Agda lecture 7 Q&A (Solved)

HoTTEST Summer School 2022

The HoTTEST TAs

8th August 2022

Q But if we want we would still be able to use eliminators?

Do you mean for the cubical interval type or for higher inductive types?

...both? I don't know

HITs in Cubical Agda work very much like ordinary inductive types in Agda: you define maps out of them using pattern matching, but you could define the eliminator by pattern matching if you prefer to use that.

Since Cubical Agda is kinda bad at type inference (I promise I'm working on it) we generally tend to define eliminators for our HITs *anyway*, because they help Agda not get confused

Q what is ==?

Definitional equality, written $\doteq$ in Egbert's book.

Definitional equality. The term $p\,i0$ has to be *precisely* x

Q Is this definition of path types equivalent to our old definition?

Yup! We'll show that the Path types satisfy J either today or in the next lecture (hopefully today). This lets you prove that they're equivalent to the inductively-defined Id types

Yeah, we can define the usual path type in Cubical Agda too, and easily prove them equal.

But they do have different computational behaviour, right?

Like many pairs of equivalent types, they behave differently up to definitional equality, but the same up to homotopy.

Q I guess we will learn why `I` is not a datatype? It seems so far like I could just be `Bool`!

Yes, we'll get there :) It is rather special type.

And I guess the reason it is not a datatype must be related to cubical mode.

If it was just Bool, everything would be equal to everything else

For now, `I` has two constructors, `i0` and `i1` and that's all?

those are not constructors: you can't pattern match on them! in other words you can't tell the endpoints apart internally

The intuition is that it represents the whole interval between 0 and 1, but we can only construct the two endpoints, not 0.5 etc., but when consuming it we need it to work for the whole interval, not just the endpoints.

To elaborate on the previous answers, if `I` were defined as `Bool` then we could prove arbitrary equations e.g. $0 = 1$ by `Bool` (`I`) recursion, sending `i0` to 0 and `i1` to 1.

Q Why does `ap` take an $i : \texttt{I}$?

To introduce an element of $x \equiv y$, you bind an interval variable. On some level, $x \equiv y$ is "just" $\texttt{I} \to A$, but with information about the endpoints

Good question! That's because the type $(fx \equiv fy)$ is "actually" functions $\texttt{I} \to B$ that are fx on `i0` and fy on `i1`.

Q Where do we get the x from in this fun ext definition?

You need a path in the function type: so take $i : \mathtt{I}$ and now an argument of the domain of the function $x : A$

Forgetting about endpoint information for a second, funext is just $\mathsf{flip} : (A \to \mathtt{I} \to B) \to (\mathtt{I} \to A \to B)$

He is actually proving that $\lambda x.f(x)$ is equal to $\lambda x.g(x)$. That's where the introduction of x comes from. But $\lambda x.f(x)$ is definitionally equal to f and $\lambda x.g(x)$ is definitionally equal to g, so this indeed gives a path from f to g.

Q In HoTT tract, funext depends on univalence. So the `--cubical` option gives *something* to prove funext and ua?

Yup :-) So funext follows immediately from the judgemental structure (you can swap the $\mathtt{I}$ argument with the A argument), and ua is... a tad more complicated, but it's also provable. I think we'll get to it next time.

Yup! Both are provable (ua is more involved since we'll have to talk about a different kind of type called Glue types, but I think Anders will mention them at some point)

The "jugdemental structure" bit is throwing me off... So there is not a "solid axiom" that cubical postulates, but it changes the whole type theory? Can we then even compare the old type theory with the new?

It is a new type theory and how it compares to the old one is a very good question that, afaik, warrants further research. E.g. some conservativity result would be nice to have.

Maybe I am confused because of the mysterious nature of interval type. Does it have the ordinary 'intro/elim/comp' definition or does it use new machinery?

It uses new machinery, it's unlike a 'normal' MLTT type.

It has new machinery, so to speak. There aren't any elimination rules. I'll ask Anders to elaborate after he's done explaining the min/max/sym connectives

Q How are the "boundary conditions" implemented?

It's magical (built very deeply into cubical agda), I can elaborate on Discord if you'd like

Q Are `min`/`max`/`sym` builtins, or are they somehow defined in the prelude from other builtins?

They are genuinely built in, there isn't any hidden lower-level machinery behind them.

Q Is `PathP` like dependent $=$-elim?

`PathP` **is more like the PathOver type**

Ohh, the constant family $(\backslash i \to A)$. **Thanks**

Q If `PathP` is primitive, what is it over? Another `PathP`?

The argument of `PathP` **has type** `I` $\to$ Type, **so there isn't an infinite regress.**

Q There is not a way to pustlate `I`, right?

There isn't. It's really quite magical. As an analogy (maybe more accurate than you might think):
You can think of the interval as a type codifying "the shape of paths";
Really, you should think of *any* type as being a "shape" of sorts. For example: The unit type is "the shape of points" — a function $\mathsf{Unit} \to A$ is the same thing as an inhabitant of A.
The interval type is something we add to type theory, which for technical reasons has to be *very* magical (so it can't be an ordinary type like Unit), to keep track of the geometric operations we can do with paths.

Q In this formalism where paths are functions out of the interval, what binds this concept to the idea of equality? Could we use this type theory (perhaps minus an axiom or something?) as a model of connections between things we don't want to think of as equal, like transitions or diffs?

There is another primitive that implements transport for this notion of equality. We haven't seen that yet, but that's basically what makes it equality.

We'll see it in the next lecture (this time around, Anders wanted to stick to the nice things and not go too much into the gory details). But essentially the idea is that (a) everything respects paths (this is the "transport" operation we've mentioned a few times) and (b) you can prove the J rule using transport and operations on the interval. But! Like you said, you can kinda weaken cubical type theory to talk about *directed* paths instead, that don't necessarily have inverses. I'd be happy to talk about this more in Discord :)

Q How is the composition • defined?

It's defined using a builtin called `hcomp`. This is a bit too advanced for now, but I think Anders will talk about it later. In Cubical, defining the basic things are usually a bit trickier than in standard Agda, but when you have them, things get easier (in my very biased opinion, at least)

Ah okay, I guess I'm too used to the way we learnt type theory so far where everything is built from the ground up

Yeah, we decided to start by showing the nice things in Cubical Agda before getting too far into the gory details, so we don't lose track of where we want to end up. This is backwards from the way we've been doing things before, which was starting from nothing and building up type theory. If we directly started building things up in Cubical Agda, the first thing you'd have to hear was "unlearn everything you've heard about so far", which.. is less than encouraging, to say the least. By starting with the niceties, we have motivation for doing things in our weird, cubical way :)

Q What is pos?

The positive half of the integers

In Anders' definition of the integers we have constructors pos for POSitive, and negsuc which represents $-(n+1)$

Q Is the intuition for helix that $\mathsf{helix}\,(\mathsf{loop}) = \mathbb{Z}{+}1$, $\mathsf{helix}\,(\mathsf{refl}) = \mathbb{Z}$, $\mathsf{helix}\,(\ \mathsf{loop}) = \mathbb{Z} - 1$?

Seems right. It's really $\mathsf{ap}\,\mathsf{helix}\,(\mathsf{loop})$ for this to be well-typed, and applying helix to loop is "the same thing" as applying the equivalence $\mathbb{Z} \simeq \mathbb{Z}$ which adds 1

What is “_”?

The underscore, as a definition, is a way of asking Agda to verify something without having to giving it a name. It’s equivalent to inventing a name that you never use afterwards

You can also use _ several times in the same module without clashes

So it gets discarded and you can’t reference back to it?

Yup.

Q Can we no longer define square as a term of $\mathsf{line1} \bullet \mathsf{line2} \equiv \mathsf{line2} \bullet \mathsf{line1}$?

We can, but when working cubically we almost always prefer to do it this way. The eliminators become much more well-behaved if we use `PathP`. This definition is equivalent however.

Q As a less-mathy person: are the constructions of these spaces considered to be part of homology? i.e. if I know homology would I understand how to construct a Klein bottle in this framework?

It’d be part of topology, I’d say. Check out gluing diagrams for CW complexes. Here’s a random introduction to gluing diagrams I found when googling. Looks kind, but I can’t say I’ve read it: https://mathcircle.berkeley.edu/sites/default/files/archivedocs/2010_2011/lectures/1011lecturespdf/bmc_topology_manifolds.pdf

Q What's the philosophy behind using cubical rather than simplicial/globular/some other model?

The cubical interval is built by taking products (there is, as always, a way to make this precise). Contexts in type theory are *also* made by taking products! So they're a match made in heaven.

It was known that simplicial sets were very difficult to do constructively and if you're trying to build a computational system it's helpful to have things work constructively.

The type theory was, I think, pretty much built by reverse engineering, starting with the model

Yup! BCH model CCHM type theory. And the 'M' there is lecturing now :)

That's incredibly cool! I know the topologists like to start with simplicial stuff, but it can certainly get really weird to wrap your head around, so I've always wondered exactly how other models can be more helpful for other purposes and this is certainly a really concrete example of one such case

On a historical note, Kan originally wanted to use cubical sets for homotopy theory, but because of a handful of technical issues with the cube category he was using, they weren't as well-behaved as the simplicial complices. Surprisingly our cubes here don't have the same issue (search for "cubical set with connections" on the nLab to read more)

Q Is there a way to avoid reasoning about intervals when pattern matching? for example shouldn't we be able to do $\mathsf{flip}\,(\mathsf{merid}\,a) = \mathsf{sym}\,(\mathsf{merid}\,a)$?

Not at the moment, but this is something I've been thinking of implementing. Agda doesn't let you omit the interval arguments to the constructors.

The mechanics of the HITs in Cubical Agda mean we have to write it with the interval variable i in scope, like Anders did in the lecture.

I don't think $\mathsf{flip}\,(\mathsf{merid}\,a)$ would type check, you need $\mathsf{ap}\,\mathsf{flip}\,(\mathsf{merid}\,a)$ (which is exactly the same as $\backslash i \to \mathsf{flip}\,(\mathsf{merid}\,a\,i)$)

the ap notation is nice I guess because it resembles how we did stuff in the previous lectures with the postulates

Q Why propositions are not definitionally equal?

Any two propositions? Surely, you don't want the empty and unit type to be equal?

A type being a proposition is something that can be proven, unlike being in the Strict Prop universe (Prop in Agda).

Or why are any two elements of a proposition not def. equal? That's because it's defined in terms of the identity/path type. It's a definition inside type theory so can't refer to def. equality.

Q Are there examples that relate more to programming? (DBs or OO or FP) ?

I guess the pushout is close

Maybe something like this: `https://arxiv.org/abs/2009.05547`? A paper by Carlo, Max, Anders and Evan (Cavallo).

Q Is synhetic roughly a synonym for constructive?

It basically means "axiomatic" or "relative to some abstracted-away interface". An analogy would be doing geometry relative to Euclid's axioms, vs doing it "analytically" by working in the Cartesian plane.

Q Do you think Kevin Buzzard will use Cubical in the future?

When I prove ZFC inconsistent, he will :-)

But jokes aside, I'm under the impression he finds Agda too "computer sciency", so I'd imagine him preferring coq. But people on Twitter are still actively working towards making him change his mind...