a = b$, we have $p^{-1} : b = a$.

Proof.

Let $P(a, b, p) \triangleq b = a$.
Then, $p^{-1} \triangleq J(\mathsf{refl}_a, p)$ is sufficient. □

The Identity Type

Theorem (Transitivity of equality)

For $a, b, c : A$, $p : a = b$, and $q : b = c$, we have $p \cdot q : a = c$.

Proof.

Let $P(a, b, p) \triangleq b = c \to a = c$.
Then, $p \cdot \_ \triangleq J(\mathrm{id}_{Id_A(a,c)}, p)$ is sufficient. $\square$

Uniqueness of Identity Proofs?

- At runtime, is every $p : a = b$ actually refl?

Uniqueness of Identity Proofs?

- At runtime, is every $p : a = b$ actually refl?

Definition (Principle UIP)

For $p, q : a = b$, $p = q$.

Uniqueness of Identity Proofs?

- At runtime, is every $p : a = b$ actually refl?

Definition (Principle UIP)

For $p, q : a = b$, $p = q$.

- It can't be proven with just J, so it needs to be added as an axiom

Uniqueness of Identity Proofs?

- At runtime, is every $p : a = b$ actually refl?

Definition (Principle UIP)

For $p, q : a = b$, $p = q$.

- It can't be proven with just J, so it needs to be added as an axiom
- But we won't, we'll add more terms to the identity type!

Univalence

- Motivation: MLTT can't distinguish between isomorphic types, but you have to prove it for every type

Univalence

- Motivation: MLTT can't distinguish between isomorphic types, but you have to prove it for every type

Definition (Homotopy)

Given $f, g : A \to B$, $f \sim g$ iff for all x, $f(x) = g(x)$.

Univalence

- Motivation: MLTT can't distinguish between isomorphic types, but you have to prove it for every type

Definition (Homotopy)

Given $f, g : A \to B$, $f \sim g$ iff for all x, $f(x) = g(x)$.

Definition (Equivalence)

f is a equivalence iff there exists $g : B \to A$ such that $f \circ g \sim \mathrm{id}_B$ and $g \circ f \sim \mathrm{id}_A$.
$A \simeq B$ iff there exists a equivalence $f : A \to B$.

Univalence

Theorem

For all types A and B, $idtoeqv : A = B \to A \simeq B$.

Proof.

Let $P(A, B, p) \triangleq A \simeq B$.
Since id_A is an equivalence, $\mathrm{idtoeqv}(p) \triangleq J(\mathrm{id}_A, p)$ is sufficient. $\square$

Univalence

Definition (Univalence)

idtoeqv is an equivalence i.e. $(A \simeq B) \simeq (A = B)$.

Univalent Type Theory

- UTT = MLTT + univalence axiom

Univalent Type Theory

- UTT = MLTT + univalence axiom

Theorem

The following term has been added to the system:

- $ua : A \simeq B \to A = B$

Proof.

From the definition of the univalence axiom. $\square$

Univalent Type Theory

- UTT = MLTT + univalence axiom

Theorem

The following term has been added to the system:

- $ua : A \simeq B \to A = B$

Proof.

From the definition of the univalence axiom. □

- What happens to $\mathrm{J}(x, \mathrm{ua}(f))$?

Generative Programming

Univalence

Generative Programming

Univalence $\implies$ UTT *actually* can't distinguish between isomorphic types

Generative Programming

$$\begin{aligned} \text{Univalence} &\implies \text{UTT } \textit{actually} \text{ can't distinguish between isomorphic types} \\ &\implies \text{Code for one type is code for an equivalent type} \end{aligned}$$

Generative Programming

$$\begin{aligned} \text{Univalence} &\implies \text{UTT } \textit{actually} \text{ can't distinguish between isomorphic types} \\ &\implies \text{Code for one type is code for an equivalent type} \\ &\implies \text{Generative programming for free} \end{aligned}$$

Lists

- How to represent lists of elements over some type A?

Lists

- How to represent lists of elements over some type A?

Definition (Lists)

$$\mathsf{List}(A) \triangleq \texttt{unit} + (A \times \mathsf{List}(A))$$

Vectors

- A type of n-dimensional vectors over a type A

Vectors

- A type of n-dimensional vectors over a type A

Definition (Vector)

$$\begin{aligned} \mathrm{Vector}(A, 0) &\triangleq \texttt{unit} \\ \mathrm{Vector}(A, n+1) &\triangleq A \times \mathrm{Vector}(A, n) \end{aligned}$$

Lists vs. Vectors

- They're basically the same, but vectors are indexed by length, so we "hide" it using the dependent pair type

Lists vs. Vectors

- They're basically the same, but vectors are indexed by length, so we "hide" it using the dependent pair type

Theorem

We have $\mathit{ListEqVec} : \mathit{List}(A) \simeq (n : \mathbb{N}) \times \mathit{Vector}(A, n)$.

Proof.

Convert list ℓ to $(\mathrm{length}(\ell), v)$ where v is the equivalent vector and length-vector pair (n, v) to the equivalent list ℓ. $\square$

Transport Design Pattern

- *Leibniz's law* is a code-generating powerhouse

Transport Design Pattern

- *Leibniz's law* is a code-generating powerhouse

Theorem

For a type A, $a, b \in A$, and a type family $Q : A \to U$, we have $transport^Q : a = b \to Q(a) \to Q(b)$

Transport Design Pattern

- *Leibniz's law* is a code-generating powerhouse

Theorem

For a type A*,* $a, b \in A$*, and a type family* $Q : A \to U$*, we have* $\mathit{transport}^Q : a = b \to Q(a) \to Q(b)$

Proof.

Let $P(a, b, p) \triangleq Q(a) \to Q(b)$. Then, $\mathrm{transport}(p, -) \triangleq J(\mathrm{id}_{Q(a)}, p)$ is sufficient. □

Transport Design Pattern

- *Leibniz's law* is a code-generating powerhouse

Theorem

For a type A, $a, b \in A$, and a type family $Q : A \to U$, we have $\mathit{transport}^Q : a = b \to Q(a) \to Q(b)$

Proof.

Let $P(a, b, p) \triangleq Q(a) \to Q(b)$. Then, $\mathrm{transport}(p, -) \triangleq J(\mathrm{id}_{Q(a)}, p)$ is sufficient. □

- What does e.g. $\mathrm{transport}^{\mathrm{id}_U}(\mathrm{ua}(f), x)$ do? It should evaluate to $f(x)$! (Licata, '12)
- That's a computation rule for ua!

Generative Programming

- Case study: auto-generate monoid instance for vectors from one for arrays

Definition (Family of monoids)

$$
\begin{aligned}
\mathrm{Monoid}(A) \triangleq\ & (\cdot : A \to A \to A) \\
& \times (e : A) \\
& \times ((a, b, c : A) \to a \cdot (b \cdot c) = (a \cdot b) \cdot c) \\
& \times ((a : A) \to e \cdot a = a) \\
& \times ((a : A) \to a \cdot e = a)
\end{aligned}
$$

Generative Programming

Theorem (List monoid)

For all types A, we have $ListMon : Monoid(List(A))$.

Proof.

Given a suitable append operation, it suffices to let $\mathsf{ArrMon} \triangleq (\mathsf{append}, \mathsf{empty}, \ldots)$ where $\mathsf{empty} \triangleq \mathsf{inl}(\star)$. $\square$

Theorem (Vector monoid)

For all types A, we have $\mathit{VecMon} : \mathit{Monoid}((n : \mathbb{N}) \times \mathit{Vector}(A, n))$.

Proof.

It suffices to let $\mathrm{VecMon} \triangleq \mathrm{transport}^{\mathrm{Monoid}}(\mathrm{ua}(\mathrm{ListEqVec}), \mathrm{ListMon})$. □

Generative Programming

Theorem (Vector monoid)

For all types A, we have $VecMon : Monoid((n : \mathbb{N}) \times Vector(A, n))$.

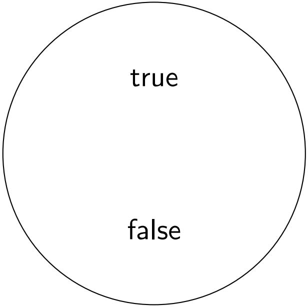
Proof.

It suffices to let $\mathrm{VecMon} \triangleq \mathrm{transport}^{\mathrm{Monoid}}(\mathrm{ua}(\mathrm{ListEqVec}), \mathrm{ListMon})$. $\square$

- Is this not the coolest thing you've ever seen? We can do this for any type family!

Types-as-Spaces

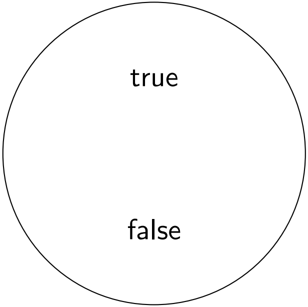
type theory	homotopy theory
type	space
term	point



The space of booleans

Types-as-Spaces

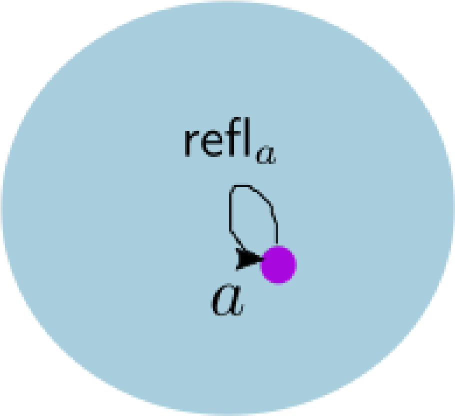
type theory	homotopy theory
type	space
term	point
$Id_A(a, b)$	path space



The space of booleans

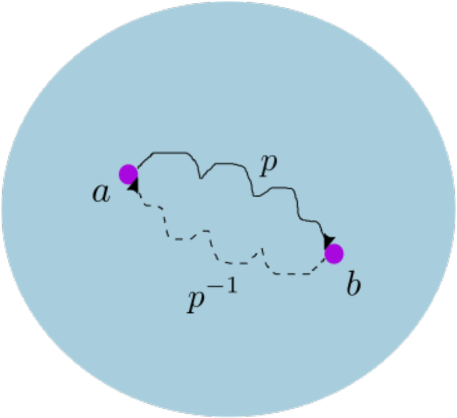
Identity-as-Paths

Identity-as-Paths



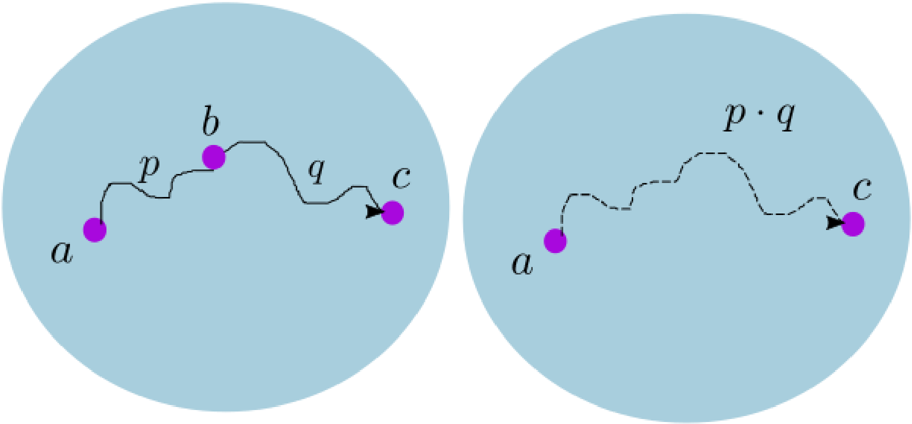
Reflexivity = Constant Path

Identity-as-Paths



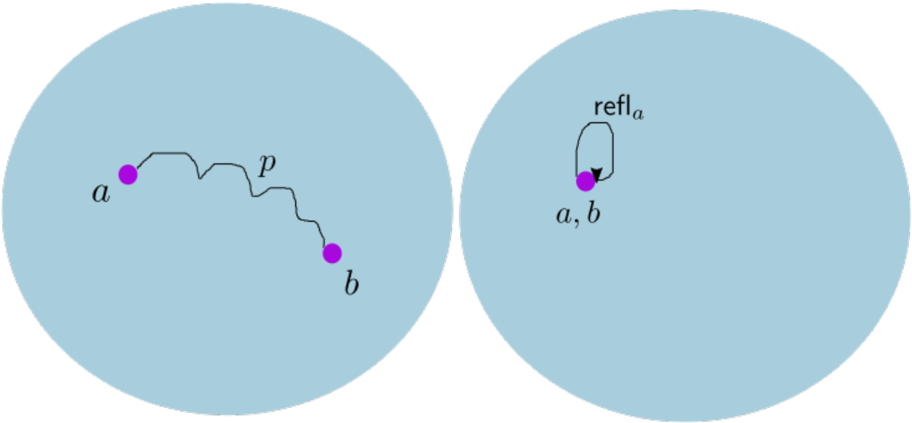
Symmetry = Path Inversion

Identity-as-Paths



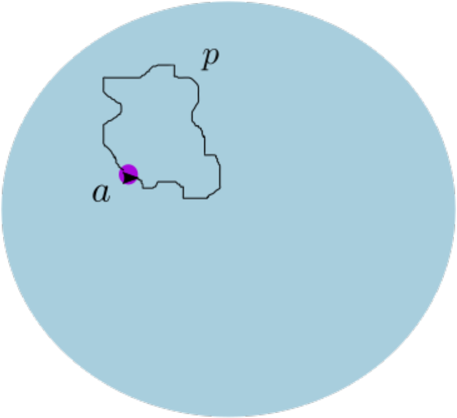
Transitivity = Path Composition

Identity-as-Paths



J = Free Endpoint Retraction

Identity-as-Paths



J = Free Endpoint Retraction $\implies$ no UIP

Fundamental Groupoid

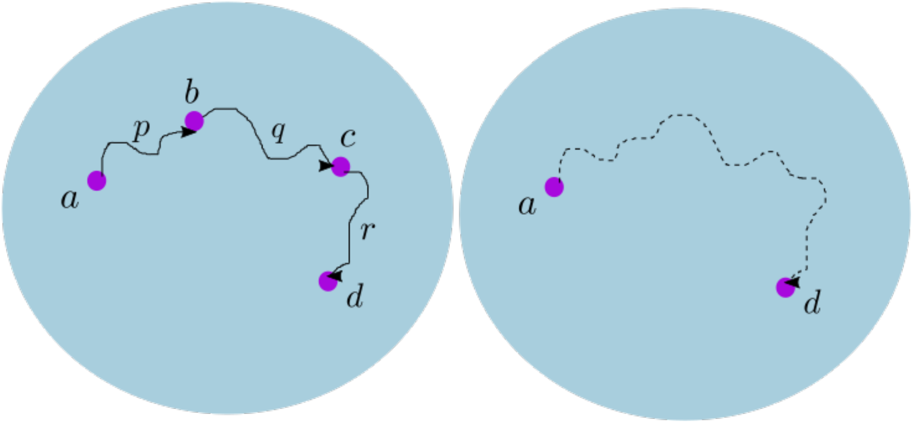
Theorem

Every type A induces a groupoid with $^{-1}$ and $\cdot$

Fundamental Groupoid

Theorem

Every type A induces a groupoid with $^{-1}$ and $\cdot$

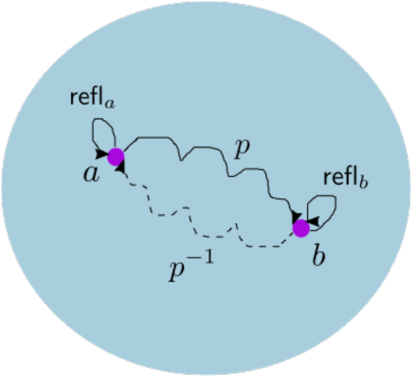


$$p \cdot (q \cdot r) = (p \cdot q) \cdot r$$

Fundamental Groupoid

Theorem

Every type A induces a groupoid with $^{-1}$ and $\cdot$



$$\mathsf{refl}_a \cdot p = p \cdot \mathsf{refl}_b = p$$
$$p \cdot p^{-1} = \mathsf{refl}_a$$
$$p^{-1} \cdot p = \mathsf{refl}_b$$

The Homotopy Model

- Types-as-spaces up-to *homotopy equivalence*
- Same as equivalences in last section $\implies$ this model is consistent with univalence

The Homotopy Model

- Types-as-spaces up-to *homotopy equivalence*
- Same as equivalences in last section $\implies$ this model is consistent with univalence
- Can we do topology/homotopy theory in UTT?

The Homotopy Model

- Types-as-spaces up-to *homotopy equivalence*
- Same as equivalences in last section $\implies$ this model is consistent with univalence
- Can we do topology/homotopy theory in UTT? Eh...

Fundamental Group

Definition (Fundamental Group)

The *fundamental group* of A based at $a : A$ is $\pi_1(A, a) \triangleq Id_A(a, a)$, with group structure endowed by the fundamental groupoid.

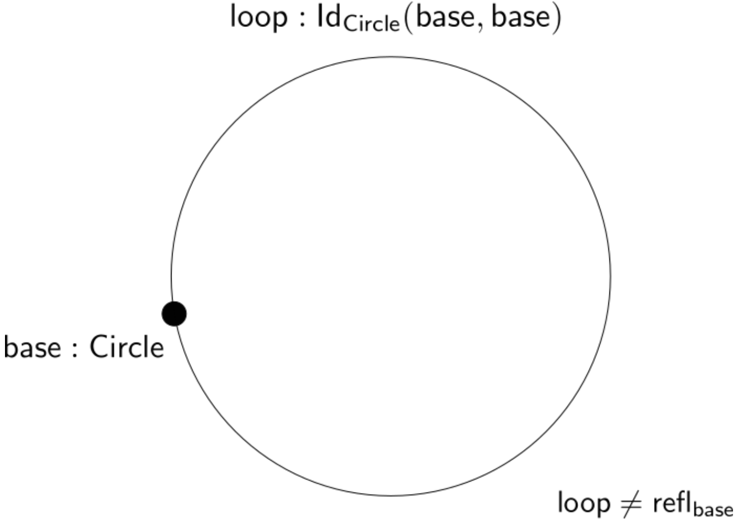
Higher Inductive Types

- Encode topological spaces up-to paths using *higher inductive types* (HITs)
- *Higher*: have an induction principle for paths as well

Homotopy Type Theory

- HoTT = UTT + HITs
- Allows us to do homotopy theory!

The Circle



$\mathsf{loop} : \mathsf{Id}_{\mathsf{Circle}}(\mathsf{base}, \mathsf{base})$
base : Circle
$\mathsf{loop} \neq \mathsf{refl}_{\mathsf{base}}$

Working with HITs

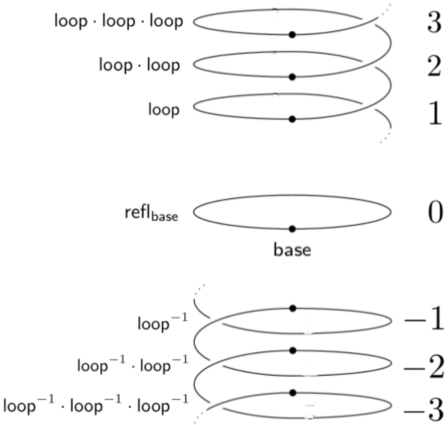
Theorem (Fundamental group of the circle)

$\pi_1(Circle, base) =$ *additive group of* $\mathbb{Z}$.

- Hard to prove in point-set homotopy theory, but pretty straightforward in HoTT!

Theorem (Fundamental group of the circle)

$\pi_1(\mathit{Circle}, \mathit{base}) =$ *additive group of* $\mathbb{Z}$.



Working with HITs

Theorem (Fundamental group of the circle)

$\pi_1(\mathit{Circle}, \mathit{base}) = \mathit{additive\ group\ of\ } \mathbb{Z}.$

Proof.

By univalence, we need $\pi_1(\mathsf{Circle}, \mathsf{base}) \rightleftarrows^{f}_{g} \mathbb{Z}$ that are mutually inverse.
$g(n)$ is easy.

$$g(n) \triangleq \begin{cases} \overbrace{\mathsf{loop} \cdot \ldots \cdot \mathsf{loop}}^{n} & n > 0 \\ \mathsf{refl}_{\mathsf{base}} & n = 0 \\ \underbrace{\mathsf{loop}^{-1} \cdot \ldots \cdot \mathsf{loop}^{-1}}_{n} & n < 0 \end{cases}$$

$\square$

Working with HITs

Theorem (Fundamental group of the circle)

$\pi_1(\mathit{Circle}, \mathit{base}) = $ *additive group of* $\mathbb{Z}$.

Proof.

$f(p)$ is hard, we need to compute the **winding number** of p. By circle induction, let Cover : Circle $\to U$ such that Cover(base) $\triangleq \mathbb{Z}$ and:

$$\begin{aligned}
\mathrm{transport}^{\mathrm{Cover}}(\mathrm{loop}, x) &\triangleq x + 1 \\
\mathrm{transport}^{\mathrm{Cover}}(\mathrm{refl}_{\mathrm{base}}, x) &\triangleq x \\
\mathrm{transport}^{\mathrm{Cover}}(\mathrm{loop}^{-1}, x) &\triangleq x - 1
\end{aligned}$$

Let $f(p) \triangleq \mathrm{transport}^{\mathrm{Cover}}(p, 0)$. □

Working with HITs

Theorem (Fundamental group of the circle)

$\pi_1(\mathit{Circle}, \mathit{base}) = \mathit{additive\ group\ of}\ \mathbb{Z}$.

Proof.

transport respects group structure, so:

$$f(\ldots \cdot \mathsf{loop} \cdot \mathsf{refl}_{\mathsf{base}} \cdot \mathsf{loop}^{-1} \cdot \ldots) = \ldots + 1 + 0 - 1 + \ldots$$

Clearly, f and g are mutually inverse. □

Conclusion

- HoTT rethinks type theory in a novel way
- Benefits for programming & mathematics

The Future^{TM}