MATH 721: HOMOTOPY TYPE THEORY

EMILY RIEHL

Contents

Part 1. Martin-Löf's Dependent Type Theory 2
August 30: Dependent Type Theory 2
September 1: Dependent function types & the natural numbers 4
September 8: The formal proof assistant `agda` 7
September 13: Inductive types 7
September 15: Identity types 7
September 20: Universes 7
September 22: Modular arithmetic 7
September 27: Elementary number theory 7

Part 2. The Univalent Foundations of Mathematics 7
September 29: Equivalences 7
October 4: Contractibility 7
October 6: The fundamental theorem of identity types 7
October 11: Propositions, sets, and general truncation levels 7
October 13: Function extensionality 7
October 18: Propositional truncation 7
October 20: The image of a map 7
October 25: Finite types 7
October 27: The univalence axiom 7
November 1: Set quotients 7
November 3: Groups 7
November 8: Algebra 7
November 10: The real numbers 7

Part 3. Synthetic Homotopy Theory 7
November 15: The circle 7
November 17: The universal cover of the circle 7
November 29: Homotopy groups of types 7
December 1: Classifying types of groups 7
December 6: TBD 7

Index cards: name, what I should call you, pronouns, year, relevant previous coursework, why you are here, something fun

Personal history: Prehistory of homotopy type theory dates to a 1998 paper of Martin Hofmann and Thomas Streicher called "The groupoid interpretation of type theory," that I'll tell you about later. Key early developments were in the mid aughts with the major players coming together for the first time in 2010 at Carnegie Mellon. This expanded to a larger group at Oberwolfach in 2011 followed by a special year at IAS in 2012-2013, during which the HoTT book (not our book) was written.

Date: Fall 2021.

I got my PhD in 2011. I started hearing a little about this in my final years of grad school but learned most of what I'll tell you about this semester during my postdoc and while here at Johns Hopkins. Point of this to say is that learning doesn't stop when you earn your PhD. Learning also doesn't necessarily precede teaching. My goal for this semester is to know a lot more `agda` by the end than I do right now. My promise to you is that I'll do all the homework before I assign it. I don't promise that I'll be able to answer very many of your questions about what is going on under the hood, but that's okay too: it's probably useful for all of us to be reminded that figures of authority don't always know what they're talking about.

Syllabus: The goal for this course is to change the way you think about doing mathematics. I certainly have. Along the way, I hope you all learn a lot, though as is typical in a graduate course, that's somewhat in proportion to the amount of work you put in. This also has a lot to do with your background. If you're coming from CS, you'll probably leave the course knowing way more `agda` than I do, but perhaps a bit less about synthetic homotopy theory. For others, it will be vice versa. Still others might engage most with the philosophical implications of these developments. I'm equally happy with all of these directions.

There are two "active learning" activities. The first is writing your own formal proofs in the computer proof assistant `agda`. This is hard in the sense that if you make one typo the whole thing breaks but a lot of help is available. I'll say more about it when the time comes. In particular, I'm hoping many folks will attend group office hours on Thursday evenings from 5-6pm. In the first meeting there won't be a problem set due: instead the goal will be to get `agda` installed. Come see me if you're worried about technical issues.

The second active learning activity is a final project. I'm very flexible about the parameters. Essentially you should pick something that sounds fun to you.

Questions?

Part 1. **Martin-Löf's Dependent Type Theory**

August 30: Dependent Type Theory

Martin-Löf's dependent type theory is a formal language for writing mathematics: both constructions of mathematical objects and proofs of mathematical propositions. As we shall discover, these two things are treated in parallel (in contrast to classical Set theory plus first-order logic, where the latter supplies the proof calculus and the former gives the language which you use to state things to prove).

Judgments and contexts. I find it helpful to imagine I'm teaching a computer to do mathematics. It's also helpful to forget that you know other ways of doing mathematics.[1]

defn. There are four kinds of **judgments** in dependent type theory, which you can think of as the "grammatically correct" expressions:

(i) $\Gamma \vdash A\ \texttt{type}$, meaning that A is a well-formed type in **context** Γ (more about this soon).
(ii) $\Gamma \vdash a : A$, meaning that a is a well-formed term of type A in context Γ.
(iii) $\Gamma \vdash A \doteq B\ \texttt{type}$, meaning that A and B are **judgmentally** or **definitionally** equal types in context Γ.
(iv) $\Gamma \vdash a \doteq b : A$, meaning that a and b are judgmentally equal terms of type A in context Γ.

These might be collectively abbreviated by $\Gamma \vdash \mathcal{J}$.

The statement of a mathematical theorem, often begins with an expression like "Let n and m be positive integers, with $n < m$, and let $\vec{v}_1, \dots, \vec{v}_m$ be vectors in $\mathbb{R}^n$. Then ..." This statement of the hypotheses defines a **context**, a finite list of types and hypothetical terms (called **variables**[2]) satisfying an inductive condition that that each type can be derived in the context of the previous types and terms using the inference rules of type theory.

defn. A **context** is a finite list of variable declarations:

$$x : A_1, x_2 : A_2(x_1), \dots, x_n : A_n(x_1, \dots, x_{n-1})$$

satisfying the condition that for each $1 \leq k \leq n$ we can derive the judgment

$$x_1 : A_1, \dots, x_{k-1} : A_{k-1}(x_1, \dots, x_{k-2}) \vdash A_k(x_1, \dots, x_{k-1})\ \texttt{type}$$

[1]Indeed, there are very deep theorems that describe how to interpret dependent type theory into classical set-based mathematics. You're welcome to investigate these for your final project but they are beyond the scope of this course.

[2]We're not going to say anything about proper syntax for variables and instead rely on instinct to recognize proper and improper usage.

using the inference rules of type theory.

We'll introduce the inference rules shortly but the idea is that it needs to be possible to form the type $A_k(x_1, \dots, x_{k-1})$ given terms $x_1, \dots, x_{k-1}$ of the previously-formed types.

ex. For example, there is a unique context of length zero: the empty context.

ex. $n : \mathbb{N}, m : \mathbb{N}, p : n < m, \vec{v} : (\mathbb{R}^n)^m$ is a context. Here $n : \mathbb{N}, m : \mathbb{N} \vdash n < m$ is a dependent type that corresponds to the relation $\{n < m \mid n, m \in \mathbb{N}\} \subset \mathbb{N} \times \mathbb{N}$ and the variable p is a witness that $n < m$ is true (more about this later).

Type families. Absolutely everything in dependent type theory is context dependent so we always assume we're working in a background context Γ. Let's focus on the primary two judgment forms.

defn. Given a type A in context Γ a **family** of types over A in context Γ is a type $B(x)$ is context $\Gamma, x : A$, as represented by the judgment:

$$\Gamma, x : A \vdash B(x)\ \mathsf{type}$$

We also say that $B(x)$ is a type **indexed** by $x : A$, in context Γ.

ex. $\mathbb{R}^n$ is a type indexed by $n \in \mathbb{N}$.

defn. Consider a type family B over A in context Γ. A **section** of the family B over A in context Γ is a term of type $B(x)$ in context $\Gamma, x : A$, as represented by the judgment:

$$\Gamma, x : A \vdash b(x) : B(x)$$

We say that b is a **section** of the family B over A in context Γ or that $b(x)$ is a term of type $B(x)$ indexed by $x : A$ in context Γ.

ex. $\vec{0}_n : \mathbb{R}^n$ is a term dependent on $n \in \mathbb{N}$.

Exercise. If you've heard the word "section" before you should think about what it is being used here.

Inference rules. There are five types of inference rules that collectively describe the structural rules of dependent type theory. They are

(i) Rules postulating that judgmental equality is an equivalence relation:

$$\frac{\Gamma \vdash A\ \mathsf{type}}{\Gamma \vdash A \doteq A\ \mathsf{type}} \qquad \frac{\Gamma \vdash A \doteq B\ \mathsf{type}}{\Gamma \vdash B \doteq A\ \mathsf{type}} \qquad \frac{\Gamma \vdash A \doteq B\ \mathsf{type} \qquad \Gamma \vdash B \doteq C\ \mathsf{type}}{\Gamma \vdash A \doteq C\ \mathsf{type}}$$

and similarly for judgmental equality between terms.

(ii) Variable conversion rules for judgmental equality between types:

$$\frac{\Gamma \vdash A \doteq A'\ \mathsf{type} \qquad \Gamma, x : A, \Delta \vdash \mathcal{J}}{\Gamma, x : A', \Delta \vdash \mathcal{J}}$$

(iii) Substitution rules:

$$\frac{\Gamma \vdash a : A \qquad \Gamma, x : A, \Delta \vdash \mathcal{J}}{\Gamma, \Delta[a/x] \vdash \mathcal{J}[a/x]}$$

If Δ is the context $y_1 : B_1(x), \dots, y_n : B_n(x, y_1, \dots, y_{n-1})$ then $\Delta[a/x]$ is the context $y_1 : B(a), \dots, y_n : B_n(a, y_1, \dots, y_{n-1})$. A similar substitution is performed in the judgment $\mathcal{J}[a/x]$. Further rules indicate that substitution by judgmentally equal terms gives judgmentally equal results.

(iv) Weakening rules:

$$\frac{\Gamma \vdash A\ \mathsf{type} \qquad \Gamma, \Delta \vdash \mathcal{J}}{\Gamma, x : A, \Delta \vdash \mathcal{J}}$$

Eg if A and B are types in context Γ, then B is also a type in context $\Gamma, x : A$.

(v) The generic term:

$$\frac{\Gamma \vdash A\ \mathsf{type}}{\Gamma, x : A \vdash x : A}$$

This will be used to define the identity function of any type.

Derivations. A derivation in type theory is a finite rooted tree where each node is a valid rule of inference. The root is the conclusion.

ex. The interchange rule is derived as follows

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash B\ \mathtt{type}}{\Gamma, y : B \vdash y : B}}{\Gamma, y : B, x : A \vdash y : B} \qquad \dfrac{\Gamma \vdash B\ \mathtt{type} \qquad \dfrac{\Gamma, x : A, y : B, \Delta \vdash \mathcal{J}}{\Gamma, x : A, z : B, \Delta[z/y] \vdash \mathcal{J}[z/y]}}{\Gamma, y : B, x : A, z : B, \Delta[z/y] \vdash \mathcal{J}[z/y]}}{\Gamma, y : B, x : A, \Delta \vdash \mathcal{J}}$$

September 1: Dependent function types & the natural numbers

The rules for dependent function types. Consider a section b of a family B over A in context Γ, as encoded by a judgment:

$$\Gamma, x : A \vdash b(x) : B(x).$$

We think of the section b as a function that takes as input $x : A$ and produces a term $b(x) : B(x)$. Since the type of the output is allowed to depend on the term being input, this isn't quite an ordinary function but a **dependent function**. The type of all dependent functions is the **dependent function type**

$$\prod_{x:A} B(x)$$

What is a thing in mathematics? Structuralism says the ontology of a thing is determined by its behavior. In dependent type theory, we define dependent function types by stating their rules, which have the following forms:

(i) **formation rules** tell us how a type may be formed
(ii) **introduction rules** tells us how to introduce new terms of the type
(iii) **elimination rules** tell us how the terms of a type may be used
(iv) **computation rules** tell us how the introduction and elimination rules interact

There are also **congruence rules** that tell us that all constructions respect judgmental equality. See your book for more details.

defn (dependent function types). The Π-**formation rule** has the form:

$$\frac{\Gamma, x : A \vdash B(x)\ \mathtt{type}}{\Gamma \vdash \prod_{x:A} B(x)\ \mathtt{type}}$$

The Π-**introduction rule** has the form:

$$\frac{\Gamma, x : A \vdash b(x) : B(x)}{\Gamma \vdash \lambda x.b(x) : \prod_{x:A} B(x)}$$

The λ-**abstraction** $\lambda x.b(x)$ can be thought of as notation for $x \mapsto b(x)$.

The Π-**elimination rule** has the form of an evaluation rule:

$$\frac{\Gamma \vdash f : \prod_{x:A} B(x)}{\Gamma, x : A \vdash f(x) : B(x)}$$

Finally, there are two computation rules: the β-**rule**

$$\frac{\Gamma, x : A \vdash b(x) : B(x)}{\Gamma, x : A \vdash (\lambda y.b(y))(x) \doteq b(x) : B(x)}$$

and the η-**rule**, which says that all elements of a Π-type are dependent functions:

$$\frac{\Gamma \vdash f : \prod_{x:A} B(x)}{\Gamma \vdash \lambda x.f(x) \doteq f : \prod_{x:A} B(x)}$$

Ordinary function types.

defn (function types). The formation rule is derived from the formation rule for Π-types together with weakening:

$$\cfrac{\cfrac{\Gamma \vdash A \ \texttt{type} \qquad \Gamma \vdash B \ \texttt{type}}{\Gamma, x : A \vdash B \ \texttt{type}}}{\Gamma \vdash \prod_{x:A} B \ \texttt{type}}$$

We adopt the notation

$$A \to B := \prod_{x:A} B$$

for the dependent function type in the case where the type family B is constant over $x : A$.

The introduction, evaluation, and computation rules are instances of term conversion: eg

$$\frac{\Gamma \vdash B \ \texttt{type} \qquad \Gamma, x : A \vdash b(x) : B}{\Gamma \vdash \lambda x.b(x) : A \to B} \qquad \frac{\Gamma \vdash f : A \to B}{\Gamma, x : A \vdash f(x) : B}$$

plus the two computation rules:

$$\frac{\Gamma \vdash B \ \texttt{type} \qquad \Gamma, x : A \vdash b(x) : B}{\Gamma, x : A \vdash (\lambda y.b(y))(x) \doteq b(x) : B} \qquad \frac{\Gamma \vdash f : A \to B}{\Gamma \vdash \lambda x.f(x) \doteq f : A \to B}$$

defn. Identity functions are defined as follows:

$$\cfrac{\cfrac{\Gamma \vdash A \ \texttt{type}}{\Gamma, x : A \vdash x : A}}{\Gamma \vdash \lambda x.x : A \to A}$$

which is traditionally denoted by $\mathrm{id}_A := \lambda x.x$.

The idea of composition is that given a function $f \colon A \to B$ and $g \colon B \to C$ you should get a function $g \circ f \colon A \to C$. Using infix notation you might denote this function by $\_ \circ \_$.

Q. $\_ \circ \_$ is itself a function, so it's a term of some type. What type?[3]

defn. Composition has the form:

$$\frac{\Gamma \vdash A \ \texttt{type} \qquad \Gamma \vdash B \ \texttt{type} \qquad \Gamma \vdash C \ \texttt{type}}{\Gamma \vdash \_ \circ \_ : (B \to C) \to ((A \to B) \to (A \to C))}$$

It is defined by

$$\_ \circ \_ := \lambda g.\lambda f.\lambda x.g(f(x))$$

which can be understood as the term constructed by three applications of the Π-introduction rule followed by two applications of the Π-elimination rule.

Composition is associative essentially because both $(h \circ g) \circ f$ and $h \circ (g \circ f)$ are defined by $\lambda x.h(g(f(x)))$. We'll think about this more formally when we come back to identity types.

Similarly, you can compute that for all $f : A \to B$, $\mathrm{id}_B \circ f \doteq f : A \to B$ and $f \circ \mathrm{id}_A \doteq f : A \to B$.

The type of natural numbers. The type $\mathbb{N}$ of natural numbers is the archetypical example of an **inductive type** about more which soon. It is given by rules which say that it has a term $0_{\mathbb{N}} : \mathbb{N}$, it has a successor function $\texttt{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$ and it satisfies the induction principle.

The $\mathbb{N}$-formation rule is

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

In other words, $\mathbb{N}$ is a type in the empty context.

There are two $\mathbb{N}$-introduction rules:

$$\frac{}{\vdash 0_{\mathbb{N}} : \mathbb{N}} \qquad \frac{}{\vdash \texttt{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}$$

[3]Really the type should involve three universe variables but let's save this for next week.

Digression (traditional induction). In traditional first-order logic, the principle of $\mathbb{N}$-induction is stated in terms of a **predicate** P over $\mathbb{N}$. One way to think about P is as a function $P\colon \mathbb{N} \to \{\top, \bot\}$. That is, for each $n \in \mathbb{N}$, $P(n)$ is either true or false. We could also think of P as an indexed family of sets $(P(n))_{n\in\mathbb{N}}$ where for each n either $P(n) = \varnothing$ (corresponding to $P(n)$ being false) or $P(n) = *$ (corresponding to $P(n)$ being true).

The induction principle then says

$$\forall P : \{0,1\}^{\mathbb{N}}, (P(0) \wedge (\forall n, P(n) \to P(n+1)) \to \forall n, P(n)).$$

In dependent type theory it is most natural to let P be an arbitrary type family over $\mathbb{N}$. This is a stronger assumption, as we'll see.

Q. What then corresponds to a proof that $\forall n, P(n)$?

The induction principle is encoded by the following rule:

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n)\ \mathtt{type} \qquad \Gamma \vdash p_0 : P(0_{\mathbb{N}}) \qquad \Gamma \vdash p_S : \prod_{n:\mathbb{N}} (P(n) \to P(\mathsf{succ}_{\mathbb{N}}(n)))}{\Gamma \vdash \mathtt{ind}_{\mathbb{N}}(p_0, p_S) : \prod_{n:\mathbb{N}} P(n)}$$

Remark. There are other forms this rule might take that are interderivable with this one.

The computation rules say that the function $\mathtt{ind}_{\mathbb{N}}(p_0, p_S) : \prod_{n:\mathbb{N}} P(n)$ behaves like it should on $0_{\mathbb{N}}$ and successors:

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n)\ \mathtt{type} \qquad \Gamma \vdash p_0 : P(0_{\mathbb{N}}) \qquad \Gamma \vdash p_S : \prod_{n:\mathbb{N}} (P(n) \to P(\mathsf{succ}_{\mathbb{N}}(n)))}{\Gamma \vdash \mathtt{ind}_{\mathbb{N}}(p_0, p_S)(0_{\mathbb{N}}) \doteq p_0 : P(0_{\mathbb{N}})}$$

and under the same premises

$$\Gamma, n : \mathbb{N} \vdash \mathtt{ind}_{\mathbb{N}}(p_0, p_S)(\mathsf{succ}_{\mathbb{N}}(n)) \doteq p_S(n, \mathtt{ind}_{\mathbb{N}}(p_0, p_S, n)) : P(\mathsf{succ}_{\mathbb{N}}(n)).$$

These computation rules don't matter so much if the type family $n : \mathbb{N} \vdash P(n)$ is really a predicate — $P(n)$ is either true or false and that's the end of the story — but they do matter if $P(n)$ is more like an indexed family of sets. In the latter case, $\mathtt{ind}_{\mathbb{N}}(p_0, p_S)$ is the recursive function defined from p_0 and p_S and these are the computation rules for that recursion.

Remark. Recall Peano's axioms for the natural numbers:

(i) $0_{\mathbb{N}} \in \mathbb{N}$
(ii) $\mathsf{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$
(iii) $\forall n, \mathsf{succ}_{\mathbb{N}}(n) \neq 0_{\mathbb{N}}$
(iv) $\forall n, m, \mathsf{succ}_{\mathbb{N}}(n) = \mathsf{succ}_{\mathbb{N}}(m) \to n = m$
(v) induction

We'll be able to *prove* this missing two axioms from the induction principle we've assumed once we have identity types and universes. We'll come back to this in a few weeks.

September 8: The formal proof assistant `agda`

September 13: Inductive types

September 15: Identity types

September 20: Universes

September 22: Modular arithmetic

September 27: Elementary number theory

Part 2. The Univalent Foundations of Mathematics

September 29: Equivalences

October 4: Contractibility

October 6: The fundamental theorem of identity types

October 11: Propositions, sets, and general truncation levels

October 13: Function extensionality

October 18: Propositional truncation

October 20: The image of a map

October 25: Finite types

October 27: The univalence axiom

November 1: Set quotients

November 3: Groups

November 8: Algebra

November 10: The real numbers

Part 3. Synthetic Homotopy Theory

November 15: The circle

November 17: The universal cover of the circle

November 29: Homotopy groups of types

December 1: Classifying types of groups

December 6: TBD

Dept. of Mathematics, Johns Hopkins University, 3400 N Charles St, Baltimore, MD 21218
E-mail address: `eriehl@math.jhu.edu`