Agda lecture 3 Q&A (Solved)

HoTTEST Summer School 2022

The HoTTEST TAs

18th July 2022

Q2 In HoTT book isn't the elim rule a dependent function in x, y and p?

We give both the dependent and non-dependent version

The dependent one is at the top of the screen, a bit cut-off right now

Q3 There was a connection between Yoneda and refl. Is it just a curiosity or is there more to it?

This is the universal property of the identity type. It gets used a fair bit, but it is proven with the function extensionality axiom, so we'll get to it a bit later.

Some interesting things can be said about this, see here for example: https://www.cs.bham.ac.uk/~mhe/yoneda/yoneda.html**. It also contains a link to something that Egbert proved in his master's thesis :)**

This is how it started: https://homotopytypetheory.org/2012/05/02/a-type-theoretical-yoneda-lemma/

Q4 Should you generally try to do induction on as few variables as possible (i.e. does that make the proof more convenient to use usually)?

That is a good principle, but sometimes it's not ideal if you're using a asymetrical computation rule and you might patter match fully to prevent that use case

As a rule of thumb, I'd say yes, because in proving things about a definition one typically needs to do induction on the same things. So fewer inductions might yield shorter proofs.

Defining a function by 'unnecessary' pattern matching is, however, a pretty neat trick when you want to prevent your definitions from unfolding when you don't want them too. This can speed up type-checking significantly

On the other hand, if you can get away with defining your function without induction, but reusing other functions, then the advantage is that you can prove stuff about your function by using known stuff about the functions you used in its definition.

Q5 Are these levels roughly the same as the successor universes we discussed in the HoTT lecture or are these fundamentally different ideas?

That presentation closely followed Agda's apprach!

Q6 Does the use of the variables i and j introduce implicit extra arguments to Σ, as in $\Sigma\{ij : Level\}$, or how does that work?

Yes! As explained by Martin now :)

Yeah, to fully specify the arguments in Σ now, you have to write $\Sigma\{i\}\{j\}\{A\}B$

This also applies to the pair constructor $\_,\_$

Q7 Between the =-elim we saw before, and this universe =-elim, is one of them more fundamental than the other?

One is more general and coveres the induction principle of all equality types simultaneously as opposed to just those at one level

This one works for all types (of any universe) to it is a generalisation of the first one you saw

I think that our original definition of the equality type was just at the base level, though, so the old principle was fully general for that type

Q8 if the universe specifier $\mathcal{U}$ is the same as Type (or Set) which we've been using before, why didn't we have to specify a level to use it before? How can $\mathcal{U}$ and $\mathcal{U}i$ both fill the same spot syntactically?

When we wrote Type before, it really meant U lzero, the universe with the 'lowest' index.

Type / Set is special syntax in that it can both take a level argument or not

Q9 Is there a reason to define $\mathbb{P}$ in the zeroethlevel? Would it be a problem to define it in an arbitrary, non-zero level instead?

No, you can always lift to higher universe levels. It would be an issue to define types in a higher universe level than necessary

It automatically is with this construction (from the definition of sigma types and the fact that N is in the zeroth universe). There is a type former usually called Lift which "lifts" a type from a universe to a higher one, but that would just complicate things unnecessarily.

You would have to write Lift, though

In more detail, the level of a sigma type is the max of the levels of it's two components, so if both are at the base level, then the sigma type will also live at the base level

Q10 In what way is it 'not good' for there to be different proofs?

If you're considering composite numbers, then presumably you want such numbers to be equal when they're equal as *just* numbers. In a way, it's the difference between studying composite numbers and compositions themselves. Does that make sense?

One reason would be that you couldn't view CN as a subtype of the naturals anymore

Sometimes, when you translate a property using the Curry-Howard interpretation, you end up with a type that can have more than one element. In other words, it has become proof-relevant due to Curry-Howard. On the other hand, mathematicians might not want to consider the property as added structure. In that case there must be a way to say that the type is inhabited without having to provide an explicit element. To say that a type is inhabited, we use the propositional truncation, wich is an operation that turns a type A into a new type $||A||$ that has at most one element.

Q11 Is there a double bar missing on the Exists version of CN?

It's a typo: the final || shouldn't be there. We write $\exists$ to be $||\Sigma||$

Q12 Is propositional truncation something that only comes with HoTT or was it also there in Martin Löf Type Theory?

Definitely HoTT only. You need to add axiomatic postulates to MLTT to truncate

But this idea was around before HoTT: http://math.andrej.com/2004/05/04/propositions-as-types/

So does that mean that the injection could not be done in Agda normal or even Lean?

Not in normal Agda or Lean without postulating its existence. You need something like higher inductive types to give an explicit construction.

Right, the specific situation is that truncation can be defined as a HIT, which is a HoTT feature, but it can also be added as a language extension of MLTT using things that don't look like HoTT at all

Lean also has propositional truncations.

Q14 Are there propositions that are not sets?

No, every proposition is a set:-)

(But the proof of this fact, I think, will have to wait)

It is proven in section 12 of the book, which is indeed a bit later.

Q16 What is the difference between data and property?

What is the difference between data and property?

Properties live in a type that is a proposition, that it, the only useful information about them is whether they hold or not

A property can hold in at most one way. But data/structure can have many elements/instances.

So there can only be one of a certain property, but multiples of a certain data/structure?

for example, "being a composite" is a property of the number 6, while "2 * 3 = 6" is data proving this property, and so is "3 * 2 = 6"

That makes sense. Then in order to have data do we have to define properties first?

it's usually the other way around: you define a property by saying "there exists data that proves this property, but i don't care which data"

in other words, if i give you a proof that 6 is composite, you don't care *which* numbers i used to prove it, you only care about the end result

Q17 How are they property automatically?

It is not automatic. It's something you have to prove. Moreover, it requires function extensionality to prove that is-set(X) is a property.

Is it is-set(X) that is used to prove that $(x : X) \to (x * 1 == x)$ is a proposition?

Yes!

Q18 Does this same definition of is-prop still work with HITs and univalence? At first glance it seems like $\mathbb{S}^1$ fits this definition of is-prop but (if I'm understanding correctly) it's not a set

You can't define a *uniform* way to connect any two points on the circle

You're maybe thinking of the fact that $\mathbb{S}^1$ satisfies $||x = y||$ for all elements x and y of $\mathbb{S}^1$. But this is different from is-prop ($\mathbb{S}^1$), because of the truncation involved!

is uniform synonymous with continuous in this context?

You could, for example say the rule "go closkwise from the first point to the second point, but then, as you move the two points to cross each other, the definition has a discontinuity

And, yes, you need to relate two arbitrary points, so you take them as arguments and they go in the context

err to be clear by "context" here I wasn't asking about type-theoretic context but "in this context" as in "while we're talking about this topic"

Oh, whups, yes, except that continuous is a topological term while uniform is a type theoretic term

But you should think about it topologically

if you try to give a path from base to x continuously as go around the circle, you'll be in trouble while you make a complete turn because you'll suddenly "jump" from the loop to refl, and these aren't equal

My example about the clockwise rule is good; you can think about other rules as well (such as "the shortest path"); nothing will work

the filled disk doesn't have that problem (you can just use the shortest path), so it is a proposition

Q19 Why Agda can prove false != true? Does it have universes internally, or it's like an axiom that these are different constructors?

This proof using universes that we are doing now, I am not sure when Agda is cheating or not

When you see () then Agda is 'cheating'. I don't know how () works internally, but it seems to be that different constructors can't be equal.

Q20 Can you tell Agda to only allow MLTT proofs?

There's no fool-proof option, but: (1) be conservative in what Agda features you use; (2) don't use the () pattern.

—safe —without-K helps a bit, but the fact is that Agda has like > 100 optimisations always on and things like pattern-matching are based on case trees and not eliminators, so no

Also —exact-split helps

Q21 How do we know when we have done a proof that is not in Martin-Löf type theory?

In this case you can tell because it's using the () pattern.

On the other hand, when we're doing large formalization projects, we often just don't bother to bring this kinds of proofs to a form that would be closer to something in MLTT. We just use Agda's pattern matching magic:)

Q22 Does this also mean that there's no MLTT proof of, say, $\mathrm{Fin}\,0 \to \emptyset$, without universes?

You can't even define Fin without universes

you can define it as an inductive data type can't you? anyway, assume you have defined it

Ah, but data is also not really a feature of MLTT. Your question was about MLTT:)

Q24 how does it relate to using 'variable'?

Variables always become implicit when they are used in a signature, whereas modules can have explicit parameters; you can also give modules a name

Modules give you more control about the scope, i.e. when importing a module from another file. Records are also considered modules.

Naming a module lets you open it with some arguments provided

If one function in a module calls another function in the same module, can it have different values of module parameters?

No it cannot!