HoTT lecture 1 Q&A (Solved)

HoTTEST Summer School 2022

The HoTTEST TAs

4th July 2022

What's the difference between formalization of mathematics and foundations of mathematics?

Formalization of mathematics is the practice of doing mathematics in a computer proof assistant (such as Agda, Coq, Lean, etc), whereas the foundations of mathematics is a field of study, and thus need not involve computers — you can do it in a computer, or you can do it on paper (in the "usual way"). A lot of times they're related, since our verifcation software is usually based in our choice of foundation, but they don't need to be related *a priori*.

What's the difference between the implication arrow and the horizontal deduction bar?

The horizontal deduction bar organizes several judgments (or judgments-in-context) into an inference rule. For instance, the rule

$$\frac{\Gamma \vdash p\colon P \quad \Gamma \vdash q\colon Q}{\Gamma \vdash (p,q)\colon P \wedge Q}$$

asserts that, if p is a term of type P in context Γ and q is a term of type Q in context Γ, then (p,q) is a term of type $P \wedge Q$ in context Γ. These judgments are statements *about* the type theory (stating the relationships between certain terms and types); a logician would call them judgments in the *metatheory.* The horizontal bar expresses a metatheoretic implication, that is, an implication between judgments *about* the type theory.

On the other hand, $\to$ is a symbol *in* the theory: if I have types P and Q, I can stick a $\to$ between them to get another type, $P \to Q$. This type may be inhabited (by a function turning proofs of P into proofs of Q), or uninhabited (e.g. if P is a true proposition but Q is a false one). We say that $\to$ is a *type former*, because it combines types to form new ones.

I am a bit confused as to what's the big picture here. Until now we have just been doing inference rules with some extra jargon thrown around.

Think of it as two ways of interpreting the inference rules: either as operations in some programming language, or as logical proofs. This will be a running theme throughout the course.

We've taken the step from natural deduction to simply typed lambda calculus as a foundation for building up more complex type systems – we are building up to eventually introduce types that can depend on terms.

What's the $=$ sign with the dot on it?

$\doteq$ is our symbol for *judgmental equality*. If A and A' are types, then the judgment $A \doteq A'$ asserts that A and A' are judgmentally equal, that is, equal by definition. Judgmentally equal types are interchangeable in type theory (see Chapter 1 of the text for a precise statement of this). We also have judgmental equality of terms, e.g. if $t : T$ and $t' : T$, then $t \doteq t' : T$ is the judgment that t and t' are judgmentally equal. Later, we will contrast judgmental equality with a different notion of equality, propositional equality.

Will we need to state rules about how the equality (with a dot) behaves? (I.e. how we can use it)

Good question! The first chapter of the book does exactly that.

Is there any significance to the fact that in $\to$ elim, p and f combine into fp in that order?

Not really. there are other function applications which flip the order, and we could have used a different notational convention if we wanted.

If we have the meta logic, and some rules that we decided upon (how?), how come we're trying to recreate logic again?

The meta logic is the ability to check whether or not proof trees are well formed (e.g. asking "did I apply that rule correctly?"), whereas, if we can agree that we can correctly apply rules, then the language itself lets us write down proofs.

If we can turn internal implication into external implication and vice versa, could we get away with just having one of them?

We want to use the external logic to study the internal one as a mathematical object, so usually we will need external implication in order to study the internal one (e.g. in order to define it).

Is "context" defined in terms of hypotheses to the left of the turnstyle?

Yes, exactly. A context is a list of variables together with their types, and appears on the left of a $\vdash$. If the context is the same for every judgment in a rule, we might omit it (and the $\vdash$).

Do we have to implicitly require that f doesn't contain f as free variable?

That's right. We always assume that every new variable we introduce is fresh. Generally, we handwave this away though (the "Barendregt convention")

Under the Lambek interpretation, should we interpret $P \to Q$ as literally a function (i.e. morphism) $P \to Q$ or as a set (ie internal hom) Q^P?

The function connective will correspond to internal homs: What corresponds to morphisms are the ***deductions***, i.e., the turnstiles! So if you have $\Gamma \vdash x \colon T$ in your logic, that means that you have some morphism (the interpretation of x) from (the interpretation of) Γ to (the interpretation of) T.

If a program doesn't halt for some input term, does it output a term? or no term? or can there be a term that means "undefined"?

Not in typed lambda calculus! You can't write any non-terminating programs; this is called a normalization theorem.

Why is η-reduction not immediate from α-equivalence?

α-equivalence says you can rename variables inside a function. η-equivalence tells you that $\lambda x.f(x)$ is the same thing as f. The latter is not provable from the former.

Is there equality of types?

Yes! Actually there's two notions: there's judgmental equality (mentioned above, and covered in Chap. 1 of the textbook), which is a judgment asserting two types are equal "by definition". Later, we'll cover ***propositional equality***, which is a way of expressing things are equal to each other ***as a type in the theory.*** How propositional equality of types works is the subject of the famous ***Univalence Axiom***, one of the central ideas of HoTT. Much more on that later!

What does "indexed" mean in "indexed families of sets"?

In classical set theory, an indexed family of sets is given by a set I – called the index set – and a set U_i for each $i \in I$. The corresponding notion in type theory is a type family

$$a\colon A \vdash B \text{ type}$$

because for each term $t : A$, there is a type $B(t)$ (also expressed $B[t/a]$).

Would $\mathsf{isEven}(3)$ be equal to $\mathsf{isEven}(5)$?

They would not be judgmentally equal types, since 3 and 5 are distinct terms of type $\mathbb{N}$. However, they are logically equivalent, which we'll have a special way of expressing in HoTT (stay tuned!).

When Paige is saying "these are the same" or "this is a special case of that", what does this precisely mean? Is this at the meta level or within dependent type theory?

For example, the type $A \to B$ is a special case of dependent functions $\prod_{a:A} B(a)$. It is the special case when the type B does not depend on the argument $a : A$. When you specialize the rules for $\prod$ to this case, they become the same, and thus they have the same behavior. So it is enough to add dependent functions to our type theory because we get usual (non-dependent) functions by having the second type not depend on the first type.

We can use the sameness of behavior to prove *equivalence*. We'll cover equivalence a lot more later, but basically means that these two things are actually just different notations for the same thing, and we can write a kind of "translation" between them *in* the language of type theory.

What's the difference between Pi here and 'forall' in, e.g., Haskell? I know Haskell's 'forall' quantification is not the same as Pi-types, but I thought that it also corresponded to the idea of universal quantification in logic.

Haskell's 'forall' is a telling us that some polymorphic function works for all types. Pi quantifies over all *terms* of a given type, not over all types. Once we introduce "universes" we'll see how to simulate Haskell's forall using a Pi type over the universe, i.e. Pi is more general than 'forall'.

Is there an analogous statement for the STLC and CCC correspondence for this dependent type theory?

Yes, it involves locally cartesian closed categories!

Does $x : P \vdash Q$ type mean that from any x in P we have a proof that Q is a type?

Close: it isn't exactly a *proof* (in the sense we used 'proof' in lecture, i.e. a term) that Q is a type, it's the *judgment* that Q is a type (assuming a variable $x : P$). In other words, it means that, if we have a term $x : P$ in context, then we're able to form a type Q, which might depend on the value of x.

Our main example was $n : \mathbb{N} \vdash \mathrm{Vect}(n)$ type: for each natural number n, there is a type $\mathrm{Vect}(n)$ of vectors (say, of natural numbers) of length n.