HoTT lecture 2 Q&A (Solved)

HoTTEST Summer School 2022

The HoTTEST TAs

4th July 2022

Q1 What tools do you use to draw on the screen like you do ?

"An iPad with Notability"

Q2 Live Captions?

"Should be enabled now!"

Q4 are type formers just the '∧ type' bits or are the intro and elim rules included too?

"Not sure what you mean by '∧ type', but types come with intro and elim rules, yes."

"they were listed in the last lecture notes as eg '∧-form'"

Q5 "Is this an exhaustive list of type formers?"

"Each type system has a choice of type formers. We are deciding to include the ones mentioned so far."

Q6 "How axiom are represented in the theory ? They are special deduction rules ?"

"Yep. The rules we assert (postulate) types much like the ZF axioms postulate certain set constructions"

Q7 "Is Π type somehow Axiom of Choice in disguise?"

"A Π-type is just a dependent function type, but you can indeed express variants of the axiom of choice in terms of Π-types."

"No. We have to define terms of Pi types, using lambda abstraction. Pi's are "constructive". AC is nonconstructive"

"I see, there is always a Π type, it may be empty ..."

"Will AC state that the Π type is nonempty?"

"When we get to propositional truncation, you'll be able to state "an arbitrary family of nonempty sets is nonempty", which indeed will be stating that a certain Π-type is nonempty."

"I did not get what you meant by an arbitrary family being nonempty. Did you mean the product?"

"Yes :) 'family' means product here"

Q8 "What does 'freely' means?"

"live answered"

"I think both "generated by" and "freely" refer to how you can define a function from the type to something else. "Generated by" means that such a function is uniquely determined by the image of the constructors. "Freely" means that those images can be picked arbitrarily in the codomain, with no further restriction.
An analogy with group presentations: to map out of a free group, you just need to give the image of each generator. To map out of a group with generators and relations (so, not a free group), you also have to check that the images you picked for the generators satisfy the relations."

Q9 "The type formers are the same as the intro and elim rules that we saw last class? So natural deduction chooses ($\to$, $\wedge$, etc..) while dependent type theory chooses (π). Am I correct in this understanding?"

"Close. So natural deduction is this style of doing type theory, where we use inference rules and judgments. In it, we've defined $\to$, $\wedge$, and Π. Now we're doing bool in a natural deduction style"

"Thanks, so nat deduction is just a way to do proofs and the intro-elim rules we choose form the theory."

Q10 "Doesn't freely mean there are no more conditions?"

"Doesn't freely mean there are no more conditions?"

"In a sense, but in higher inductive types, some of the formers may themselves be thought of as conditions on the other formers. It's a bit subtle!"

"For intuition, think of freely generated algebraic objects, like groups. For example, the type of all finite lists with elements of a given type A looks like the free monoid on A."

"groups are actually the example i had in mind!"

Q11 "Would it be possible to define a type that's not freely generated in the way that Paige just described, with some extra constraints on its behavior?"

"Yes. That's something we'll use identity types for later on"

Q12 "What is the relation between inductive types and positive types?"

"What is the relation between inductive types and positive types?"

"Positivity is a condition on how we can write down new inductive types in our system. We may want to impose such conditions (or slightly stringer ones) to avoid problems."

"positive types is different from positivity, all inductive types are positive types, see https://ncatlab.org/nlab/show/polarity+in+type+theory"

"Is there a positive type which is not an inductive type? I can't find an example..."

Q13 "Why does this make sense? Because the booleans are finite and something we understand? Same for nat?"

"Nat isn't finite of course"

"What are you referring to by 'this'?"

"This need not make sense in general (even more, there is no general statement), but the existence of corresponding models answers this question for a given inductive type."

"Thanks!"

Q14 "Is types being completely determined by its canonical types similar to how, in propositional logic/ZOL, we can prove that interpretation of WFFs is completely determined by interpretation of its atomic propositions?"

"Is types being completely determined by its canonical types similar to how, in propositional logic/ZOL, we can prove that interpretation of WFFs is completely determined by interpretation of its atomic propositions?"

"Yeah, same basic idea: there, the connectives are used to inductive generate all the WFFs, i.e. freely generate them"

Q16 "is the word "type" needed in bool-form?"

"Our convention is to write the judgment like that. Some authors omit "type" there"

Q17 "How does Agda's dependent pattern matching relate to elimination rules? (Maybe the answer to this would be too long to answer during the lecture though?)"

"When you pattern match to write a dependent function in Agda, what you're doing is essentially supplying arguments to the associated ind term for that type. Maybe we could give a fuller explanation on discord"

"Which channel would be good to ask on?"

Q18 "ind_{bool} is a Pi type, right? or is it something special in this context?"

"It can be formulated as a term of a Pi type."

Q20 "Does D(true) means substitution like D(true/x) ?"

"Yes"

Q21 "So why do we call this an elimination rule? Product and function/pi elimination consumed values of those types and gave us values of simpler types, but there are booleans above and below the line in bool-elim and the term below is more complex. Why don't we call this $ind_{bool} - intro$?"

"So why do we call this an elimination rule? Product and function/pi elimination consumed values of those types and gave us values of simpler types, but there are booleans above and below the line in bool-elim and the term below is more complex. Why don't we call this $ind_{bool} - intro$?"

"Because we can use it to write functions out of bool, as we're about to see"

"An "elimination principle" states how to make functions out of a given type, it's not about "eliminating" things above the line."

Q22 "Can ind_{bool} be defined more precisly, or is it a black box ?"

"It is not a black box. It is a specific term we're adding to the type theory that expresses induction out of Bool."

"is it "just syntax", similar to "pr1" and "pr2" projections?"

"It is just new syntax, but the projections can be derived from the induction rule, whereas we are postulating ind_{bool}."

Q23 "has the ind_{bool} term been explained before? what is the type of it?"

"ind_{bool} is the thing given in the conclusion of the bool-elim rule"

"(ind_{bool} was just explained in the example, but feel free to ask a follow-up.)"

Q24 "This way of writing constructions is very hard to write down on paper, will we keep using natural deduction this entire course?"

"We won't be writing proof trees this entire course, no. As we move on, we'll be doing proofs in type-theory in a more informal fashion."

"In the book, elimination rules are more often written as an informal paragraph/list, or as a depedent function"

Q25 "The order that arguments go into induction matter but not the order they show up in the derivation tree? This is what chat is saying. Wanted confirmation"

"If you write a derivation tree, it's clear how the previous line relates to the next line, so you could move things around. But if you start mixing up the order of the variables you give to ind that will be confusing."

Q26 "why do the intro rules not have both the types above the horizontal line. don't you need an element of both types?"

"coproduct is like union, so it's enough to have just one type i think"

"The intro rules for + express functions from each factor speartely into the coproduct (or disjoint union)."

Q27 "I think that should be inr(q)"

"live answered"

Q28 "Why must we prove both D(inl p) and D(inr q) instead of just one of them, intuitively?"

"If you think about the union of two sets A and B, then to map out of the union you need to say where both A and B go."

"Paige just said it suffices to show one of them. I'm not sure I understand"

Q29 "Should the $in_+(a, b, x)$ not take functions as its first two parameters? a and b vary with p and q. or do they do that implicitly"

"You can lambda-abstract ind_+ to define a dependent function out of P+Q (like we did with ind_{bool} earlier)."

"The dependence is explicit, not implicit, since p is in the context when defining a, and likewise for q and b."

Q30 "Is co-product type just a union type? or is there some difference between the two types?"

"You can think of it like the disjoint union of two sets."

"where is the idea of the name of a co-product come from? is there a reason to it?"

"It generally expresses the behavior of a coproduct of two objects in category theory. The disjoint union is the coproduct in the category of sets."

Q31 "Is there a way to actually produce a non-canonical value of a type in Agda? Or are they just kind of ghosts required to make the theory apply to the language like Haskell's bottom value?"

"Not my area of expertise, but univalence lets you produce non-refl paths in identity types. Isn't that a non-canonical element?"

"what's univalence?"

"A topic covered later in the school :)"

Q32 "By including, for example, x: P+Q in the context in the conclusion of +-elim, are we saying that ind_+ can only accept a name in the third parameter? Why not an arbitrary term, with $Gamma \vdash r : P+Q$ above the line and $Gamma \vdash ind_+(p, q, r)$ below it?"

"Oh wait, is it because it has to be possible for D to depend on the coproduct term? I don't quite see why that's a difficulty if so."

"ind_+ accepts an arbitrary term of the coproduct type, but only computes for the "names", if I understand what you're saying. And yes, D can depend on the element of the coproduct type."

"Wait, are you saying that $ind_+(a, b, inl(x))$ doesn't reduce to $a[x/p]$?"

"No, I'm saying that it does. I'm saying that $ind_+(a, b, z)$ doesn't reduce to anything for general z."

"My original Q was about whether $ind_+(a, b, inl(x))$ is even a well-formed term, since the rule that introduces ind_+ (+-elim) requires the third parameter to be a name in the context, and inl(x) isn't a thing that can exist in a context, right? What is the rule that allows the formation of the term $ind_+(a, b, inl(x))$?"

"It's function application. You form inl(x) : A + B and then you substitute it in for z in $ind_+(a, b, z)$."

Q33 "Do inductive types only have beta (and no eta) rules?"

"Do inductive types only have beta (and no eta) rules?"

"This is a choice you can make for a type system, but some of our inductive types do also have eta rules."

"Later on, we'll be able to show in HoTT that induction principles give a uniqueness that can be interpreted as an a propositional eta-rule (so not a judgemental eta-rule)."

"What's the difference between propositional and judgemental? Is either of them the same as definitional?"

"judgemental is the same as definitional"

Q34 "is the ind function a postulate here that we can just use them like that? what is the type of ind?"

"The ind term can be shown to have the type of a certain Pi type. But yes the ind function is a postultaed term."

Q35 "Should p and q not then be in the context of $ind_+(a, b, x)$?"

"Sorry, meant this as a reply to my last question."

"Sounds like a good question for the discord :)"

"Fair enough, I'll post it there"

Q36 "I'll repeat my question, since it hasn't been answered fully - In the +-elim rule, why are both D(inl p) and D(inr q) above the bar, when it suffices to show one of them in the logical interpretation?"

"we need to show booth"

"both"

"Why do you say it suffices to show both of them in the logical interpretation?"

"Elimination rule is a case match. To do something in the general case, you need to do it for inl p and for inr q"

"I thought I heard her say we only need to show one of them."

"The intro rule is where you only need one"

"A dependent product over P+Q logically is "for every proof x of P+Q, we can prove D(x)". A proof x is either inl(p), where p is a proof of P, or inr(q) where q is a proof of Q"

Q37 "If all these things just behave the way you'd expect in sets/logic, is there any situation in which you'd be led astray if you just tried to do all this using your usual intuition regarding sets and logic to write down these deduction rules and such? Or perhaps (as was suggested Monday) in a generic locally Cartesian closed category? What's the benefit of writing all this in terms of these rules?"

"For one, by writing things in terms of type theory you can implement it on a computer. The intution you get from a lcccs is usually valid, but it's less clear when we get to higher inductive types."

"Well, by deriving terms in the type theory, you get interpretations of these terms in any model of the theory (such as LCCC for our current type system). For full HoTT, there are very strange models that do not behave like set-based categories."

"I guess my question is then why not define the conditions required for a category that models such a type theory, and just work within that. I've certainly never been led astray just kind of pretending everything's in a category when programming something in Haskell, for example.
Maybe this is just because I'm more category theorist than logician, though, that that would seem far more natural to me :)"

"You can define such categories (e.g. contextual categories, categories with families...) and they are in my opinion harder to work with than type theory itself."

Q38 "Does + (and times) behave categorically? Like if G : Group and H : Group, does G + H "look like" the free product in some sense?"

"'+ and times are defined on types, and groups are not types. So G + H is meaningless (until you define what it means to add groups)."

"Yes, these have very category-theoretic behavior. Times behaves just like a product in a category, e.g. the product of groups. Just an analogy tho, because, as pointed out, types and groups are quite different"

"Yevhenii is correct – you'll have to encode the coproduct of groups in terms of an inductive type on the underlying sets (types)."

"Thanks!"

"Does the CCC interpretation of lamda calculus relate to this question?"

"I don't think the category of groups is Cartesian closed, is it?"

"Group doesn't have internal homs (you need the codomain to be abelian)."

Q39 "is a Pi type a dependent product?"

"Yes"

"Yep, that's another name for it."

"how does that work intuitively? I tend to think of products as containing "two things" in their terms and sums as containing "either of two possible things" in their terms, but that doesn't seem to apply here."

"It's a product of n elements, rather than of just two elements. The notation is like the Sigma and Pi notation for sums and products in maths. So $A \times B$ is a product of two elements. $Pi_(a : A)B(a)$ is a product of —A— elements."

"I think replace 2 with any number, possibly infinite—or more precisely, if you think of 2 as indexing over Bool, replace that with indexing over any possible set"

"If A is Bool, then it becomes the binary product"

"ahhhh that makes sense"

Q40 "How is the word "dependent" really modifying the word "sum/coproduct" here? It seems to me this is going from sums over 2 (the + types we just saw) to sums over arbitrary sets, rather than adding any sort of special "dependency"—like, these would still just be called plain old "coproducts" (maybe "small coproducts," "arbitrary coproducts"), no? Why "dependent"?"

"It's dependent because the factors of the coproduct can vary over the "elements" of the base type."

"But isn't that already true for + types? You can add two different things, right, not just the same thing?"

"If you think of them as functions, then the codomain is varying ("depending") on the elements of the domain. When we get to higher inductive types, there will also be an action of the indexing type on the elements of the product/family which is a further "dependence"."

"The + type is the special case where the base type is Bool."

"I guess then I'm wondering what a *nondependent* sum would look like? Would that just be a binary product? Why call them sums to begin with, then?"

"If the type family P which yo're forming the Π-type with is constant (i.e. doesn't depend on it's parameter), then you get a constant (or nondependent) product or sum"

"A nondependent sum is a way of defining the binary product of two types."

"The name 'sum' is there for historical reasons but really it is misleading from a categorical point of view"

Q41 "So is Π 'universal' and is Σ 'existential'?"

"Yep!"

"That's one possible interpretation"

Q42 "Why doesn't Σ-form have '_ type' like 'Q(x) type'? And why isn't 'P type' a premise?"

"You're correct on all three. For the third one, we often omit premises like 'P type' if the other premises make clear that P is a type"

"Is there an implicit assumption that the context is well formed and, so, all type symbols within are well-formed?"

Q43 "Do some use notations in type theory that abstract out this common pattern to be able to define inductive type more concisely (e.g. something more similar to agda's notation)?"

"Yes, you'll frequently see some of the rules omitted or written more concisely/informally. See Sect 4 of book for examples"

Q44 "Is there any upside to defining sigma-elimination via an induction principle vs postulating the existence of projections directly? The latter seems more intuitive to me when viewing sigma types as a generalization of non-dependent pairs."

"The two are inter-derivable, so I can't think of a major reason to use one over the other."

"All inductive types have an elimination rule, but not all inductive types have projections (e.g. + doesn't have projections) So using elimination rule better generalizes to other more complicated types"

"And also the type of second projection is kinda ugly"

"There's actually a benefit to using the projections, because we can see sigma-types (and the unit type) as *record types*, and for these we can have eta-laws, i.e., for sigma: (pr1 z, pr2 z) =*= z"

Q45 "Jack Romo asked on discord: In the +-elim rule where we define ind+(a, b, x), each of a and b have p : P and q : Q in their contexts. Why don't both show up in the context of ind+(a, b, x)? How do they go away?"

"That sounds correct, but maybe let's discuss on discord"

"sure"

Q46 "Maybe a bit out of context, but can we have a brief digression on exotic models of the inductive type N?"

"I bet discord would love that :)"

Q47 "Why doesn't N-elim produce a Π type? Wouldn't $\Pi_{n:N} D(n)$ be a way of saying 'predicated D pertains for all naturals'?"

"It does produce a term of a Pi type."

"Oh. Hmm, I think I might have some deeper misunderstanding I need to address, but am not quite sure what it is yet."

"By abstracting over the x : N in the context in the conclusion of N-elim, you get the Π-type you wanted."

Q48 "What is the meta-rule that governs which inductive types are allowed in HoTT? if I just write down a collection of formation, intro and elimination rule, how do I know if that defines a type in HoTT?"

"That's a sophisticated question (maybe ask again on discord). There are some very broad classes of inductive types we know we're allowed to do, but for some more sophisticated inductive types, it can be hard to justify why they exist. We'll touch a bit on this later on with inductive types, but this question gets into a number of active research areas in HoTT. Thanks for asking!"

Q49 "what is the "technical sense" in which products & coproducts are "opposite", which you alluded to? Keywords for reading on the topic?"

"For reading you could look into limits and colimits in category theory."

"In category theory, these concepts are dual to each other—the definition of one becomes the definition of the other in a category where you reverse all the arrows of your original category."

"In type theory, to create a function from a type X to the product A x B, you need two functions X -¿ A and X -¿ B. Dually, to create a function out of the coproduct A + B to a type X, you need two functions A -¿ X and B -¿ X. The principles are "opposite"."