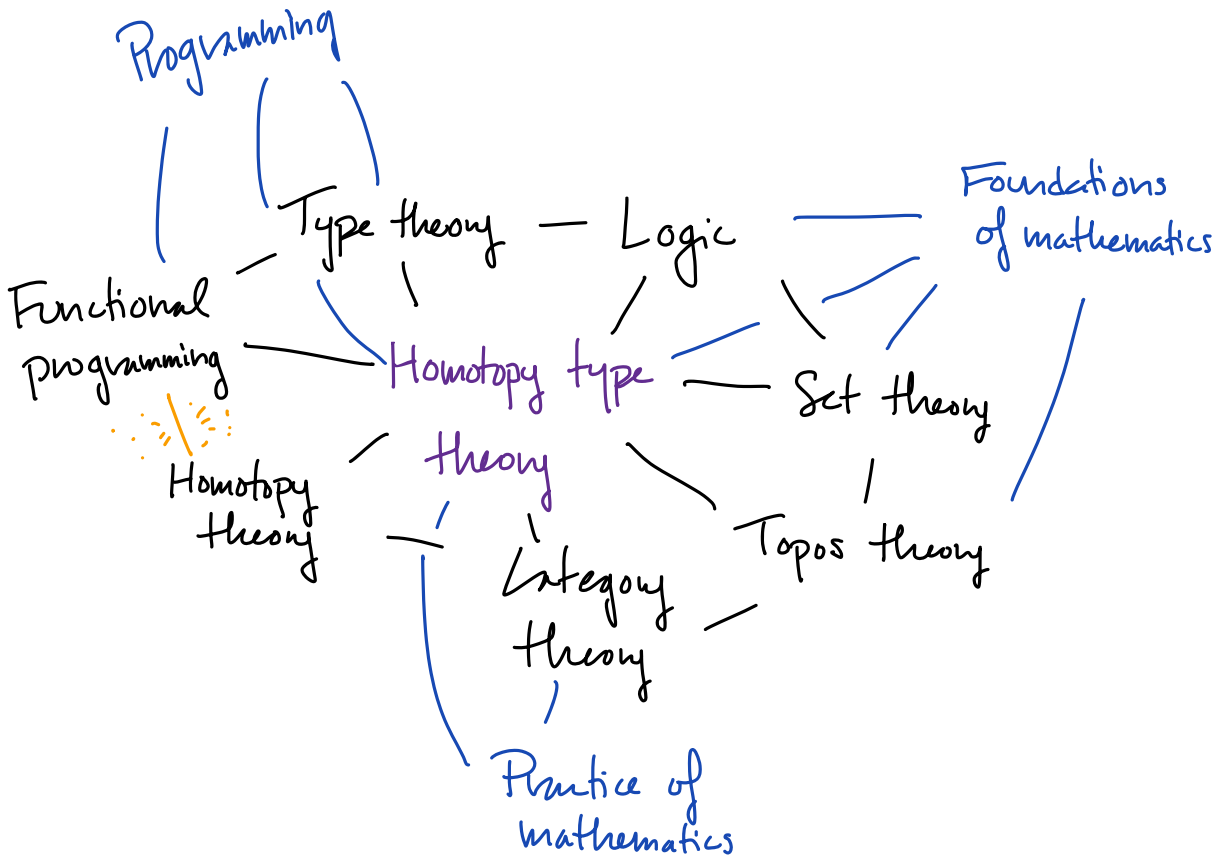
Map of subjects surrounding homotopy type theory:



Martin-Löf type theory

(Similar to CIC, but no universe Prop of propositions and with Σ-types)

- 3 basic judgments
  - Γ ctx
  - $\Gamma \vdash T$ TYPE
  - $\Gamma \vdash t : T$
- Corresponding equality judgments
  - $\Gamma \doteq \Gamma'$ ctx
  - $\Gamma \vdash T \doteq T'$ TYPE
  - $\Gamma \vdash t \doteq t' : T$
- Type formers
  - $\emptyset$, $\mathbb{1}$, $\mathbb{B}$, $\mathbb{N}$, $\mathbb{W}$-types
  - $=$ types
  - Σ types, Π types

- universe hierarchy U_i

Univalent foundations / type theory

MLTT + univalence axiom (focus on h-levels)

Homotopy type theory

UF + higher inductive types

Schedule

Lecture 1: Martin-Löf type theory

Lecture 2: Univalent type theory

Lecture 3: H-levels, logic and sets

Lecture 4: Category theory

Lecture 5: Higher inductive types

Inductive types

Inductive types are freely generated by their canonical terms.

Ex. The booleans are freely generated by their canonical terms true and false.

In Coq:

```
Inductive Bool : Type :=
  | true
  | false.
```

```
Inductive _ : _ :=
  | _
  | _.
```

tells Coq we're defining inductive type

To define a (dependent) function out of an inductive type it suffices to define it on its canonical elements.

In pen-and-paper HoTT, we specify the behavior of inductive types by hand.

The booleans: bool

bool-form:

$$\frac{}{\vdash \mathsf{bool}\ \mathsf{type}}$$

bool : Type

bool-intro:

$$\frac{}{\vdash \mathsf{true} : \mathsf{bool}} \qquad \frac{}{\vdash \mathsf{false} : \mathsf{bool}}$$

bool-elim:

$$\frac{\Gamma,\ x:\mathsf{bool} \vdash D(x)\ \mathsf{type} \qquad \Gamma \vdash a : D(\mathsf{true}) \qquad \Gamma \vdash b : D(\mathsf{false})}{\Gamma, x:\mathsf{bool} \vdash \mathsf{ind}_{\mathsf{bool}}(a,b,x) : D(x)}$$

D (x:bool) : Type

a : D true

b : D false

$\mathsf{ind}_{\mathsf{bool}}(a,b)\ (x:\mathsf{bool}) : Dx$

bool-comp:

$$\frac{\Gamma,\ x:\mathsf{bool} \vdash D(x)\ \mathsf{type} \qquad \Gamma \vdash a : D(\mathsf{true}) \qquad \Gamma \vdash b : D(\mathsf{false})}{\Gamma, x:\mathsf{bool} \vdash \mathsf{ind}_{\mathsf{bool}}(a,b,\mathsf{true}) \doteq a : D(\mathsf{true})}$$

$$\Gamma, x:\mathsf{bool} \vdash \mathsf{ind}_{\mathsf{bool}}(a,b,\mathsf{false}) \doteq b : D(\mathsf{false})$$

Exercise: Define a function not : bool → bool.

(→ works as usual:

- To produce a function bool → bool: produce x:bool ⊢ ? :bool and λ-abstract to get ⊢ λx.? : bool → bool.
- Given a function f:bool → bool and x:bool, get fx : bool.)

The natural numbers $\mathbb{N}$

$\mathbb{N}$-form:

$$\frac{}{\vdash \mathbb{N}\ \text{type}}$$

$\mathbb{N}$-intro:

$$\frac{}{\vdash 0 : \mathbb{N}} \qquad \frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash sn : \mathbb{N}}$$

$\mathbb{N}$-elim:

$$\frac{\Gamma, x:\mathbb{N} \vdash D(x)\ \text{type} \quad \Gamma \vdash a : D(0) \quad \Gamma, x:\mathbb{N}, y:D(x) \vdash b : D(sx)}{\Gamma, x:\mathbb{N} \vdash \mathrm{ind}_{\mathbb{N}}(a,b,x) : D(x)}$$

$\mathbb{N}$-comp:

$$\frac{\Gamma, x:\mathbb{N} \vdash D(x)\ \text{type} \quad \Gamma \vdash a : D(0) \quad \Gamma, x:\mathbb{N}, y:D(x) \vdash b : D(sx)}{\Gamma \vdash \mathrm{ind}_{\mathbb{N}}(a,b,0) \doteq a : D(0)} \qquad \Gamma, x:\mathbb{N} \vdash \mathrm{ind}_{\mathbb{N}}(a,b,sx) \doteq b\left[{}^{\mathrm{ind}_{\mathbb{N}}(a,b,x)}/_{y}\right] : D(sx)$$

Exercises: 1. Define $2 : \mathrm{bool} \to \mathbb{N}$

2. Define $\mathrm{add} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$

3. Define $\mathrm{mult} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$.

Σ-types

Given $b:B \vdash E(b)$ type (in log $E\ (b:B) : \mathcal{U}$), want to form a type whose terms are dependent pairs $\langle b, e \rangle$ where $b:B$, $e:E(b)$.

Dependent pair types Σ

Σ-form:
$$\frac{\Gamma, x:P \vdash Q(x)}{\Gamma \vdash \sum_{x:P} Q(x)}$$

Σ-intro:
$$\frac{\Gamma \vdash p:P \quad \Gamma \vdash q:Q(p)}{\Gamma \vdash \text{pair}(p,q) : \sum_{x:P} Q(p)}$$

Σ-elim:
$$\frac{\begin{array}{l}\Gamma, z:\sum_{x:P} Q(p) \vdash D(z) \\ \Gamma, x:P, y:Q(x) \vdash a:D(\text{pair}(x,y))\end{array}}{\Gamma, z:\sum_{x:P} Q(p) \vdash \text{ind}_\Sigma(a,z) : D(z)}$$

Σ-comp:
$$\frac{\begin{array}{l}\Gamma, z:\sum_{x:P} Q(p) \vdash D(z) \\ \Gamma, x:P, y:Q(x) \vdash a:D(\text{pair}(x,y))\end{array}}{\Gamma, x:P, y:Q(x) \vdash \text{ind}_\Sigma(a, \text{pair}(x,y)) \doteq a:D(\text{pair}(x,y))}$$

Exercise. Construct a function $\pi_1 : \sum_{x:P} Q(x) \to P$

Types as logic, sets, programs (Curry-Howard, Brouwer-Heyting-Kolmogorov)

	Logic	Sets	Program
Γ ctx	hypotheses	indexing set	names in scope
$\Gamma \vdash T$ type	predicate T on Γ	family T of sets on Γ ($T \downarrow \Gamma$)	program specification using values from Γ
$\Gamma \vdash t:T$	proof of T	section, i.e., $T(\gamma)$ for all $\gamma \in \Gamma$	program
$\mathbb{N}$	—	$\mathbb{N}$	program w/ no input that outputs a nn

$S+T$ $\left(\sum_{i:\text{bool}} T_i\right)$	$\vee$	$\sqcup$	$\checkmark$
$S \times T$ $\left(\sum_{s:S} T\right)$	$\wedge$	$\times$	$\wedge$
$S \to T$ $\left(\prod_{s:S} T\right)$	$\Rightarrow$	$\to$	turn one kind of program into another
$\sum_{b:B} E(b)$	$\exists$	$\bigsqcup$	Σ
$\prod_{b:B} E(b)$	$\forall$	$\prod$ (set of sections)	$\prod$

The strangest inductive type : Id

Why do we need the identity type?

(If we're not interested in homotopy.)

We already have a notion of equality:

judgmental equality $\doteq$

(The identity type is called propositional equality =:)

Logical interpretation: propositions are types / proofs are terms.

To prove an equality (and be consistent with the logical interpretation) we want to produce a term of a type of equalities.

Why do we need the identity type actually?

We can prove many judgmental equalities using computation rules...

Ex. $\mathrm{add}(x, 0) \doteq x$
$\mathrm{add}(x, \mathrm{s}y) \doteq \mathrm{s}\,\mathrm{add}(x, y)$

... but not all the equalities we want!

Ex. One cannot prove $\mathrm{add}(0, x) \doteq x$.

To prove this, we need to induct on n (i.e. use $\mathbb{N}$-elimination), but this only allows us to construct a term of a type.

We will be able to prove $\mathrm{add}(0, x) = x$.

Type constructors often internalize structure

- bool
- $\mathbb{N}$
- $\emptyset$
- $\mathbb{1}$

} can also be seen as internalizing external versions.

- The universe type internalizes the judgment of the form

  A type

- We'll see how the identity type internalizes judgmental equality...

Identity type $=$

$=$-form

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma \vdash a : A \quad \Gamma \vdash b : A}{\Gamma \vdash a =_A b \text{ type}}$$

$=$-intro

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash r_a : a =_A a}$$

$=$-elim

$$\frac{\Gamma, x:A, y:A, z: x =_A y \vdash D(x,y,z) \text{ type} \qquad \Gamma, x:A \vdash d : D(x,x,r_x)}{\Gamma, x:A, y:A, z: x =_A y \vdash \text{ind}_=(d,x,y,z) : D(x,y,z) \text{ type}}$$

$=$-comp

$$\frac{\Gamma, x:A, y:A, z: x =_A y \vdash D(x,y,z) \text{ type} \qquad \Gamma, x:A \vdash d : D(x,x,r_x)}{\Gamma, x:A \vdash \text{ind}_=(d,x,x,r_x) \doteq d : D(x,x,r_x)}$$

Type constructors often internalize structure

- At a 'meta' level, we can talk about judgmental equality:

  Ex. $a = b : A$

We can discuss this at the 'type-and-term' level by using identity types:

Ex. $r_a : a =_A b$

Note that the rules governing equality say that

if $a \doteq b : A$, then $(a =_A a) \doteq (a =_A b)$, and
if $r_a : a =_A a$ and $(a =_A a) \doteq (a =_A b)$, then $r_a : a =_A b$.

→ Reflexivity (r_-) turns judgmental equalities into propositional equalities.

Exercises
- (functoriality) We have a function
$$ap_f : a =_A a' \to fa =_B fa'$$
for all types A, B, functions $f : A \to B$, terms $a, a' : A$.
- $add(0, n) = n$ for all $n : \mathbb{N}$.
(Note: one of $add(0,n) \doteq n$ and $add(n,0) \doteq n$ will hold definitionally depending on how you defined add. Show the other one holds judgmentally.)
- (transport) We have a function
$$p_* : Da \to Db$$
for any type T, dependent type $t : T \vdash D(t)$, $a, b : T$, $p : a =_T b$.
- Show that $=$ is an equivalence relation, i.e. for all types A, terms $a, b, c : A$, we have terms in
$$a =_A a$$
$$a =_A b \to b =_A a$$
$$a =_A b \times b =_A c \to a =_A c$$