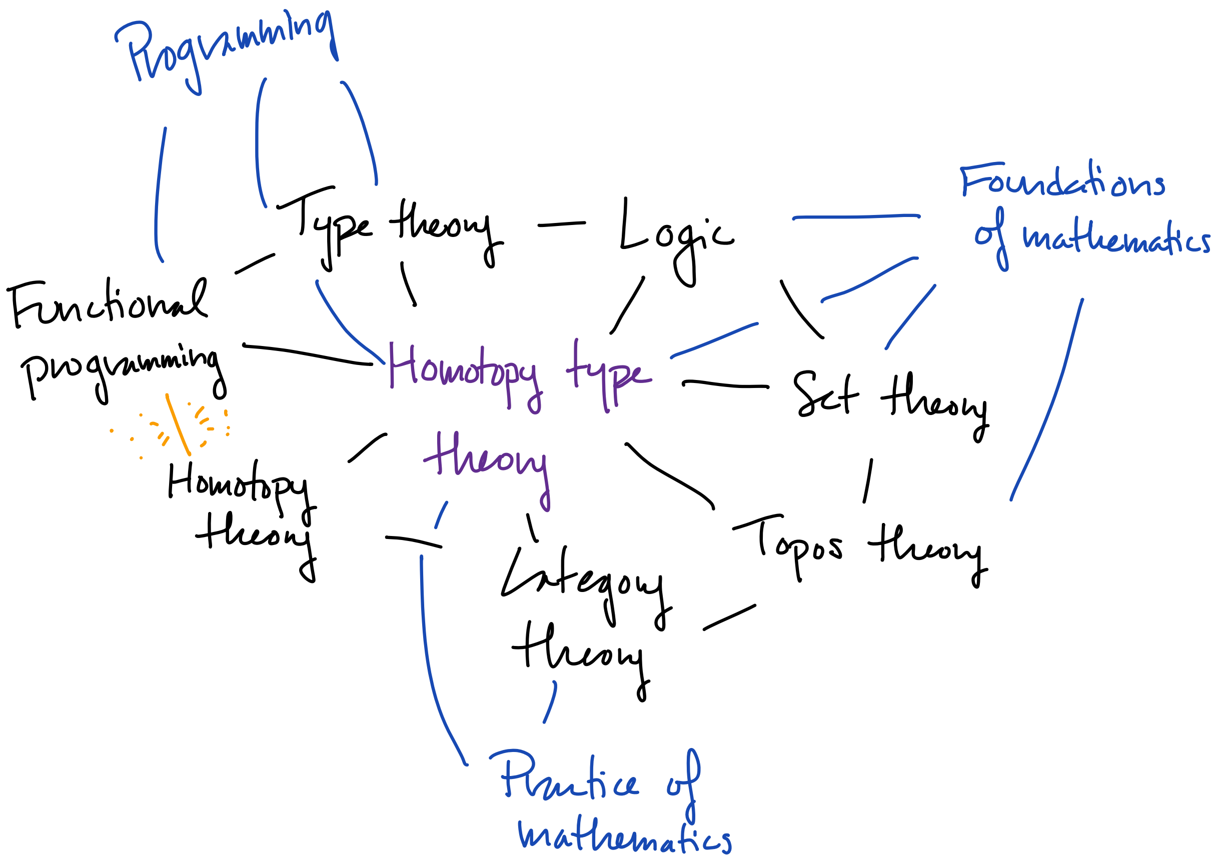
Map of subjects surrounding homotopy type theory:



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
  - $\emptyset, \mathbb{1}, \mathbb{B}, \mathbb{N}$, 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:

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

bool : Type

bool-intro:

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

bool-elim:

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

$$\frac{D\ (x:\mathrm{bool}) : \mathrm{Type} \qquad a : D\ \mathrm{true} \qquad b : D\ \mathrm{false}}{\mathrm{ind}_{\mathrm{bool}}(a,b)\ (x:\mathrm{bool}) : D x}$$

bool-comp:

$$\frac{\Gamma, x:\mathrm{bool} \vdash D(x)\ \mathrm{type} \qquad \Gamma \vdash a : D(\mathrm{true}) \qquad \Gamma \vdash b : D(\mathrm{false})}{\begin{array}{l}\Gamma, x:\mathrm{bool} \vdash \mathrm{ind}_{\mathrm{bool}}(a,b,\mathrm{true}) \doteq a : D(\mathrm{true}) \\ \Gamma, x:\mathrm{bool} \vdash \mathrm{ind}_{\mathrm{bool}}(a,b,\mathrm{false}) \doteq b : D(\mathrm{false})\end{array}}$$

Exercise: Define a function $\mathrm{not} : \mathrm{bool} \to \mathrm{bool}$.

($\to$ works as usual:

- To produce a function $\mathrm{bool} \to \mathrm{bool}$: produce $x:\mathrm{bool} \vdash ? : \mathrm{bool}$ and λ-abstract to get $\vdash \lambda x.? : \mathrm{bool} \to \mathrm{bool}$.
- Given a function $f : \mathrm{bool} \to \mathrm{bool}$ and $x : \mathrm{bool}$, get $f x : \mathrm{bool}$.)

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

$\mathbb{N}$-comp:
$$\frac{\begin{array}{c}\Gamma, x : \mathbb{N} \vdash D(x) \text{ type} \\ \Gamma \vdash a : D(0) \\ \Gamma, x : \mathbb{N}, y : D(x) \vdash b : D(sx)\end{array}}{\begin{array}{c}\Gamma \vdash \mathrm{ind}_{\mathbb{N}}(a, b, 0) \doteq a : D(0) \\ \Gamma, x : \mathbb{N} \vdash \mathrm{ind}_{\mathbb{N}}(a, b, sx) \doteq b[\mathrm{ind}_{\mathbb{N}}(a,b,x)/y] : D(sx)\end{array}}$$

Exercises:
1. Define $z : \mathrm{bool} \to \mathbb{N}$
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 Lag $E (b:B) : UU$), 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:
$$\frac{\Gamma \vdash p:P \qquad \Gamma \vdash q : Q(p)}{\Gamma \vdash \mathrm{pair}(p,q) : \sum_{x:P} Q(p)}$$

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

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

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; $\begin{array}{c}T\\\downarrow\\\Gamma\end{array}$	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

$S+T$ $(\sum_{i:\mathrm{bool}} T_i)$	$\vee$	$\sqcup$	$\checkmark$
$S\times T$ $(\sum_{s:S} T)$	$\wedge$	$\times$	$\wedge$
$S\to T$ $(\prod_{s:S} T)$	$\Rightarrow$	$\to$	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)	Π

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

... 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

$\left.\begin{array}{l} \cdot\ \mathrm{bool} \\ \cdot\ \mathbb{N} \\ \cdot\ \emptyset \\ \cdot\ \mathbb{1} \end{array}\right\}$ 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
$$\frac{\Gamma, x:A, y:A, z: x =_A y \vdash D(x,y,z) \text{ type} \qquad \Gamma, x:A \vdash d : D(x,x,r_x)}{\Gamma, x:A, y:A, z: x =_A y \vdash \mathrm{ind}_=(d,x,y,z) : D(x,y,z) \text{ type}}$$

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

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
- $\mathrm{add}(0, n) = n$ for all $n : \mathbb{N}$.
- (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 $a, a' : A$.
- Show that $=$ is an equivalence relation, i.e. for all types A, terms $a, b, c : A$, we have terms in
$$a =_A a$$
$$a =_A b \to b =_A a$$
$$a =_A b \times b =_A c \to a =_A c$$