📝 Notes

Apr 30, 2026

{ lean-mode }

Invited Claudiu Leoveanu-Condrei robert maria nhigueraruiz@gmail.com corpaci.luiza@gmail.com andrei.reanu@gmail.com

Attachments { lean-mode }

Meeting records Transcript

Summary

Study session on dependent type theory covered Lean programming fundamentals and system architecture via constructive mathematics analysis.

Lean Type Theory Fundamentals
Participants explored first-class types and proof tactics using the info view. Discussions clarified the distinction between propositional truths and boolean values.

Constructivism and Universe Hierarchies
Analysis of inductive definitions highlighted the importance of constructivism in type theory. The team examined infinite type hierarchies to prevent self-referential paradoxes.

Study Goals and Scheduling
The meeting resulted in a decision to move recurring sessions to Monday. Study focus remains on foundational mastery to facilitate future software verification.

Rate this Summary: Helpful or Not Helpful

Decisions

We've **updated the Decisions section** using your feedback.

Let us know what you think: Helpful or Not Helpful

ALIGNED

- **Curriculum coverage scope for next session** The curriculum coverage scope is set to chapters 2.3 through 2.7 for the next session, in accordance with the model's recommendation.

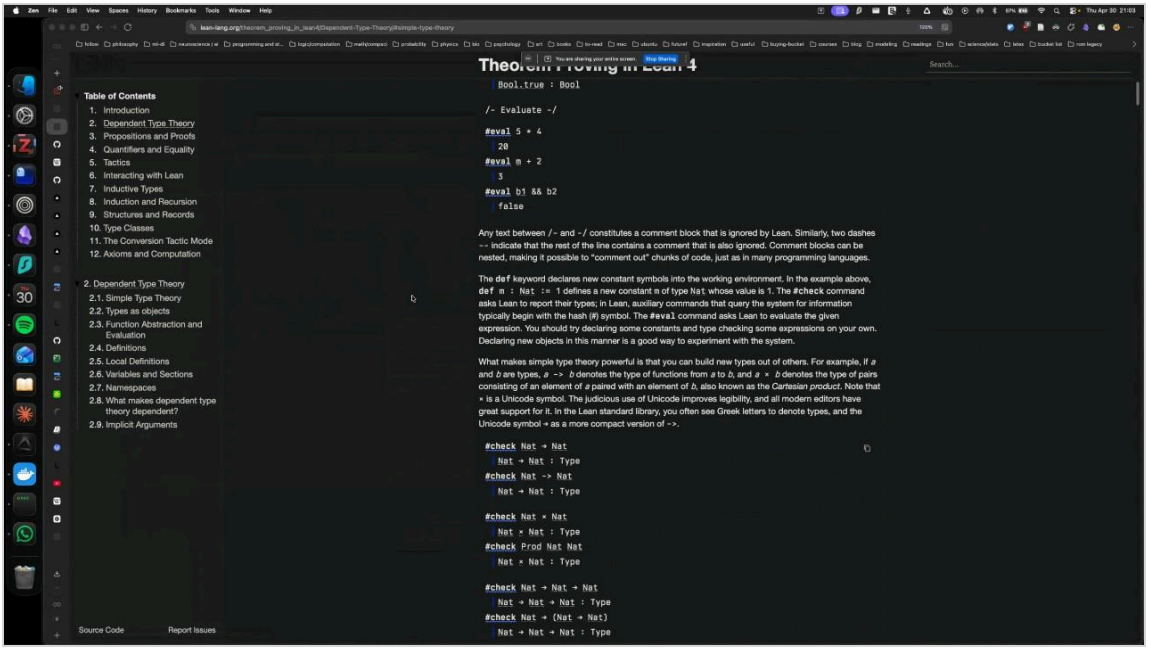
Next steps

- [ ] [Claudiu Leoveanu-Condrei] Study Inductive: Reread material; constantly strive to fully understand inductive concepts and connections to constructive mathematics.
- [ ] [Claudiu Leoveanu-Condrei] Share Notes: Share the file containing all researched notes and questions on type theory concepts, including the rabbit holes discussed.
- [ ] [Claudiu Leoveanu-Condrei] Update Invite: Send new calendar invite. Update recurrence to Monday meetings, beginning May 11.
- [ ] [Claudiu Leoveanu-Condrei] Upload Files: Upload the current meeting file/document onto GitHub. Ensure the file is accessible for review.

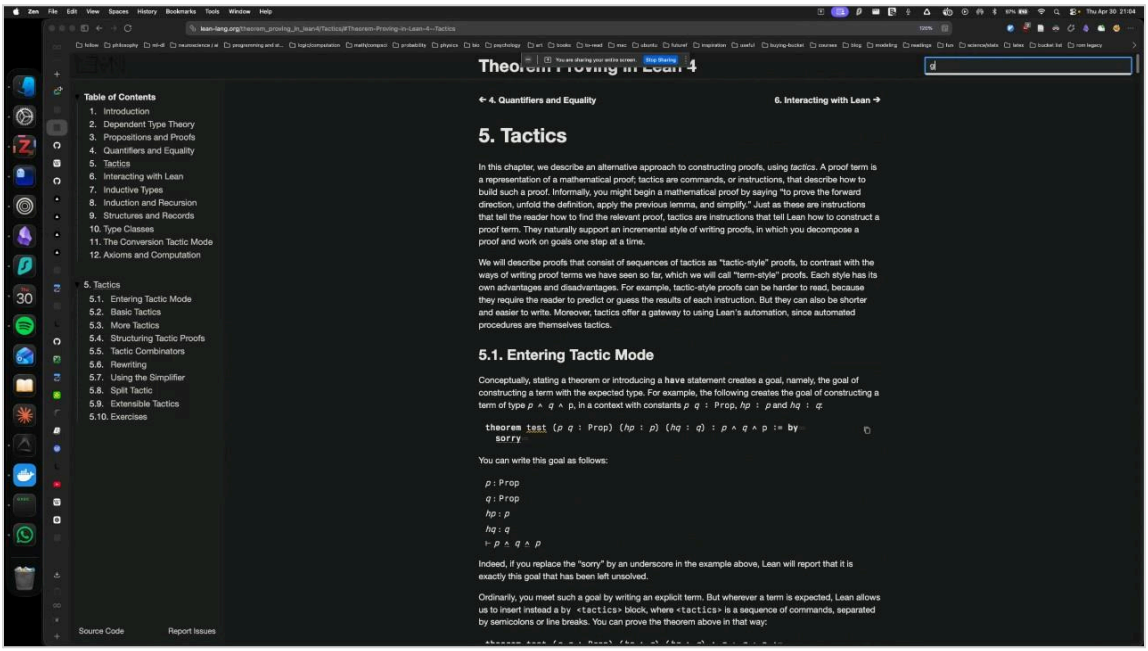
Details

Did the screenshots in this section make your notes better or worse?

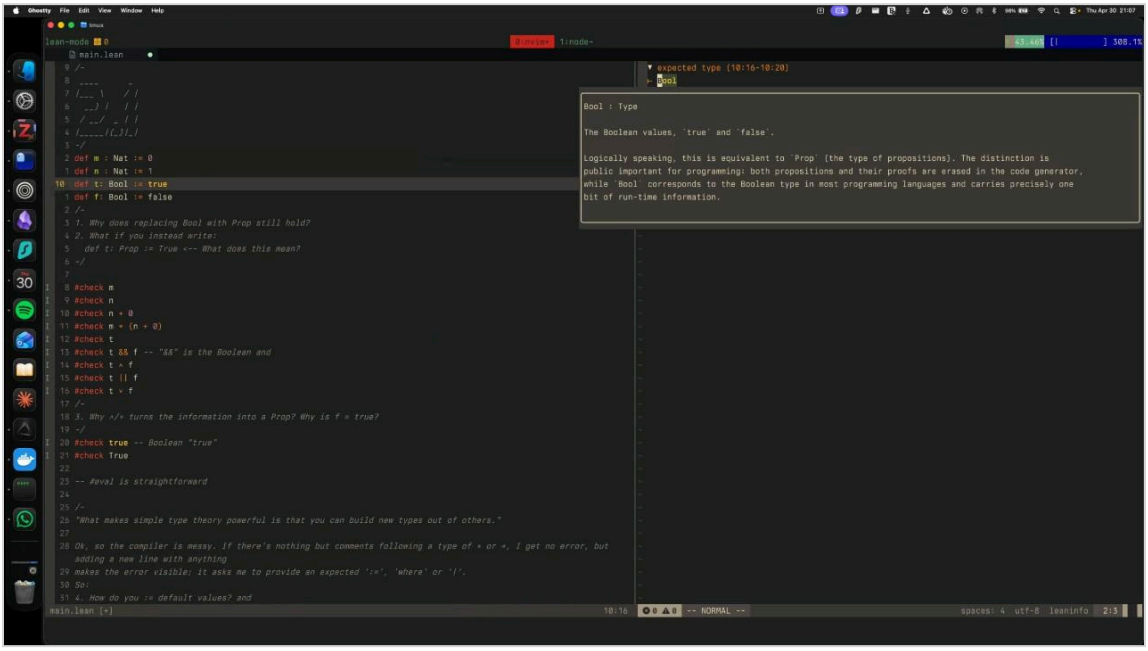
- **Session Introduction and Chapter Scope**: Claudiu Leoveanu-Condrei initiated session number two, covering the first two subchapters of the book's Chapter 2, "Dependent Type Theory": "Simple Type Theory" and "Types as Objects". Claudiu Leoveanu-Condrei noted they focused heavily on the code examples, which led to significant time consumption and the emergence of several questions. They expressed curiosity about future sections involving "tactics," which they explained are ways to prove something, such as using reflexivity or applying certain theorems (00:01:25).



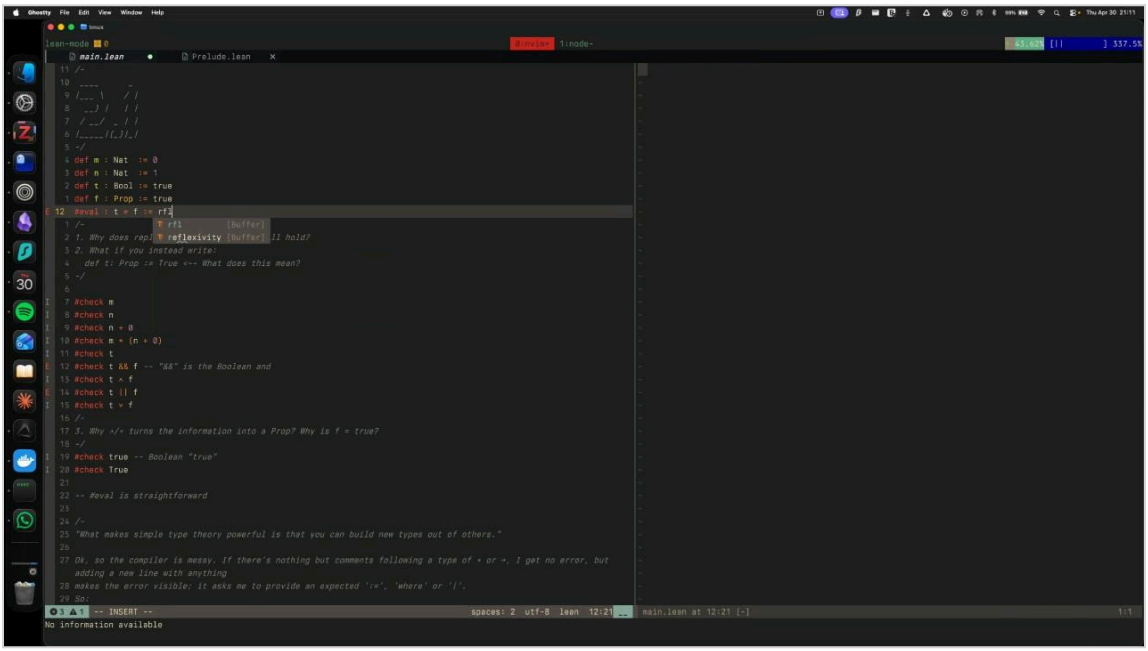
- **Tactics and the 'Grind' Tactic**: Claudiu Leoveanu-Condrei discussed proof tactics, mentioning that 'grind' is a more recent tactic that abstracts away a significant amount of manual programming by instructing the computer to perform the proof. They noted that the 'grind' tactic is not included in the current version of the book (00:02:24).



- **The Info View Panel**: Claudiu Leoveanu-Condrei introduced the panel on the right, called the "info view," which displays extra information, metadata, and details about types. They mentioned that new "widgets" have been added that allow for images and visualizations to appear in the info view (00:03:20). Andrei Ungureanu confirmed that they also use the info view and mentioned using the NeoVim extension (00:04:25).



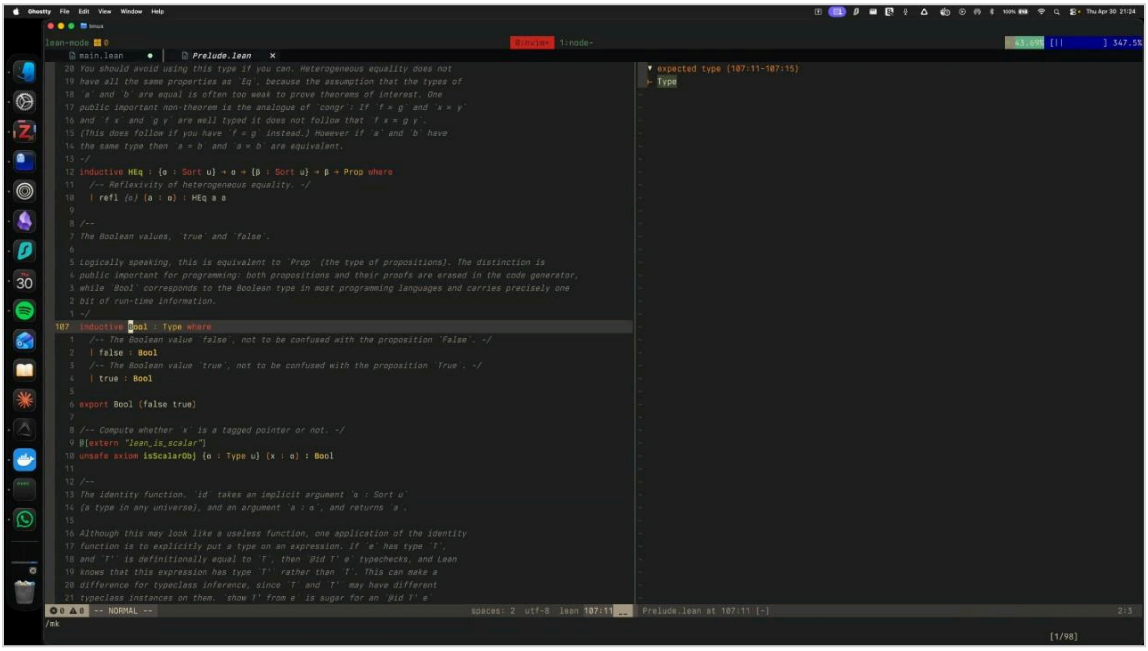
- **Using the Info View and Code Navigation**: Claudiu Leoveanu-Condrei explained that they frequently use the info view to "peek into things" and noted that the panel moves as they navigate through the code on the left (00:03:20) (00:06:20). The info view's display is based on the current range in the code, showing the corresponding details (00:05:28). Claudiu Leoveanu-Condrei also noted that pressing Shift+K in NeoVim usually shows similar information to the info view (00:04:25).
- **Variables, Types, and Propositions (Prop vs. Bool)**: Claudiu Leoveanu-Condrei observed that defining variables and assigning values is basic functionality but raised a question about why replacing `Bool` with `Prop` still holds true in certain examples (00:06:20). They explained that `Prop` (proposition) and `Bool` (boolean) live in different type universes, which was a point of confusion for them, and noted that `Prop` results in a sort (00:07:22).
- **Equality and Proof by Reflexivity (RFL)**: Claudiu Leoveanu-Condrei demonstrated that defining two variables as equal, such as `t = f`, can be proven using reflexivity (`RFL`) (00:08:31). They noted that this statement is essentially their first proof. They found that lowercase `true` and capital `True` behave differently, with capital `True` being a type and lowercase `true` being a value (00:09:47).



- **Types as First-Class Objects**: The discussion clarified that in Lean, types are first-class objects, meaning they can be returned, manipulated, and passed to functions without values. This concept explains why one can replace and

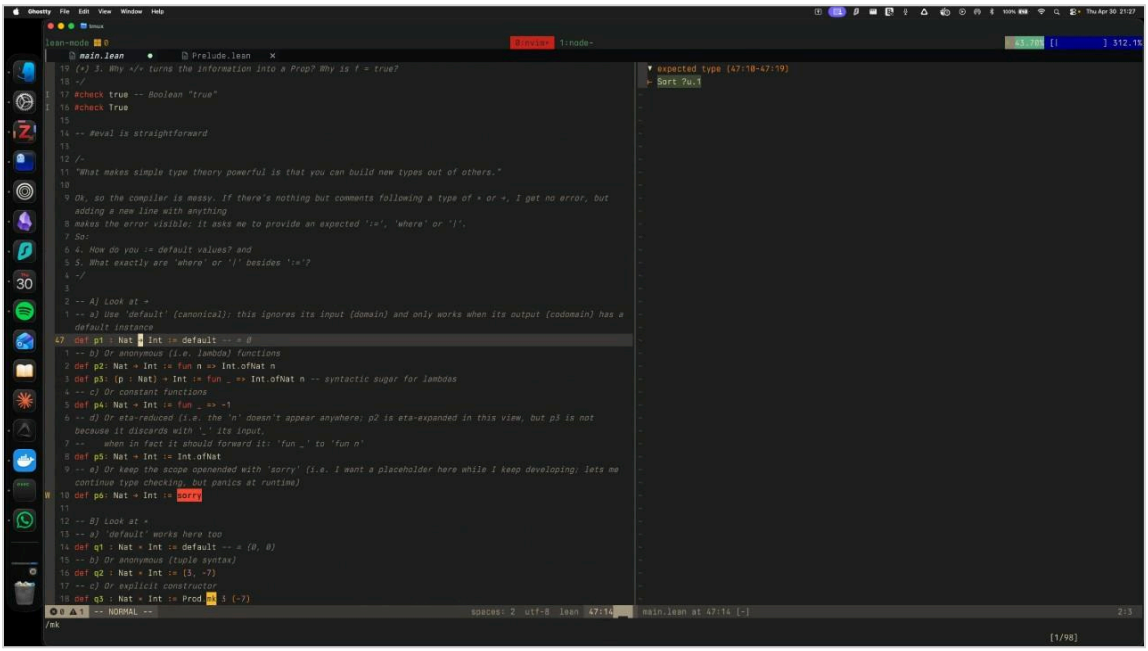
perform certain manipulations, such as equating a type (like `Bool`) to a value (like `true`), though they noted this is where the language starts to be "mindbending" (00:12:52).

- **Distinction Between Propositional and Boolean Truth**: Claudiu Leoveanu-Condrei noted the distinction between the boolean value `false` and the proposition `False`, which the documentation explicitly warns not to confuse. They assumed that capital `False` denotes a proposition where one is stating that P is false without performing value checks (00:21:55).

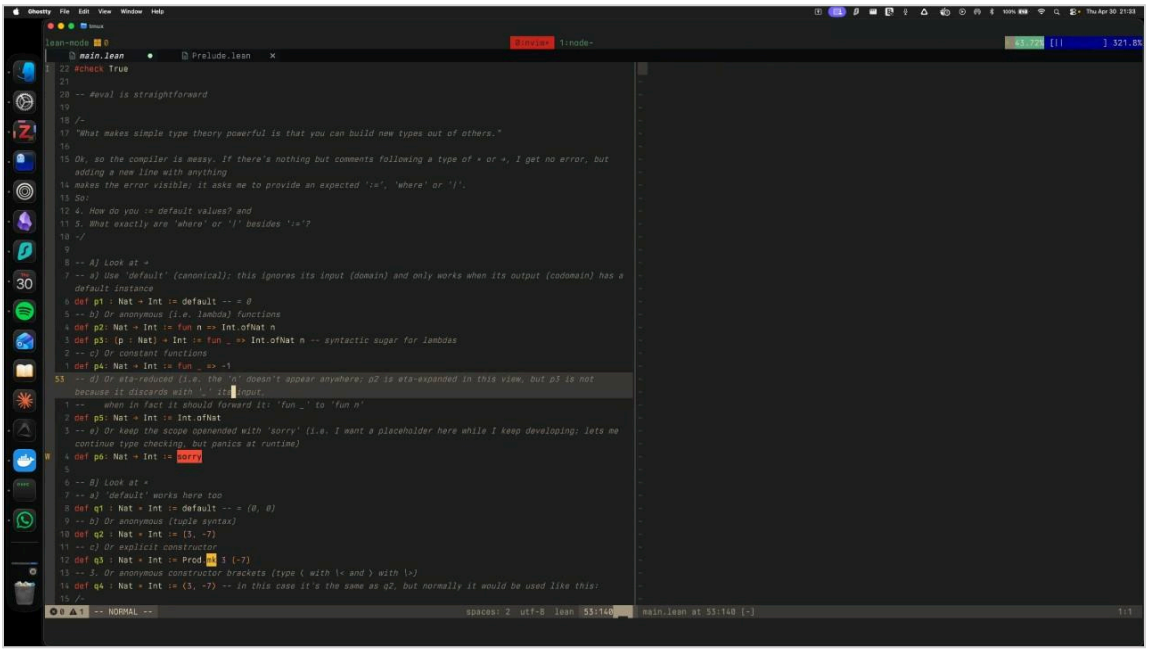


- **The `eval` Command and High-Precision Arithmetic**: Claudiu Leoveanu-Condrei demonstrated the `eval` command, which allows for evaluating expressions, including high-precision arithmetic (00:21:55). They noted that the language handles extremely large numbers quickly, such as calculating two to the power of 59. They further clarified that the system uses the GMP library for big integers (00:23:00).

- **Compiler Behavior and Code Blocks**: Claudiu Leoveanu-Condrei observed a peculiarity in the compiler where an error only becomes visible if a new line with any expression follows a type definition; if only comments follow, no error is displayed (00:24:01). They also looked into function definitions, noting that using the arrow notation creates a mapping (a function), for example, from the natural type to integers (00:24:56).



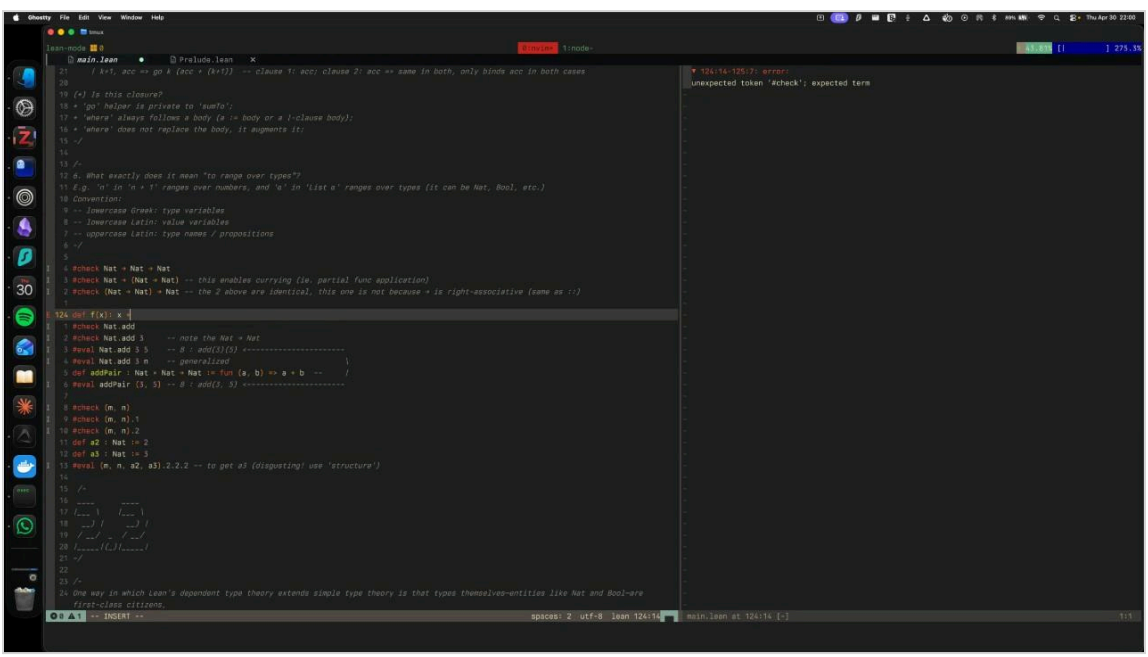
- **Default Values and Lambda Functions**: The discussion covered how default values can be set for functions using a canonical default that generally aligns with the zero equivalent for a specific type (00:26:11). Claudiu Leoveanu-Condrei also reviewed syntax for anonymous lambda functions, noting that the arrow and the `lambda` notation are syntactic sugar for the same concept (00:27:11).
- **Eta Reduction and Expansion**: Claudiu Leoveanu-Condrei introduced the concepts of eta reduction and eta expansion, noting that eta reduction occurs when a variable, such as `n`, does not appear in the resulting function body (00:29:21). The difference between functions P2 (which propagates `n`) and P3 (which discards the input using an underscore) illustrates this concept (00:30:18).



- **The `sorry` Placeholder and Proof Structure**: Claudiu Leoveanu-Condrei explained that `sorry` is a very useful placeholder for writing proofs, allowing them to lay down the structure of a proof without filling in the details (00:31:32). They noted that the interpreter accepts `sorry`, but the script will crash if run (00:32:29).
- **Cartesian Products and the `mk` Constructor**: They discussed the use of Cartesian products, noting that the default for a tuple is `0, 0` (00:32:29). They also examined the `mk` constructor, which serves as a convention for initialization, similar to `new` in Rust or `__init__` in Python, and is used to create structures (00:33:29).
- **Pattern Matching and the `where` Clause**: Claudiu Leoveanu-Condrei explained that the pipe symbol (`|`) is used for pattern matching, which can perform "very smart pattern matching" even at the type level (00:37:11). They clarified that the `where` clause is used to create local helper functions or closures within the scope of the main function's body (00:41:30) (00:48:27). They also noted the convention that type definition notation (using arrows) should be read from right to left, where the last type is the output and the first types are the inputs (00:42:30).
- **Variable Conventions and Associativity**: They reviewed conventions for variable naming: lowercase Greek letters for type variables, lowercase Latin letters for value variables, and uppercase Latin letters for type names and propositions

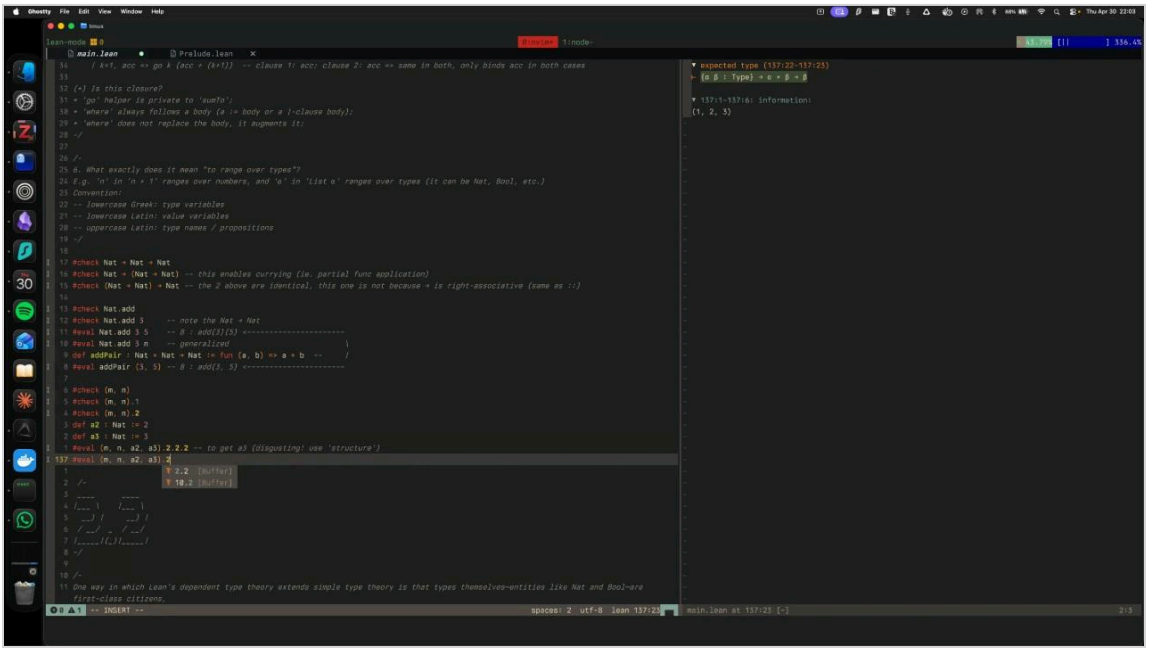
(00:49:38). Claudiu Leoveanu-Condrei highlighted that Lean's operations are generally right associative (00:50:54).

- **Currying and Partial Function Application**: Claudiu Leoveanu-Condrei explained that the right-associative rule for function types enables currying or partial application, which is the default mode in Lean for function application (00:52:54) (00:56:31). They demonstrated how an addition function (which normally takes two inputs) can be partially applied with one input to return a function that expects the second input (00:54:24) (00:57:51). They also showed how to define a function that takes a Cartesian product (a pair) to explicitly avoid the currying behavior (00:56:31).
- **Evaluation and Function Application**: The participants discussed the application of functions, noting that when a function is partially applied, such as `add of three`, it cannot be fully evaluated (`eval`) without the remaining input. However, the `check` function allows for inspection of the resultant type even when inputs are incomplete, providing insight into how the application will occur (00:58:56). The deduction of the application mechanism is made possible by observing the function's type (00:59:53).

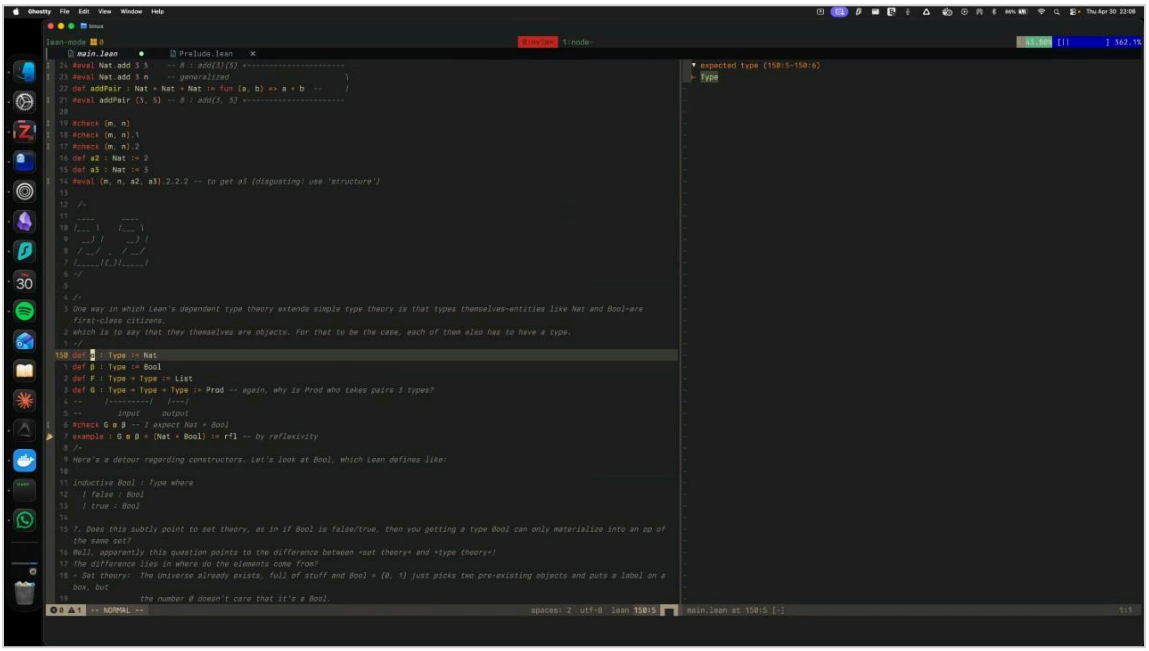


- **Checking Tuples and Unpacking**: The conversation shifted to checking tuples and the implied limitation of manually addressing numerous inputs. Tuples are unpacked sequentially by repeatedly taking the second pair, which is described as similar to a head-and-tail operation where the head is one member and the tail

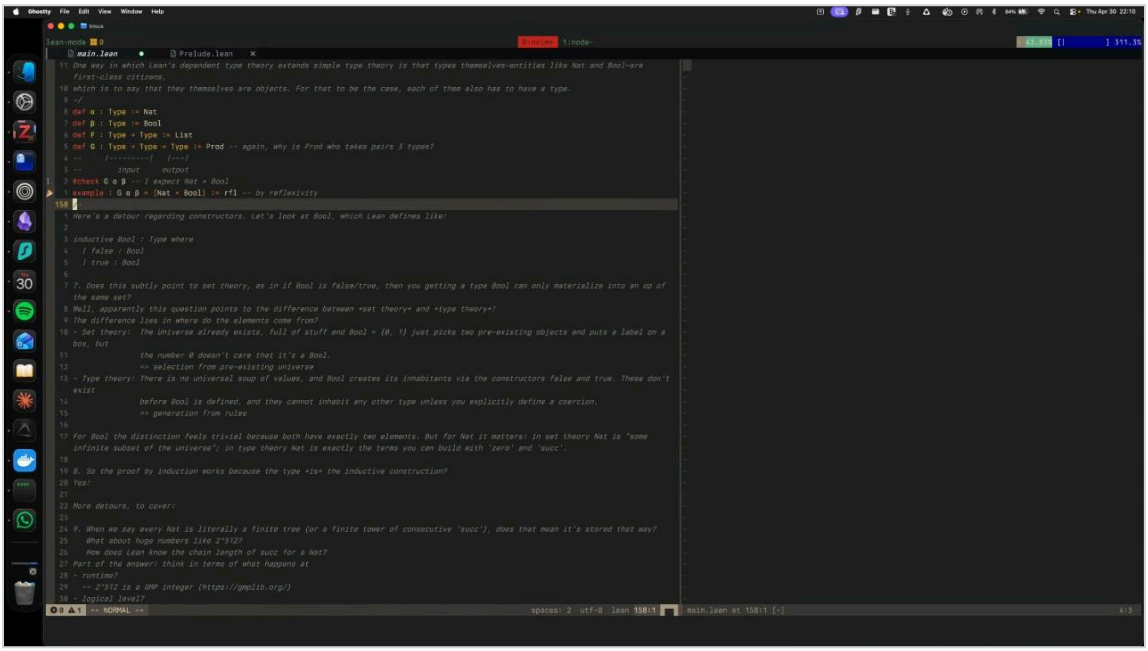
is the rest (00:59:53). This process of unpacking is likened to a literal binary search due to the continuous splitting and branching (01:01:15).



- **Alternative to Tuple Unpacking**: For situations involving numerous inputs, the recommendation is to use a structure with dot notation to access memberships instead of relying on the successive unpacking method (01:01:15). The structure of a tuple is demonstrated by showing how an input is broken down into its components, where `one` represents zero and `two` represents a subsequent grouping (01:02:22).



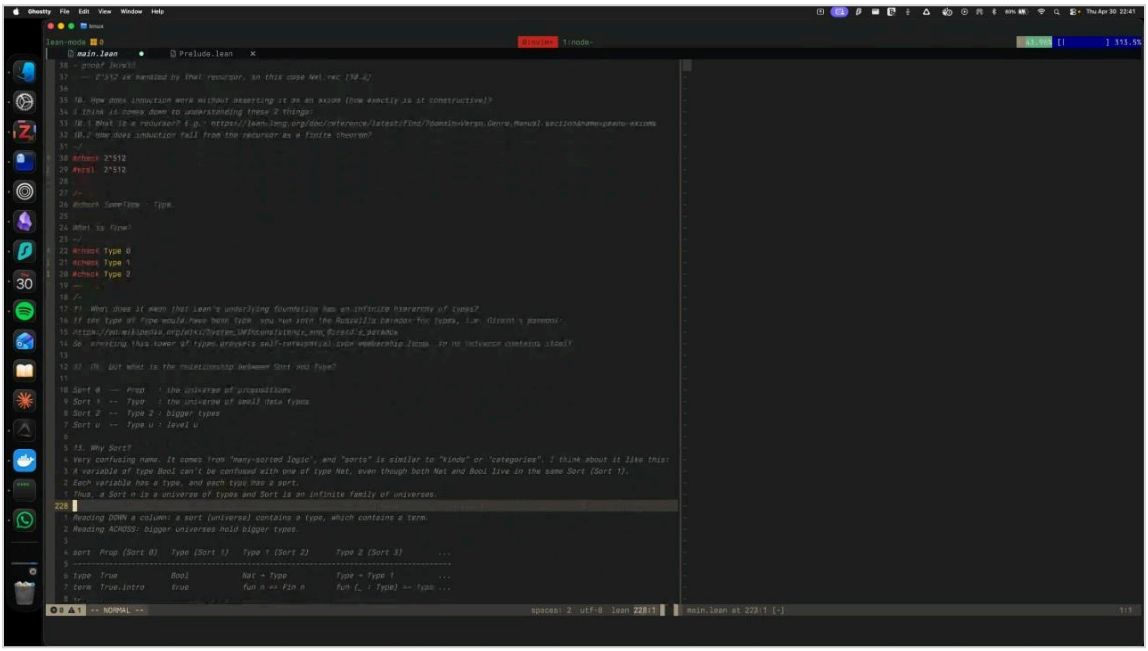
- **Types as First-Class Citizens**: The discussion moved to Section 2.2, confirming that type theory extends simple type theory by making types, such as `Nat` and `bool`, first-class citizens. This means types themselves are objects and must therefore possess a type (01:02:22). Defining a variable that is a type, like a natural type, enables abstractions, allowing the definition of mappings that take one type and return another or take two types and return one, such as the `prod` function (01:03:48).
- **Type Abstraction and Pattern Matching**: Having types as first-class citizens permits abstraction, enabling the definition of complex concepts like `prod` using only mappings and functions. This level of abstraction allows for abstract pattern matching, where one can prove if an unknown mapping, like `y`, matches the signature of a known construct like `prod` (01:05:38). The expectation for a construct `G` defined as a type of type that takes `Nat` and `bool` is that it represents the Cartesian product of `Nat` and `Bool`, which was confirmed using reflexivity (01:07:03).
- **Inductive Definition of Constructors**: The concept of constructors was explored, which are defined using the `inductive` keyword. For example, the `bool` type is defined inductively as a type where `false` and `true` are both booleans. The meaning of `inductive` remains a complex, deep topic that is difficult to fully grasp (01:08:11).



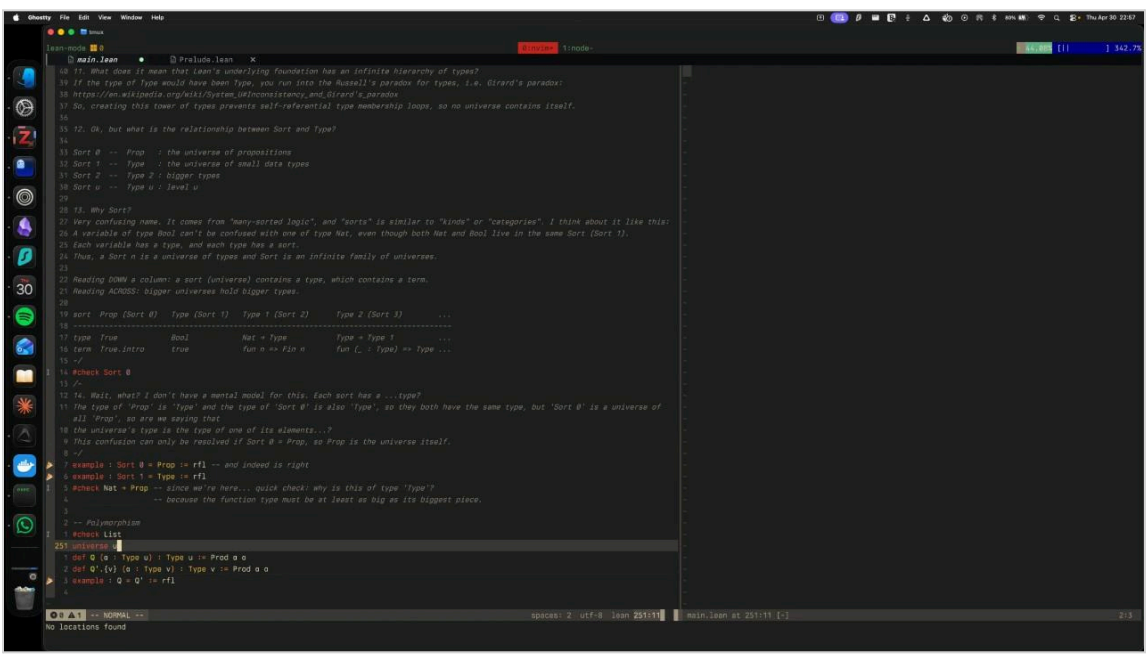
- **Set Theory vs. Type Theory**: A distinction was drawn between set theory and type theory, focusing on the use of constructive mathematics in type theory (01:09:17). Set theory relies on classical logic, including the law of the excluded middle (A or not A), which can lead to paradoxes and proofs of non-existent things, as the logic is not restricted by necessity (01:10:40). Type theory, following the intuitionist framework of constructive mathematics, does not accept the law of the excluded middle and requires palpable evidence for the existence of variables (01:14:18).
- **Constructivism in Programming Languages**: The participants affirmed that individuals using programming languages are operating as constructivists because type theory is founded on constructivist theories. Set theory views the universe as pre-existing, where defining a boolean set is merely labeling pre-existing objects (01:16:56). In contrast, type theory requires the explicit creation of rules, or constructors, to define a type like `bool`, meaning its inhabitants (`false` and `true`) do not exist before the definition is provided (01:18:27).
- **Inductive Definitions and Natural Numbers**: While the difference between set theory and type theory for `bool` is trivial, the distinction is significant for `natural numbers`. In type theory, a natural number is defined by rules using `zero` and `sock` (successor), and all natural numbers are finite constructions built from these two rules (01:20:51). This definition structure, inherent in the `inductive` keyword found in every type definition, leads to the question of whether the proof

by induction works because the type itself is an inductive construction (01:21:51).

- **Physical Representation of Large Numbers**: A question arose concerning how large numbers, such as 2 to the power of 512, are stored and represented internally. Although the concept is logically represented as many applications of the successor function (`sock`) at the proof level, at runtime, the number is represented using the GMP library (01:26:33). The concept of `recursor` is introduced as another complex element related to inductive proofs and the axiom of induction (01:27:50).
- **Hierarchy of Types and the Russell Paradox**: The system has an infinite hierarchy of types, where checking a type (e.g., `type zero`) reveals that it is of the next type in the sequence (`type one`) (01:33:54). This hierarchy is necessary to prevent self-referential type membership loops, such as the equivalent of Russell's paradox in type theory ("the set that cannot contain itself"), thereby ensuring system consistency (01:36:04). This necessity stems from Godel's proof that a system cannot be both complete and consistent, leading to a tradeoff where consistency is prioritized (01:37:04).
- **Sorts, Propositions, and Universes**: Sorts are synonymous with universes, and `sort zero` is the universe of all propositions (`prop`), while `sort one` is the universe of small data types (01:38:28). The hierarchy continues infinitely, with `sort two` being an abstraction over types, creating universes of types of types to maintain consistency and avoid paradoxes (01:41:03). Type and sort are distinguished in that natural and boolean are both types residing in the same sort (universe), yet they are distinct from each other (01:43:36).
- **Distinction Between Propositional and Type Universes**: The difference between "true" (lowercase, a value/term) and "True" (capital, a type/proposition) is that they exist in different universes. The lowercase `true` is a term in `sort one` (type universe), while the capital `True` is a type in `sort zero` (prop universe) (01:46:07). The type of a function or mapping is determined by its highest component type, ensuring that the function type is at least as large as its largest constituent piece (01:51:42).



- **Polymorphism and Universe Abstraction**: Polymorphism is demonstrated with lists, where the list definition uses `U` for the universe, allowing the list to accept any kind of type. Abstracting over universes is achieved by defining a universe keyword, like `U`, and then creating mappings that take types from that universe, effectively abstracting over all existing types (01:52:47). Syntactic sugar is often used in coding to simplify the notation for universe definitions, making the code more readable (01:55:04).



- **Reflecting on the Learning Process and Depth of Study**: Claudiu Leoveanu-Condrei expressed that the learning experience is engaging but noted that splitting just two sub-chapters took them two hours, indicating a lot of material to cover (01:57:30). Andrei Ungureanu commended Claudiu Leoveanu-Condrei on their presentation style, noting they are a good teacher who went through a lot of unexpected material beyond the course itself. Claudiu Leoveanu-Condrei admitted to frequently going down "rabbit holes" during their study, driven by a need to find answers, and finding the decision to stop the exploration very difficult (01:58:40).
- **Concerns Regarding Foundation and Pacing**: Andrei Ungureanu suggested that if there are gaps in understanding at this early stage, it might hinder progress later on, emphasizing the need to distinguish between foundational concepts and less critical "rabbit holes" (02:01:12). Claudiu Leoveanu-Condrei agreed that having a solid foundation is essential and views the entire Chapter 2, which introduces types like "double" and "boolean," as crucial since it is the "backbone of lean" (02:02:31). The goal is to understand the language well enough to be able to control a coding agent, rather than becoming a Lean expert (02:04:54).
- **The Goal of Learning Lean for Agent Control and Verification**: Claudiu Leoveanu-Condrei explained that the purpose of learning Lean is to understand the language sufficiently to debug and oversee code written by an agent, asserting that one needs to know how to code to effectively use agents (02:04:54). Claudiu Leoveanu-Condrei compared using an agent without understanding the code to visiting China without speaking Chinese, stating that without comprehension, one is lost and does not gain much (02:05:59). The ultimate goal is to apply this knowledge to software verification, proving that software is correct, and they referenced a project to formalize the Signal protocol in Lean 4 as an example of desirable work (02:09:06) (02:11:17).
- **Scheduling Change for the Meetings**: Claudiu Leoveanu-Condrei announced that the meeting day must be moved from Thursday, excluding Tuesday, and suggested either Monday or Wednesday as alternatives. The participants agreed to move the recurring meeting to Monday, starting the week well with this work (02:12:30).
- **Planned Chapters and Upcoming Vacation**: Andrei Ungureanu informed Claudiu Leoveanu-Condrei that they will be on vacation from May 18th to May 30th, suggesting a possible pause in meetings during that period (02:13:25). Based on the current date, the next two scheduled meetings would be on the 4th and 11th of the month (02:14:29). Claudiu Leoveanu-Condrei noted that they could aim to

cover sections 2.3 through 2.7, which was recommended by GPT, before the vacation, although they expressed some apprehension about the amount of complex material involved (02:15:49). The participants confirmed they would proceed with the proposed plan for the next meeting and cover as much as possible (02:16:59).

You should review Gemini's notes to make sure they're accurate. Get tips and learn how Gemini takes notes

*How is the quality of **these specific notes?** Take a short survey to let us know your feedback, including how helpful the notes were for your needs.*

📖 Transcript

Apr 30, 2026

{ lean-mode } - Transcript

00:00:02

Claudiu Leoveanu-Condrei: cool. So, we're on session number two. So, we decided to go over um the two subchapters of the of the book um of the chapter two dependent type theory. Let me quickly share my screen. Uh Can you see it?

Andrei Ungureanu: Yes. Yes, it's okay.

Claudiu Leoveanu-Condrei: Okay, cool. So, we decided to go over the dependent type theory uh main chapter, but only the two subchapters. So, simple type theory and types as objects. And um yeah, we kind of went for uh me being the guide. So, let me show you what I did. So basically the way I approach this topic was to go over the code but as usually um as usual it's like it took me sometimes a lot of time because I had questions and what the hell is this and how do you do that and I went oh man I did not like I

Andrei Ungureanu: Exactly. My case.

Claudiu Leoveanu-Condrei: honestly did not so I allocated like you know some hours and then I was

00:01:25

Andrei Ungureanu: Yes.

Claudiu Leoveanu-Condrei: like those hours like went like they just disappeared and then I was like god damn I

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: need more hours then another day came and then those hours went too they

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: just went away uh but yeah it's cool I mean it's so many so many things uh to talk about in my opinion even here at this at this early stages so I'm really curious like what what the hell is going to be like for the tactics and all of that I'm super curious because I only know like the terms I only know like some tactics but uh I actually let me check because I didn't check. Did they add grind here? Okay, they didn't.

Andrei Ungureanu: I I don't know what tactics is actually.

Claudiu Leoveanu-Condrei: So there are some so so tactics is just a way

Andrei Ungureanu: So,

Claudiu Leoveanu-Condrei: to um solve to prove something. So it's it's just like uh I don't know reflexivity or you apply certain

Andrei Ungureanu: okay.

Claudiu Leoveanu-Condrei: theorems and you just prove something prove some some some statement.

00:02:24

Claudiu Leoveanu-Condrei: Um, but I don't think they added grind anywhere.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: Yeah. Okay. So, so grind is just like a way uh this is more recently but it's just like a way to it's like a I don't know how to describe it's like an not an exhaustive but it's like there were before having grind you just had to do a lot of manual you know programming which they now obstructed to that tactic which you just say grind and then it's like it's literally in the name it's like grind put the computer like go ahead do it for me I don't care about writing all of this stuff. Uh but it's not in this book. So they didn't they didn't update that.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: But anyways, uh getting here. So let me switch to new. Is this big enough? I don't know. Something like this.

Andrei Ungureanu: Oh, yeah. Yeah.

Claudiu Leoveanu-Condrei: Does it work?

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: Okay.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: So um let's go over some of the basics from for me as well.

00:03:20

Claudiu Leoveanu-Condrei: So uh this panel on to the right. Uh let me quickly kill this thing. I don't know if you see it, but for me it's annoying. This you are sharing the screen. I hope it went away. Okay. So this this uh this panel on the right is called the info view and it's it's just a

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: way to u show a lot of details um about the types about um basically extra information and metadata and they have some uh I actually tried to play a bit with it but I could not make it work uh easily so I just uh didn't do it because I didn't have time but there are some something they added recently which is called widgets and what's nice about widget widgets is that you can literally um I don't know you can add images or you can add uh visualizations and then they would appear to the right as you uh most of the code they they will appear to the right. So um yeah I I will I will use this info view a lot as we go but

00:04:25

Andrei Ungureanu: Yeah, actually I use this as well.

Claudiu Leoveanu-Condrei: in Okay,

Andrei Ungureanu: I mean I installed the Neo Vin for uh the

Claudiu Leoveanu-Condrei: nice.

Andrei Ungureanu: extension.

Claudiu Leoveanu-Condrei: So you're also a neoim

Andrei Ungureanu: Yeah. Uh yeah,

Claudiu Leoveanu-Condrei: guy.

Andrei Ungureanu: depends on the task. Actually, I switch between neim or zed

Claudiu Leoveanu-Condrei: Oh, I actually have that. It's it's my it's my editor,

Andrei Ungureanu: that.

Claudiu Leoveanu-Condrei: but only for SSH I'm using it.

Andrei Ungureanu: Oh, okay.

Claudiu Leoveanu-Condrei: Yeah, cool. So uh on the neo like uh that there is this uh yeah I probably you saw it but there

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: is there are there is this page with shortcuts. So if I press if I press uh shift K so it's capital K. Uh usually it's showing uh I think the same thing that it's it's it's showing to the to the right. So if I go here and press shift K uh not all the time. There are some things which are not the same. I think here they are not the same but usually it's uh it's just like a way to peek into the uh comment section.

00:05:28

Claudiu Leoveanu-Condrei: So if we go to true um basically it's this comment from here is just

Andrei Ungureanu: Sorry to interrupt. I have a question here.

Claudiu Leoveanu-Condrei: uh

Andrei Ungureanu: So my understanding is that the window on the right is kind of an interpreter right of where

Claudiu Leoveanu-Condrei: Yeah. Yeah.

Andrei Ungureanu: you with the cursor in the left

Claudiu Leoveanu-Condrei: Exactly. Yeah. Yeah.

Andrei Ungureanu: screen.

Claudiu Leoveanu-Condrei: Exactly. Yes. It's it's and it's what's nice is that it it moves as you move through the uh lines of the left but not only the lines I think it's it's based on the you see that it's I don't think it okay I can so you see it's the range it's also the range

Andrei Ungureanu: Yeah, it show shows the right.

Claudiu Leoveanu-Condrei: so it's also line and character to the

Andrei Ungureanu: So, it's kind of it selects the word you're on

Claudiu Leoveanu-Condrei: left yeah and I guess it checks

Andrei Ungureanu: basically.

Claudiu Leoveanu-Condrei: it if it's like something lean related Because for instance, this is not uh but uh n is and all all these

00:06:20

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: other things they just are uh anyways I use this a lot. So I I use it to peek into things. I just you know I didn't know for instance that uh colon equal is uh is showing me anything or not. So I just press and say okay it doesn't show me anything. So I guess it's just like the syntax more than anything else. Uh I hate that. Okay, fixed. I'm a bit OCD. So,

Andrei Ungureanu: Yeah, I know.

Claudiu Leoveanu-Condrei: sorry about that.

Andrei Ungureanu: I know the feeling.

Claudiu Leoveanu-Condrei: Okay, so good. So, basically this is just like I would say this is very basic. Uh there are some questions here which I still didn't have didn't manage to answer them because I just went on and on and on and then some questions started to pop up and I was like okay but what is that? What what the hell is this? So, uh yeah, this is I guess this is very basic. is just a way to define uh variables um and uh assign them some values.

00:07:22

Claudiu Leoveanu-Condrei: Um what's uh let me see what I got here. Why does replacing bull with props still hold? Uh yeah, this is the question I still don't fully understand. I think maybe it is because uh bull actually no it's not because so prop is a proposition.

Andrei Ungureanu: But what what is prop?

Claudiu Leoveanu-Condrei: So I I I will talk a bit down about it but the point is that

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: crop and bull they are they live in different type universes. I I will show a will be down because that was one rabbit hole I went into. But they basically live at different um in different universes in different type

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: universes. But I don't know exact. So this I know but I don't know why it still holds because for instance if I take here and I put prop you see there's no error but it's this weird type which is a sort.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: And again, this is another thing I went really into the into the weeds about what the hell is sword and it was a huge confusion for me for me, but it's going to be down uh at some point, but this is a sword basically.

00:08:31

Claudiu Leoveanu-Condrei: So, it's a universe of um yeah, it's a universe. So, I don't I don't understand why why these two are I don't think they are the same. So if we do something like eval let's say t um oops like this t equals uh let's say f let me make this true uh yeah so so these are not equal wait what did I mess up unexpected token oh because it's not a val it's example Ah,

Andrei Ungureanu: So

Claudiu Leoveanu-Condrei: you see it is the same thing. So I don't get why why these two things are the same

Andrei Ungureanu: just to be clear the question here is that uh B and P are from different universities but

Claudiu Leoveanu-Condrei: thing.

Andrei Ungureanu: they are interchangeable right that's the issue

Claudiu Leoveanu-Condrei: Yes. So so the way this works for instance what we wrote here so we wrote an example in

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: which we said that basically t is equal with f. So basically these two uh variables they are equal in this case but and and we basically prove this by reflexivity.

00:09:47

Claudiu Leoveanu-Condrei: So you know reflexivity like in in u in mathematics is just a way to

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: yeah so is that RFL is called reflexivity. So when you call this RFL basically lean then goes and try to prove this um um try to prove this thing uh underneath is literally using the theorem prover to prove that c equals f. So this statement right here is basically our first proof if you want.

Andrei Ungureanu: We're

Claudiu Leoveanu-Condrei: Um so but again I don't know why this is true like uh conceptually

Andrei Ungureanu: good.

Claudiu Leoveanu-Condrei: I don't understand it why is this true? So for instance, if I'm making this false, this is no longer true. Which I guess it's I expected it. But if we make this capital true, this is no longer true. So I I don't get exactly why. So basically this is I think this is a type right now. So this is Yeah, this is a type. So this is a type and this is a value. So capital true is a type and capital T is a value.

00:10:48

Claudiu Leoveanu-Condrei: Uh but this this is true. This is equal. And I don't I don't exactly understand why they are the same things. But they are the same thing here. So that's worth uh pointing out because it's something that I uh I this is the order of the question that I have. So you will see how my mind works. They have no I don't know why I asked myself this at this point. I have no idea. I don't remember why I asked myself but

asked myself this. Uh then what is the second thing that I wrote here? What if you instead write def crop? Um so basically what I what I said above. Yeah. So I basically did this. Uh I think what this means is that you're basically saying that in this case f is is of type prop and it's the same type as true. So f in this case is the type true. And I don't know if we can do like something like check f.

00:11:47

Claudiu Leoveanu-Condrei: Yeah. So it's a prop as you can see it's a it's it's a type of prop and I don't know if we write true like this yeah is the same thing. So my assumption is that we we basically made the um the f like a type in this case and the convention I think is to make them capital F. Uh if I if I'm if I remember correctly that's just a convention of how to write these things but I'm not yet sure. So let's move on. from this. Uh I have no idea if this is correct. That that's why these are actually unsolved for me. Let me put

Andrei Ungureanu: You assign those at the time of the

Claudiu Leoveanu-Condrei: uh I hear you. I hear you very badly.

Andrei Ungureanu: possible

Claudiu Leoveanu-Condrei: I don't know if it's from my side. You're you're cutting previously it was fine.

Andrei Ungureanu: Maybe now is it better?

Claudiu Leoveanu-Condrei: Yeah. Now, now it's better.

Andrei Ungureanu: Uh I was asking why how can you assign both a type and a value to the

00:12:52

Claudiu Leoveanu-Condrei: Yeah.

Andrei Ungureanu: variable? Uh let's say you had to do it with capital T or false or whatever you put capital and it Yeah.

Claudiu Leoveanu-Condrei: Like this. Yeah. So you see even this works. So basically you said the bull is equal. It's like it's it's true. So this this works because um it's very important to understand that in lean that types are first class uh objects. So basically you can return type, you can manipulate type, everything variables can be type, you can pass to the function types like literally types without any values like you pass to the function a type and that's it. and then that that function operates with type let's say true uh in in the domain or in in its code domain. So that's why you can replace and do this kind of stuff. That's if you remember last time I said that if you're trying to do that game that they are showing it's very it's going to be mindbending at first because you have to it's you'll be like what what is there?

00:14:03

Claudiu Leoveanu-Condrei: What do you mean?

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: Like it's it's a it's a you're hitting that right now.

Andrei Ungureanu: Exactly.

Claudiu Leoveanu-Condrei: It's you you have to uh you have to keep in mind all the time that that uh uh types are first class in this in this uh in this language. So that's why you can manipulate them like this. You you can you basically say that the this type is equal to this type. So that's why you can do this kind of manipulations. But for instance here I cannot do I cannot say one. You see this doesn't work. This I'm not allowed to do.

Andrei Ungureanu: Yes.

Claudiu Leoveanu-Condrei: But what I can do is say oh actually I cannot do this but I can say this. So this these two are the same thing because if we go into bull um I mean sorry if we go into true. So true is just like it's just the proposition where the intro is true. So I I don't yet ex understand what exactly does this mean.

00:15:04

Claudiu Leoveanu-Condrei: Uh this is the case you will see because I split it down. I had the same question uh what is where and what is this? Basically this is just like a case enumeration but I don't know exactly what is this intro. this intro came up a lot uh in in in my rabbit holes but I don't exactly know I think it's something um yeah it's something about the way they the question that I had it's about um some conventions or some sort of like you know the way they baked the logic into it and this is one of them another one is MK I don't know if it's anywhere here yeah this thing which is make so you're making a a class constructor in this But I I will show this uh in a I will touch them. But again you see here I can do this but I cannot say that the bull is natural

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: number this I cannot do it. Uh but I can say that it's false or I can say that it's true.

00:16:01

Claudiu Leoveanu-Condrei: This I'm allowed to do because I'm basically saying that I think I'm not sure right. I think that you're saying here. So if it's like this then I'm saying that t has the value through. So t is the variable. So if I'm checking, see here. Oops. See? Yeah, it's a bull. So if I'm invalid, it's true. You see? But if I'm doing this, it's still true because I think it's just this capital true.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: I don't know exactly. I think they have some like um how you call it like coercion. These are coercion things like I I always hate this.

Andrei Ungureanu: So it drops a true value being a true

Claudiu Leoveanu-Condrei: Yeah, exactly. So I I always hated I always hated smart defaults like because I don't get them like I don't understand like you I mean you you are a programmer right so when you make smart defaults you have some idea why you're making something like that default right so if you assume that the programmer of your thing or somebody that uses your thing understands why you made that small default uh but here I don't get it I don't get why is this a default I don't That's I at least I don't get it,

00:17:14

Claudiu Leoveanu-Condrei: but it's something to keep in mind. So that's uh that's my point. Okay.

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: Then uh yeah, we have the check um command which just check things. There's nothing fancy here. It's just showing you the types of things. Um yeah, this is basically boolean ends. This is like uh uh uh this is I I tested this though I was curious because this is boolean end and this is like logical end. This is like apparently this is used in propositions.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: Uh so yeah I I I say I even asked myself here why and and or turns the information into a prop. Uh why is f equal true? So again I'm not sure exactly how these things work underneath. I I'm I'm pretty sure that they they only make it like logical. So I assume when you try to prove something, you won't be using this stuff. You would be using this because if you're

Andrei Ungureanu: Yeah, that's what the first example in chapter one uses

00:18:17

Claudiu Leoveanu-Condrei: having

Andrei Ungureanu: this to prove that T and F is equal to F and T.

Claudiu Leoveanu-Condrei: Okay. I uh you mean the logical end, right? Or or yeah,

Andrei Ungureanu: Yeah, the

Claudiu Leoveanu-Condrei: I I I assume like that's the case because it it automatically turns it into a prop.

Andrei Ungureanu: logical

Claudiu Leoveanu-Condrei: And prop is again it's a proposition. You're making it a proposition and you will see the ch the universes of um of types how they work because it's like a tower of of things. Okay. But I this is like nothing fancy here. Uh this is to me yet uh not answered. I don't know what exactly happens here. Uh this is yeah again

check equals true. So basically uh you see that if I'm checking the type of true then it's saying that it's a boolean. But if I'm checking the type of this guy it's saying that it's a prop. So basically what this is telling me is that this should be used with propositions.

00:19:19

Claudiu Leoveanu-Condrei: That's how I understand it. So if you if you would be trying to write some sort of like proof, you wouldn't use um or not not pro but like I mean you can use proofs and and use variables like true but my point is that when you write something over let's say so your domain is types or something that's doesn't have a value, right? It's it's like saying for all x's, right? So when you say for all x's, I'm not sure you can use like uh lowerase true. It's like because that's not that's not operating over type, that's operating over variables. You're just you're basically saying there is an x that is true,

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: a value that is true. But if you use capital true then you say I would assume like you're looking over all instances of X where this thing would be true like whatever condition you have and in that kind of like language if you phrase it like that then you would use capital T but I'm not sure if that's the case.

00:20:20

Andrei Ungureanu: But one question here why isn't true to capital to the

Claudiu Leoveanu-Condrei: Uh

Andrei Ungureanu: type I mean how do you

Claudiu Leoveanu-Condrei: so everything everything it's a type here literally everything you see.

Andrei Ungureanu: differentiate

Claudiu Leoveanu-Condrei: So if we check one this is a type you

Andrei Ungureanu: so all right So basically uh

Claudiu Leoveanu-Condrei: see

Andrei Ungureanu: uh what we have there prop is like a sub case of the type.

Claudiu Leoveanu-Condrei: uh you mean here oh sorry

Andrei Ungureanu: No no like line 22 we had prop there uh

Claudiu Leoveanu-Condrei: because 11 ah okay so

Andrei Ungureanu: like 11 now.

Claudiu Leoveanu-Condrei: here

Andrei Ungureanu: Yeah. So this prop is a subset of type.

Claudiu Leoveanu-Condrei: Right.

Andrei Ungureanu: Basically the type is like the um main everything is the type right.

Claudiu Leoveanu-Condrei: Yeah.

Andrei Ungureanu: So yeah I think I

Claudiu Leoveanu-Condrei: Every everything it's a type. Yes. So yeah everything it's a type.

Andrei Ungureanu: get

Claudiu Leoveanu-Condrei: If you you see we check one is telling us that it's uh so this basically this notation here means one in this case it's a variable is of type not this is saying that true is of type prop and and so on and you see this is the difference between these two for instance it's like a true this is a boolean true uh so yeah and and it's a boolean so basically it's like you're indexing like It's this boolean has a membership u

00:21:55

Claudiu Leoveanu-Condrei: like condition like you have dot through and if we go indeed into boolean I think we can see it you see you have two cases let's not call it number but you have cases and they even write here you see the boolean value false not to be confused with the

Andrei Ungureanu: Very

Claudiu Leoveanu-Condrei: proposition false so that's my point I I think that when you write capital

Andrei Ungureanu: nice.

Claudiu Leoveanu-Condrei: false with f capital false then it means that the proposition so it's like writing P is false right you're not saying that P

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: has the value uh false you're just saying P is false there's no 01 there's no checking there's nothing like that that's my assumption I think that's that's the

Andrei Ungureanu: Right.

Claudiu Leoveanu-Condrei: case okay so let's move on uh yeah eval is straightforward as I said you can eval whatever you want I actually have a case below which is insane And you can do like I don't know, man. Look at this.

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: That's insane,

Andrei Ungureanu: Never tried anything like this here.

Claudiu Leoveanu-Condrei: right?

00:23:00

Claudiu Leoveanu-Condrei: Up until this point is starting to process. Like you can do two to the 5 9th in like under

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: one second, which is Yeah. I mean,

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: it's it's insane. Okay, then.

Andrei Ungureanu: They know uh in what language uh is it

Claudiu Leoveanu-Condrei: Yeah,

Andrei Ungureanu: written?

Claudiu Leoveanu-Condrei: it's a GMP is the new library GMP lib. I actually even have it below because I was curious too about the same thing. They are using the the big integers from from GMP. Okay, so let's see what I wrote here. What makes type theory powerful is that you can build new types out of the others. Exactly. So this is uh this I took from the the book. Um then I found this stuff. Uh the compiler is messy. If there's

nothing but comments following a type, I get no error. But adding a new line with anything makes the error visible. So let let me explain what this means.

00:24:01

Claudiu Leoveanu-Condrei: So uh I don't know. Let me copy this. So I I've noticed this stuff. So if I do this, you see there's no error, right? No, nothing in Christ. If I go below,

Andrei Ungureanu: Yes.

Claudiu Leoveanu-Condrei: there's no error. I can put a comment there. There's no error. But if I try to write anything, I don't know oneplus 1, you see that suddenly there's an error and it says that unexpected token expected something something. So basically uh yeah this continues on a new line. So you can actually say um yeah you can do something like this on a new line. So you don't have to do it here. You can do it um yeah you can do it below. But I didn't know. So I found it like this. And then I went uh into the rabbit hole obviously because then I asked myself what the hell is

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: uh yeah what is like where and what is this guy?

00:24:56

Claudiu Leoveanu-Condrei: Um then I had another question which is how do you default values and yeah and then I had the other one what exactly are war and um and those things.

Andrei Ungureanu: Right.

Claudiu Leoveanu-Condrei: Yeah four and five. So I had this four these two questions here. Then I took the cases. So I I dig a little bit into uh this uh or sorry into this imply. So basically every time you write um I don't know something like this like like this you're making a function. So this is a mapping. Every time you're using this guy you're creating a mapping. You're creating a mapping right now from the natural type to integers in this

case. So always keep in mind that P1 is um is a function in this case. is taking a natural number and it's it's returning an integer.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: So I again I was looking into okay how do you set defaults? So basically um apparently there is something called default which is um implemented for most of the of the functions as I wrote here it's a canonical way to make a default it's always like defaulting to the zero equivalent.

00:26:11

Claudiu Leoveanu-Condrei: So there are I don't know I don't know what's the zero equivalent for a specific type but that there could be they already have implemented some defaults so that you can play with some real variables you know if you want for instance to say uh I don't know that the for any x or like for every x something like that then you want to say okay but like let's see what if x is the default value right and you're just calling it x a default they offer you this possibility and then you just see exactly like okay can I work with this default or not do I need more assumptions or is def sometimes the default is enough right it's it's enough to check something uh but I don't know how far this extends I'm not sure exactly like how long how many um

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: mappings or how many how many types because this is this is also a type by the way I'm not I'm not exactly sure like how many types um have a default but in this case uh so we're looking at imply Okay.

00:27:11

Claudiu Leoveanu-Condrei: Or two. This is the default. It's basically zero in this case. Uh and it's always taking the uh the default of the return type. So in this case of the int. Uh then you can do anonymous lambda or lambda functions. This is like a notation for how to do uh anonymous functions. So you just uh yeah put a function of n and then you return the um this is like a way to call the method. So every I don't know if you saw that

probably saw but everything uh why doesn't it complete no okay not okay so I guess it doesn't show but it the point is that every um every uh type has some functionality some methods that you can apply and in this case uh yeah you just call that int is of natural n and this this is

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: basically making an anonymous function. This right here is again syntactic sugar for lambdas is just is the same these two things are the same thing. You just can write it uh like this or you can write it like this.

00:28:26

Claudiu Leoveanu-Condrei: So basically you say here that uh P is of type of type natural. So you can do this kind of stuff. You can put also a type that you want here um to the uh like to an argument and

Andrei Ungureanu: But you also have it uh for P2,

Claudiu Leoveanu-Condrei: then

Andrei Ungureanu: right? It's the same. I mean,

Claudiu Leoveanu-Condrei: Yes.

Andrei Ungureanu: and you don't use it afterwards.

Claudiu Leoveanu-Condrei: Yeah. Yeah. You drop it here.

Andrei Ungureanu: What's the point of using Oh, you drop.

Claudiu Leoveanu-Condrei: Yeah.

Andrei Ungureanu: You understand?

Claudiu Leoveanu-Condrei: Yeah. So,

Andrei Ungureanu: Doesn't matter what you do,

Claudiu Leoveanu-Condrei: so that's why that's why they are the same thing. Okay. Then we have a constant function. So,

Andrei Ungureanu: right?

Claudiu Leoveanu-Condrei: basically it's from natch to int and then again we drop it, right? We we discard the thing and we just return in this case minus one. Then there is

this idea of eta reduce or eta expansion. So at reduce is basically um like you're not adding any value at all.

00:29:21

Claudiu Leoveanu-Condrei: So there's no n and there's no default, right? The default is a type. So the default is int is of natural not. That's what you're saying here. You're you're not putting any number or anything. Right? So as you can see the difference here is that you said well I have this type right? Uh but it's n.

Andrei Ungureanu: value.

Claudiu Leoveanu-Condrei: Yeah, it's it's an it's a variable, not value. It's it's a variable n,

Andrei Ungureanu: Yeah,

Claudiu Leoveanu-Condrei: some variable that we have. It's gonna who knows what values it takes,

Andrei Ungureanu: there

Claudiu Leoveanu-Condrei: but we know for sure it's going to take a natural uh a natural n. It's going to be here. We're saying, well, I don't care. It's just minus one. It's constant. Here, we're basically not saying anything whatsoever. We're just saying that um yeah, it's a it's this is what we expect. Something like this, something of this shape. We don't know if it's an n.

00:30:18

Claudiu Leoveanu-Condrei: We don't know if it's a function maybe that returns something. We just know it has to have this uh shape or form. Um

Andrei Ungureanu: This would be like a sort of uh generic maybe. Is that how you could see it

Claudiu Leoveanu-Condrei: yeah maybe yeah exactly so basically

Andrei Ungureanu: conceptually?

Claudiu Leoveanu-Condrei: I wrote here uh so at aa reduce um iie the n doesn't appear anywhere. So P2 is ETA expanded in this view but P3 is not because it discards with um U with the underscore its input when in fact it should forward it. So the difference is that basically P2 here you see the N. So basically we propagate the end right we propagating means you're moving it from

Andrei Ungureanu: Yes.

Claudiu Leoveanu-Condrei: here to the other side. Um but in case of uh of P3 it's not ETA at expanded even though these two are the same. This is ETA expanded but this is not at expanded because here you're not carrying anything.

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: You're just dropping it and you're basically saying that I'm expecting some n here and that's it.

00:31:32

Claudiu Leoveanu-Condrei: That's all you all you're saying. Uh again weird stuff. I know I I still didn't wrap my hand around all all of this stuff, but for now I I'm just taking it like syntax and weird stuff that you have to probably remember for a while. Um then what do we have here? So yeah, this is sorry. Sorry is going to be like a very useful um placeholder. So this is just a way so when when we will be writing proofs we will be using a lot of stories because it's like so sorry it's like

Andrei Ungureanu: You have to do macro or something,

Claudiu Leoveanu-Condrei: um yeah exactly so it's like you're laying down the

Andrei Ungureanu: right?

Claudiu Leoveanu-Condrei: structure of the proof but you're not feeling the the details in you're just saying uh I think this proof might look like this so I'm going to use this and this and this and you're just feeling with stories you don't know what exactly is going to you know what the cases will be or how it's going to look like.

00:32:29

Claudiu Leoveanu-Condrei: And then the basically the the uh interpreter here to the right, it's fine. It's not crying or screaming or anything back at you because as as we saw, if I'm doing this, it's crying, right? It doesn't like this stuff. I just I showed before how I found it out. Uh but you put sorry, then it's like, okay, fine. I'm gonna believe you.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: But if you try to run the script, then it's going to obviously compile. uh when it when it sees sorry it it's it crashes. Okay, cool. So this is just the two then uh this is the cartisian product and here again you have defaults u the the the default for this it's just 0 0 um this is how you basically write the equivalent of anonymous uh in tpple syntax. top you're just putting like a default like this. Um, you can also use uh this constructor which again is that MK that I I I I said before which is like to me it's like an intro.

00:33:29

Claudiu Leoveanu-Condrei: It's something similar to that but at least MK I kind of got it. So MK is um it's it's basically the same thing as this but MK is like it's like the init of Python or like uh you know what I mean? It's it's like a way to initialize like your constructor like how to make um like

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: a structure if you want. I even give an give an example here who

Andrei Ungureanu: Go as this has make as well.

Claudiu Leoveanu-Condrei: sorry ah okay then there you go so it's

Andrei Ungureanu: Go go go L.

Claudiu Leoveanu-Condrei: I assume it's like that I even gave this example here where you make a structure person where name is a string and age is a natural number and then um yeah mk is a convention as far as I understand this can be anything so you don't have to write it with mk mk it can Go whatever. It doesn't have to be.

Andrei Ungureanu: Yeah, it's like new in Rust,

Claudiu Leoveanu-Condrei: This is still going to work.

Andrei Ungureanu: you know, the convention is to use new.

00:34:30

Claudiu Leoveanu-Condrei: Exactly. Yeah. Exactly.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: That's the best example. Yes. It's like the new in ROS. Yes. So, this is the new in ROS. This is what I this is how I understood it at least. Not sure but I think this is this is right.

Andrei Ungureanu: Mhm. Yeah,

Claudiu Leoveanu-Condrei: Uh

Andrei Ungureanu: this is a great way to view it because you have to like build your mental model around

Claudiu Leoveanu-Condrei: then

Andrei Ungureanu: stuff you already know. So when you uh you do stuff with lean,

Claudiu Leoveanu-Condrei: yes,

Andrei Ungureanu: uh it probably will make more sense. And like um

Claudiu Leoveanu-Condrei: I agree. Yeah,

Andrei Ungureanu: yeah.

Claudiu Leoveanu-Condrei: unfortunately like if you if you if you play with dependent type theory lang with dependent type languages, it's like there's not not much out there, you know, that it's fitting the the the Yeah,

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: the I I I think I even use this word mental.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: And there you go.

00:35:20

Claudiu Leoveanu-Condrei: You see, I even use it myself below.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: But uh yeah, it's it's it's there are some things which are you will see below.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: I I really got very confused and I and I'm not sure I'm yet out of the confusion there, but it was very fun. Okay. Then apparently you have you can do anonymous constructor brackets which is just like um by the way this is uh we have to you have to do it with the the uh so it's not this you you know what I mean it's it's

Andrei Ungureanu: Yeah. Yeah. I saw that you you use for uh freak with

Claudiu Leoveanu-Condrei: this yeah

Andrei Ungureanu: letters. You also use that symbol before.

Claudiu Leoveanu-Condrei: this is this is like the vector uh inner product thingy parenthesis th those

Andrei Ungureanu: Yes.

Claudiu Leoveanu-Condrei: kind of stuff.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: So, uh, yeah, by the way, I like Lynn so much. Uh, so like it's amazing to even take notes in Lynn. I mean, I love this that you can I I I would like every language to have this stuff to to have this kind of interpreter like just do this s***.

00:36:16

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: I love it. Like it's amazing. So, it's like like you just can take notes in LI and you just kind of render nicely and it's

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: amazing. Uh, okay. So, moving on. Uh then again on on on

Andrei Ungureanu: Sorry, sorry,

Claudiu Leoveanu-Condrei: on

Andrei Ungureanu: off topic question. Have you used the latex like you know it use it

Claudiu Leoveanu-Condrei: uh yeah I even have latex.

Andrei Ungureanu: right?

Claudiu Leoveanu-Condrei: I even have latex. Yeah. I mean I I I had to use it because I

Andrei Ungureanu: I mean you have a lot of symbols there as well. So papers.

Claudiu Leoveanu-Condrei: have I have papers. Yeah.

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: So I had to use it.

Andrei Ungureanu: Okay. Okay. Nice.

Claudiu Leoveanu-Condrei: But yeah I I I have a Vim plugin as well for Latex. So I'm using it locally. But some I use overly for collaboration but when I do stuff myself I I'm using uh delin locally with the neo extension.

00:37:11

Claudiu Leoveanu-Condrei: Okay. Uh cool. So then getting back to the point five which was um what exactly are were and uh like the the bar pipe. Yeah. So uh so there are two different ways of getting a body. So the body of a function if you want. So this is just to define one expression. So there's you're not allowed to have cases. It's like basically all the above. Yeah, exactly. All the examples above are uh just a way to describe to describe one expression. Then this is pattern matching and this is like I'm curious when we will use it in more but this is like very clever because it does very smart pattern matching even on the uh type level and it knows how to de like how to match like very weird patterns. I don't know exactly how they did it but it's very interesting. So here this is just so the pipe is just pattern matching. You just say that the length of the list is a natural number uh and then if it's empty the list uh then you return zero and if it's like again this is lambda uh and and you're saying uh so basically you discard the again this was yet another rabbit hole for me.

00:38:27

Claudiu Leoveanu-Condrei: What the hell is this? What is this man? I was like what is this? So, and it's just a way to to slice a list. This is sugar sugar syntax for slicing a list in in ling. So, this is the head and this is the tail. And basically, you're discarding the head and you're uh keeping the tail and you're just adding in this case one

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: plus the length of xs after you sliced it. So, you just return the tail of it. And I even showed it here. Um so yeah this double column stands for head scale. Uh so in this head we discard the head and keep the rest. And uh very important here I just noticed that this is right associative. So basically you solve it from uh as you can see here. So if you have multiple operations uh you just bracket them from right to left you know it's not from left to right. Uh I think which one was this like matrices?

00:39:23

Claudiu Leoveanu-Condrei: I think it's the same when you when you multiply. Yeah. Yeah. Matrices is the same. I think I think you're again dropping or

Andrei Ungureanu: Yeah. Sorry.

Claudiu Leoveanu-Condrei: something.

Andrei Ungureanu: Uh I don't know. Maybe now better I turn the volume. Uh maybe uh so what does

Claudiu Leoveanu-Condrei: Mhm. No, it's fine.

Andrei Ungureanu: that uh line do? Like three now. Uh upper three.

Claudiu Leoveanu-Condrei: Yeah. Yeah. So this does basically here I put an example.

Andrei Ungureanu: So all right.

Claudiu Leoveanu-Condrei: So we have the inputs right? So the input is matches one because the list is empty.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: So it matches this case and it turns zero. And let's say we have this we have one two three right? So it matches now uh the second row the second thing and it just split it right. So it takes one, two, three, right?

Andrei Ungureanu: All

Claudiu Leoveanu-Condrei: So that so then we drop the one because it's the underscore.

00:40:22

Andrei Ungureanu: right.

Claudiu Leoveanu-Condrei: So it's the wild card. You just drop it. Then you just have the xs is two three round and you return one plus the I don't know why it stopped here, but it's basically Yeah,

Andrei Ungureanu: on plus two.

Claudiu Leoveanu-Condrei: exactly. So you have actually you can put like like this.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: So yeah, this is basically what's happening there. Uh the above uh again this is something that I read. So basically the hold this thing from here is the equivalent of this stuff. But you see it's again it's in a very compiled form because here you have again this. So you can always use apparently this guy all the time. you can use it. But then if you use this, you have to use match, right? And then you have to match access with this stuff exactly as it was before. But basically, you can drop these guys and you just keep it like this. And apparently this is the same stuff.

00:41:30

Claudiu Leoveanu-Condrei: So

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: yeah,

Andrei Ungureanu: So, which one is in tactic sugar? Which one is like the

Claudiu Leoveanu-Condrei: this one is syntactic sugar because you don't you don't have this uh all this

Andrei Ungureanu: the

Claudiu Leoveanu-Condrei: stuff, right?

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: Okay. Then the where is um the wear is just like creating uh local helpers like local functions if you want in the scope in the body of the function of the main function. So here I made like a sum two uh and then you have uh it takes an n which is a natural and returns a natural and then we define like a go which is an takes takes the n and a zero like a counter if you want and then you say where you because you put it here right so then you say okay what the hell is go then you define where go is this guy so go is a natural to a natural to a natural again this I I had a like What what what exactly does this mean?

00:42:30

Claudiu Leoveanu-Condrei: So this is basic all the time when you see so if you would have been poor here right

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: one more then you always have to keep in mind again do do it from uh do it from right to left always this this the last one is the outputs right then the first are inputs that's how you should think about it so in this case note that we match against two inputs but in this

Andrei Ungureanu: Yes.

Claudiu Leoveanu-Condrei: case only the first does real work, right? So why only the first does real work? Because the second is the same, right? So in this case, so basically we're saying here we're defining where go is a natural to a natural. Sorry, it's a natural natural and it returns a natural. That's just a way to I I don't actually like this syntax very much. I would say prefer maybe something like this. uh net maybe or something like this. I don't know. Uh would have been more intuitive for me.

00:43:35

Claudiu Leoveanu-Condrei: But yeah, whatever. Uh oops. Sorry.

Andrei Ungureanu: It's the same issue because it's pretty uh weird like

Claudiu Leoveanu-Condrei: Uh

Andrei Ungureanu: to think how the the parentheses go and there were some cases where

Claudiu Leoveanu-Condrei: yeah.

Andrei Ungureanu: Um,

Claudiu Leoveanu-Condrei: Yeah, maybe. But in this case, basically we're defining the where sorry we're defining the go as being this guy. So we have two inputs and return is the a natural. So we're we're then just pattern matching with the type.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: So basically we're saying zero and uh acc. So you the accumulator. So when it's zero, this is a recursive function. Basically when it's zero, you return ACC. When it's k + one ACC, then you do recussion. you do the go kc plus k plus on right so in this case as I wrote on the right basically only so only this first input does real pattern matching there's no this is the same guy right so basically what lean does is like it's lazy in this way it doesn't if it sees this the same it's just like I this doesn't exist anymore it just dropped drops it it doesn't exist and and then it looks only add zero and k +

00:44:56

Andrei Ungureanu: Um I'm

Claudiu Leoveanu-Condrei: one. So basically the pattern matching

Andrei Ungureanu: not sure what you mean here. Yeah. Yeah, I understand that.

Claudiu Leoveanu-Condrei: right

Andrei Ungureanu: But uh the last statement about ACC, I didn't get it. Why? Why is it dropped?

Claudiu Leoveanu-Condrei: not but it's like um it's lazy in the sense it's it's not loaded if it's so we're doing pattern matching here right so for pattern matching to make sense you're you're basically loading

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: only what you need So I don't know how it does this because this this looks like look ahead right. So you see that this this input differs but this doesn't.

Andrei Ungureanu: Yes.

Claudiu Leoveanu-Condrei: So in this case the only contribution to your cases. So basically what's the difference between case one and case two? It's the first input. That's the thing. This the the second input doesn't matter because it's invariant.

Andrei Ungureanu: Yes.

Claudiu Leoveanu-Condrei: So I I I think there is some efficiency underneath and some lazy way of evaluating this stuff but what I know from what I read again I don't know if this is exactly true but from what I read this is basically just looking okay how I know I have two inputs and I have two cases.

00:46:20

Claudiu Leoveanu-Condrei: So from the two inputs do I need both? Do I need one? uh in this case you just need one because the second one is the same. So basically the second one doesn't contribute to anything. It's like comparing two

Andrei Ungureanu: Yeah, you said it's redundant.

Claudiu Leoveanu-Condrei: PS.

Andrei Ungureanu: Uh, but maybe you have to put it there because you have to not as

Claudiu Leoveanu-Condrei: Yes. Yeah.

Andrei Ungureanu: inputs.

Claudiu Leoveanu-Condrei: Yes. It's it's basically like tuples, right? When you're comparing two poles.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: So basically when when you're saying uh this stuff 21 and uh 31, right? So basically uh you you will only pattern match based on the first thing because the second is the same in the tupo. So it's something like this.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: This is how at least how I think about it. Uh then what I wrote here go helper is private to some too. Sure. Uh where always follows a body. So um apparently this is a rule.

00:47:15

Claudiu Leoveanu-Condrei: If you don't do it like this then so where cannot stand alone. You always have to follow into a body with it. Which kind of makes sense because where would you use it uh other than in a body to define something? Uh but where does not replace the body, it augments it. And here is just like um I'm not sure exactly like I mean I could not make an example but from what I read they said that it only augments the body and not replacing the body. So I assume at least you know in my mind is like um I'm not even sure exactly like what does it mean to replace the body because it's just like a so where you would use it to define another thing right because that's what's happening here you're saying there is a go function but I don't define it anywhere I'm just it's like a promise and I'm introducing it and then I'm saying okay where this. So basically I assume that where is just a way to make one function more expressive uh than it is normally.

00:48:27

Claudiu Leoveanu-Condrei: So you're just using it to add extra functionality. It's like you would write methods in in other programming languages. But it's a weird way of writing methods, right?

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: It's like I don't know maybe in Python it's like defining a function inside another function something like that. Uh closer.

Andrei Ungureanu: So kind of closure

Claudiu Leoveanu-Condrei: Yeah, it's closer. Yeah. Yeah, I think it's closer. Maybe actually this is a good question. Uh I don't know. This is actually a good question. Good question. This didn't think about it, but now that you said it, I think it's closer. Yes, at least looks like it.

Andrei Ungureanu: Well, at least uh like uh we talked about earlier, it's a good mental uh way to picture this this

Claudiu Leoveanu-Condrei: Yeah. To think about Yeah.

Andrei Ungureanu: moment.

Claudiu Leoveanu-Condrei: Yeah. I agree. Then uh they said in the book u ranges over types and I was like what what does it mean ranges over types man?

00:49:38

Claudiu Leoveanu-Condrei: And uh yeah, I I digged a little bit and basically it's just saying and this was also at the end of 2.3 I think that basically when you say list of a right a can be anything and it it makes sure so lean makes sure that when you put it something like this then it's like a placeholder for a type and and this is once again why types are first class citizens right because here you can literally You're literally making a statement that alpha can be natural, boolean, or whatever, right? Whatever type it can be, the list is going to return to you back that same type uh when you're when you're making it. That's what you're claiming here, which kind of makes sense. Then I had I kind of was like always looking into conventions because of that

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: MK. Uh so apparently this is like conventions for them. So lowerase Greek are type variables then uh lowerase Latin are value variables and then uppercase Latin are type names and propositions. Um so fine I guess like we saw with true capital true.

00:50:54

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: So that's yeah that would be like a type name.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: Cool. Uh then here I think it was the thing about um showing uh sensitivity and all of this and here was interesting. I I actually didn't notice this. They said that you if you notice or I didn't I didn't notice it but if you look at this stuff so you get an to right. So again this is basically something that has two inputs and returns one. But these two things um are the same because they are right associative. So this operations are I I think everything is more or less right associative here in this programming language. But in this case, these two are the name because you first evaluate this, right? So, and then it's this. So, it's these two things are the same thing. It's like writing this without parenthesis. But if you put it like this, then basically these two get wired together and then you're

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: saying that you're returning a a n.

00:51:58

Claudiu Leoveanu-Condrei: If it's if it's basically if it would be like this, then you're saying that you're taking a nat and you're returning a function. you're returning a mapping. This is what this would mean above. But it's not true because it's right associative. But in this case, it is. In this case, basically you're taking a mapping. You're taking an function and you return a natural. This is what this says. Essentially,

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: this they are not the same thing right now because this no longer evaluates right to left. I mean sorry it still evaluate right to left but it treats this at one object.

Andrei Ungureanu: Oh.

Claudiu Leoveanu-Condrei: So in this case this is one input. We have one input and one output. There are no longer two inputs and and one

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: output.

Andrei Ungureanu: So you can have like the cartisian product right in the third. Yeah. There, right? Because you have two inputs and you have one output.

00:52:54

Andrei Ungureanu: This Yeah.

Claudiu Leoveanu-Condrei: You mean you mean here? Not really. Um.

Andrei Ungureanu: Don't you have two two inputs or you have one input? You have a function there with one input or one

Claudiu Leoveanu-Condrei: Oops. No. So, so again when you see this stuff,

Andrei Ungureanu: output.

Claudiu Leoveanu-Condrei: this is a a mapping. So, this is a function in this case. It's a Yeah, it's it's a mapping. It's it's like what it was above here. Uh yeah, so it's it's we're in this case right here. So this is basically saying that we have an add to an int and then yeah, it's a it's a function of one input variable that returns an n. So that's in this case right here this is one input right now which is a function and again if you think about this because this is something that I

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: seen then this this I didn't notice but this right here is what so this right association rule is what enables curring or partial application which is insane If you think about it, so this stuff right here is basically curring this line that I'm that

00:54:24

Andrei Ungureanu: I'm not sure.

Claudiu Leoveanu-Condrei: I'm currently Yeah.

Andrei Ungureanu: Uh I mean I didn't understand this to be

Claudiu Leoveanu-Condrei: So I have an example here.

Andrei Ungureanu: honest.

Claudiu Leoveanu-Condrei: Uh what does it mean? So we have the natural, right? And I said that every single type has some functionality, right? So like methods. So we have the M add.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: And in this case, as you can see here to the right, it's uh basically of it's taking two inputs and returns one output, right? Which kind of makes sense. Uh then what we have here, right? I mean, look at the distinction between these two.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: So this one takes not and it turns a not right. So we look at this one right here and it says, oh, it's a n to a not now suddenly.

Andrei Ungureanu: Thank you.

Claudiu Leoveanu-Condrei: So it's like you collapsed one input. So the only way to collapse one input if this is if this is if this is occurring, right?

00:55:21

Claudiu Leoveanu-Condrei: So if we look at this example right here, so if we do eval, right? This is eight. So if we do eval and you're saying 35, this is an eight. But is eight in a curring form? As you can see here. So you first take the ad with an input three and this is a function right? So add with the input three is a function and then you're applying the function to yet another

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: input which is five.

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: So it's it's like f of g of five. It's something like this where f has a default if you actually want to be more pedantic is something like uh I don't know

let's call it three right it's something like sorry it's it's this sorry it's this so it's f with a parameter are already applied and then inside the function there is yet another function that would call five it's curring.

Andrei Ungureanu: I think I get it

Claudiu Leoveanu-Condrei: So this is so this is how they default in lean.

00:56:31

Andrei Ungureanu: now.

Claudiu Leoveanu-Condrei: So the default mode in lean is to curry every everything function has as a concept has curing application. So when you do not oops sorry when you do not uh when you do not and three and five what you're basically saying take three put it into the add function and then that's a function returns a function and then take that function and add it basically take another input in so it's always function application this is how it works it's is function application in in LIN.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: Uh which is interesting. And then I wanted to see okay but how would I write this not as a curring right? How would I write it as a like literally a function that takes two arguments and adds them without curring. And I define here a function, right, which takes a cartisian product over the pairs of possible integers and returns an integer. And then this is just a function that takes the tuple a and b and then sums a and b, right? And then I can do the normal add pair like you would we would normally do it,

00:57:51

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: right? So it would be like add and you pass three and five directly to the main function. You're not doing this like you pass only three and then you call the function again with five

Andrei Ungureanu: And what does uh net add three return? So you don't have like a five. Uh yeah.

Claudiu Leoveanu-Condrei: uh net attri you mean here.

Andrei Ungureanu: So it

Claudiu Leoveanu-Condrei: Yeah. So so this is a function now.

Andrei Ungureanu: it's

Claudiu Leoveanu-Condrei: So this returns the function that you would apply it to five.

Andrei Ungureanu: okay.

Claudiu Leoveanu-Condrei: That's it.

Andrei Ungureanu: Okay. Think I get

Claudiu Leoveanu-Condrei: Now you see so so so that's the point it's like this this basically this returns a function right now

Andrei Ungureanu: it.

Claudiu Leoveanu-Condrei: it's it's like writing define f of x uh uh I don't know x I don't know x + 5 or whatever something like this and then you just leave it there right you're not calling it so it's something like

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: this

Andrei Ungureanu: But in this case the f of x would be three no matter what the x would be.

00:58:56

Claudiu Leoveanu-Condrei: yes You're you're defaulting the input of the ad to three.

Andrei Ungureanu: Right.

Claudiu Leoveanu-Condrei: Yes.

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: So whatever you would add now afterwards,

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: right? Like here because that's why I put them one after the other. Whatever you would add here,

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: it's going to be something plus three.

Andrei Ungureanu: Right. And that in turn returns another function and we can go on forever. Right. Like this.

Claudiu Leoveanu-Condrei: Yes. Only when you would call eval then you would evalid to to this. But for instance, right, let's look let's look at something important, right? If I'm doing a val of not of add of three, I cannot do this, right? I'm not allowed because I'm I'm lacking.

Andrei Ungureanu: cuz you'reating a

Claudiu Leoveanu-Condrei: Yeah,

Andrei Ungureanu: function.

Claudiu Leoveanu-Condrei: I'm lacking one more input. I'm not allowed to to do this.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: But in check, you are allowed to do this. And you can see the type basically checking checking

00:59:53

Andrei Ungureanu: Yeah. But check it just shows you the T. It doesn't have a

Claudiu Leoveanu-Condrei: Yeah, exactly.

Andrei Ungureanu: concrete.

Claudiu Leoveanu-Condrei: So you're seeing the type but when you invoke it then it applies that but the way it applies it you can deduce it from the from the type.

Andrei Ungureanu: Yes.

Claudiu Leoveanu-Condrei: Cool.

Andrei Ungureanu: Yeah, I get it.

Claudiu Leoveanu-Condrei: Okay. So then uh what do we have here?

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: So here we have the uh checking of tupose uh which again I seen that they put the one2 I actually this is nice but then I obviously have the question like okay but what if I have 1,000 goddamn input can I do 1000 and of course it cannot do that right that's not possible it's like this is how you get a3 so just look at this like uh you So basically you always take the second pair right you take the second

Andrei Ungureanu: 2.2.2. Okay.

Claudiu Leoveanu-Condrei: pair I mean if if you if you want to see it is basically so let's do again a a2 a3 right so if we do two if we look to the right it's taking the one through three right now if we're doing one say zero right so now if you do two again then it's going to take two and three right boom if we write one it's going to take the one so you always unpack pack it.

01:01:15

Claudiu Leoveanu-Condrei: Uh you unpack it like this guy right here. If you remember where was it the uh the maybe yeah this guy. So you unpack it exactly like this list. You you take the head and then you keep the rest head and tail. It's all it's always like this. So this this is what's happening here as well. you you you keep unpacking it. Well, all is you you the head is only one number so or or one member and the tail is all the others and then you keep applying this one two one two one two until you get the the one that you want. It's like it's like binary search lit literally you keep splitting

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: branching.

Andrei Ungureanu: Can you leave the only two there to see the outcome?

Claudiu Leoveanu-Condrei: Yeah.

Andrei Ungureanu: No, just one. One. Two.

Claudiu Leoveanu-Condrei: There's not possible because you take zero

Andrei Ungureanu: Uh okay.

Claudiu Leoveanu-Condrei: directly.

Andrei Ungureanu: Oh no no uh no with two now instead of one like you have.

01:02:22

Claudiu Leoveanu-Condrei: Okay.

Andrei Ungureanu: So it's one two three there.

Claudiu Leoveanu-Condrei: Mhm.

Andrei Ungureanu: Okay. Okay.

Claudiu Leoveanu-Condrei: So one is zero and two is one to three.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: Yeah. So that's how it's how this works. And they recommend obviously so if you get a lot of this stuff then you make a structure and you you do like uh uh dot notation to get the memberships. That's what they they recommend. Okay.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: So that was 2.1 for me. Uh then let's move on to 2.2. They wrote there that um exactly what we already discussed uh which is that one way in which led dependent type theory extend simple type theory is that type themselves entities like not you see they they call them like variables like Nat and bool are first class citizen which is to say that they themselves are object for that to be the case each of them also has to have a type right so then what they what they went through here is to basically define um again if we remember the convention which was I think somewhere here you see lowerase Greek are type variables so if with this in mind I don't know where we are here so basically this is a type variable right now according to them uh so yeah it's basically a natural

01:03:48

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: um a natural type. So we just define a variable that is a type. So that's why they are class uh first class citizens because you can do stuff like this. Uh or you can define being a bull or you can define uh types themselves. So this is not a type variable. This is basically um sorry this is a type variable and this is like uh this is

used to basically do propositions or types themselves. So here you create a type itself like u let's say I don't know like a list or like a prod or something like that. Uh I I'm not sure exactly like what is the difference between these two because to me they look the same. Uh, I mean I would still call this So I Wait a sec. How did they call it here? Uh, where was that? Oh, uppercase Latin type names. Okay, so not type variables but type names. Okay, so I assume oops where was it? So I assume is because uh this is the variable because it has only one like you have only it's like alpha and is of type basically natural that's it here is again a mapping right it's something that takes a type and returns the other type uh here it's again it it takes a type so you take two inputs here and it turns again it's you see

01:05:38

Claudiu Leoveanu-Condrei: even now for me it's weird to talk about it like this but in this case it takes two inputs types and returns one output type and this is what prod is. So this is a way to abstract things right. This is just a way having types as first class citizens allow you to do this kind of like abstraction in which you just define prod or you just define I don't know weird things like you can define very abstract things that you basically there's no variables anywhere right it's just like mappings and functions and that's it and then you

Andrei Ungureanu: Yeah. Yeah.

Claudiu Leoveanu-Condrei: you Yeah.

Andrei Ungureanu: You define your own types basically.

Claudiu Leoveanu-Condrei: So basically what you can do with this is I mean when you kind of like get the gist of this I would assume that you can then make you basically try to prove right that you know that this is plot right but maybe there is some other mapping out there something let's call it uh I don't know like y the mapping y we don't know what it is but we know that if we would have we would know that if it matches prod right so if we prove that y and prod they have the same signature then we kind of know like that it should be of this kind of type right it should take two inputs and return uh an auto type right so you can do this kind of abstract pattern matching because of

01:07:03

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: it okay um here I just checked like okay yeah I expect it to be a natural uh uh and bull and that's exactly what it I mean they they write it like that but that's uh I I checked it and I checked it here just to prove myself that is it what I expect it to

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: be or not and apparently it is. So basically so G this which we define as a type of type of which type. So G takes two inputs right in in this case we gave it a NAT and a bool because we already defined it and I expected this because of this to be the cartisian product of of NAT and B right

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: that's what this has to be by definition and I proved this with reflexivity uh which is true you see this uh nice I know confetti on the

Andrei Ungureanu: Mhm. Yeah.

Claudiu Leoveanu-Condrei: left goals accomplished yeah exactly so when you

Andrei Ungureanu: I saw that you have a competitive

Claudiu Leoveanu-Condrei: prove Something is confetti.

01:08:11

Claudiu Leoveanu-Condrei: You mess it up. I don't know what it is. It should be a whip. Okay. Uh then yeah, here here comes the the rabbit holes, man. It's like this is just like okay. So then I had the question about I I read about constructors and I were like what the hell is a constructor, man? I don't get it. And then basically constructors are defined with inductive. So wherever you go in so if you go into the bool you would see that there is this right it's inductive bool which is of type

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: whatever u it's a type sorry it's a type so boolean is a type where false is a boolean and true is a boolean uh so that this is how you define the thing

and there is this keyword like inductive and this is a very very uh deep rabbit hole because you will see you will see my question and I they they they will not even answer all of them and I'm not even sure I fully understand them.

01:09:17

Claudiu Leoveanu-Condrei: So I have to reread and constantly have to like really really understand like like a puzzle. I don't know if if you did if you ever did puzzles like and and you solve a puzzle by um accident you just solve it but then you just try to solve it again and you just you don't know how because you solve it by accident.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: So it's the same here. I I feel like I solved it somehow in my mind by accident but I it didn't stick like what the hell does it mean?

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: But inductive is just this is how they define things. So then I had the question okay well uh does thisly point to set theory as in if boolean is false or true then you getting a type boolean cannot immaterialize into an op of the same set. So basically what what what is saying what I'm saying here is this kind of notation right here is is just a way of saying like you know in set theory you have that there is a set that has two elements right false and true and you're just picking uh only these two elements from the set and that's boolean right but no because I forgot right uh and and apparently this is like a difference between uh set theory and type theory and and the difference is in the sense

01:10:40

Claudiu Leoveanu-Condrei: that type theory is like it's using constructive mathematics. So in constructive mathematics you have to as you will see right so the law of the excluded middle right if you know that so the law of the excluded middle is is basically saying that uh it's either a or not a right that's that's more or less like a a tautology at this point and with that you can prove things right I mean let me uh law of the

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: excluded middle Yeah, you see, so it's either P or not P. So if you think about it, right, you can prove anything because the or accepts that all if one of them is true, then this this whole statement is true, right? So with with with the law of the excluded middle, you get into a lot of paradoxes because you can prove that God exists,

Andrei Ungureanu: Yes.

Claudiu Leoveanu-Condrei: that unicorns exist, that anything exists, right? So you can do whatever you want because you just have to find a you can say the the sky is blue uh or God doesn't exist.

01:12:02

Claudiu Leoveanu-Condrei: Well, this sentence is true. There you go.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: You just made a sentence that is

Andrei Ungureanu: But doesn't this imply that you have to have some sort of relation between the

Claudiu Leoveanu-Condrei: true.

Andrei Ungureanu: two statements?

Claudiu Leoveanu-Condrei: No, doesn't have any relationship. That that's the whole point is that as long as you have two propositions, right?

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: So you're basically I mean in in this case that was how to say it that was like a little bit more in in one direction, but it's not like that. It's like basically this is saying like it would be more correct to say that this is saying that the sky is blue or the sky is not blue. Well, thank you. like how does that help me right? It's like uh sure you can prove a lot of things with this stuff but usually this extends this extends also to what I said before because in classical logic you can say things like the sky is blue and God doesn't exist right and the second uh clause doesn't even count because you just stop at the first one because the sky is blue you no longer evaluate the second one.

01:13:14

Claudiu Leoveanu-Condrei: So by by definition of the or you just all the entire the entire sentence is is correct. So this leads to a lot a lot of paradoxes like a lot of that's why classical logic doesn't apply in reality and that's why even if you treat this like this or even if you say that uh well the sky is blue or the sky is not blue right this admits if you think about it only to it's a binary choice right u but what if it's not I mean and this whole question like what if it's not a binary choice leads to fuzzy logic so fuzzy logic It's something I even use in one of my papers. But um at least to this stuff, it's an entire field where we just basically say that it's not only zero and one and it could be zero and you know 0.15 and 0.25 and 35 and however you want to define it, right? Uh so that's like a yeah you see here it's you have three variables, right? It's not only zero and one. It's like it's zero cold, right?

01:14:18

Claudiu Leoveanu-Condrei: Warm and hot.

Andrei Ungureanu: Everything is

Claudiu Leoveanu-Condrei: Yeah.

Andrei Ungureanu: here.

Claudiu Leoveanu-Condrei: So uh again this is one thing about always to keep in mind about set theory. So set theory and classical logic uses the law of the excluded middle. Okay that's fine. But there are some guys which they you see this guys intuitionist this guys hated this idea. They didn't like it at all. They didn't like that you can do this stuff. They they they believed it's like shenanigans because you can prove weird things with it. And while it's fine, they didn't like it. Right? So they invented these intuitionist framework and that is constructive mathematics. So in constructive mathematics, for instance, you cannot use the law of the excluded middle. That's not an accepted axiom in the framework. And you're also not allowed to say well you know for any x right what does it mean for any x like again in classical logic you can start with for any x a real number

right let's say and you go on and you go on and you do some proof by contradiction right and then you say well x cannot be a real number x can only be natural number Right.

01:15:37

Claudiu Leoveanu-Condrei: So what you did here is like again shenanigans like you said well I'm

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: assuming the X is a real or whatever and then I'm going to prove the contradiction that is not a real uh by doing some shenanigan like I'm creating an example or something that is not real. Well these guys had the problem with this as well. uh they didn't like this idea that you can say any any what is any like how can you say there exists an X and their whole point was well before you say this thing you have to show me the X you you have to make the X more palpable more you know for instance this guy did didn't believe in infinity for them infinity was like that thing doesn't exist the only thing that exists is affinitism or hyper perfinitism or something like this. There's a lot of philosophical uh debate. If you want to read about this, I recommend you this stuff. Um and I think they even had on visionism. I think I they had on this come on sorry for the flash this stuff.

01:16:56

Claudiu Leoveanu-Condrei: So there are a lot of good entries in Stanford encyclopedia philos philosophy and yeah here if you if you're interested in this kind of they have a lot they have on on whatever you want to search here there is paro philosophy they have about more more anything but you can read more about here it's like it's very interesting to um yeah

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: to to why did they not want this so I for instance when I'm doing math I'm definitely a constructivist because I'm using programming languages and we're all constructivists if we're using programming languages because that that's

what type theory is. Type theory is peeled on the foundation of uh constructivist theories. So we're not using set theory and that was the whole point of this uh number seven that it's not set theory it's type theory and I tried to make here a bit more clear what do I mean by that. So uh as I explained or as I said before like uh in set theory the universe already exists right so it's full of stuff and the fact that you say the boolean is this set of zero and one is just picking like these two pre-existing objects and puts a label on them right uh but zero doesn't care that it's a boolean right zero can be an integer right or zero can be I don't know it can be some some other You can define zero to be many many things and that's just a prompt of of

01:18:27

Claudiu Leoveanu-Condrei: this uh of this idea of set theory and mathematics itself is because is is is this if I summarize like selection from a pre-existing universe but in type theory there is no universal soup of values like when you say bull well you have to create what does it mean right when you say bull what what does it mean like you have to create some rules that define what the bool is. So when you apply the rules, you get the bool back, right? So that's the constructor. So basically bool has two inhabitants which is false and true. And this thing they don't exist before bool is defined. They just they are not there. There's no universe in which they are. You're just making them. You are saying that they exist. Um and that's it. And what's also important is that they cannot they should not conflict with any other. So there cannot be any other true and false other than for bool right as you can see here above like zero can be the zero of the integer right or something like that right is the same thing but it cannot here it cannot be the same thing I mean as you saw in in in tupils right it's a it's a pair of 0 0 and so on so everything has a default as long as you can make it a default as long as that default exists That's the point.

01:19:57

Claudiu Leoveanu-Condrei: That's why you cannot have the default for anything. But that's you can make it for something only for some certain things have a default like the

natural numbers have the default, right? Or uh I don't know like the reals and so on. They they have a default. But this this is a nutshell. It's basically the difference between selecting from a pre-existing universe which is the set theory and generation from rules which is the type theory. So obviously we're generating rules here, right? You're you're defining what a bull is and you're saying that well this is the rule for the boolean. It's like it has to have two values, a false and a true and there you go. That's what it is. This is a boolean. Um but it's more than exactly that

Andrei Ungureanu: but it's defined by itself or how how is that possible?

Claudiu Leoveanu-Condrei: that's the deeper question because that's where it leads to inductive. What the hell is inductive? And yes,

Andrei Ungureanu: Oh,

01:20:51

Claudiu Leoveanu-Condrei: that's the Yes.

Andrei Ungureanu: that's why it's inductive. the

Claudiu Leoveanu-Condrei: you you will see because I have it I have it below. So basically uh I even wrote here.

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: So for boolean the distinction feels trivial because both have exactly two elements right. Uh so when I say boolean the boolean in type theory and the boolean in set theory right I mean if you look at this it might be a bit confusing but if you take the natural it matters right because in set theory natural

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: is some infinite subset of the universe right I mean it's infinite we know the natural never end right but in type theory it's exactly the terms you can build with zero and stock right so if you go into the

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: net Right? You see that we get an inductive net where well the zero is the net and the sock is this uh basically uh type right is taken it takes an n which is a natural and returns a natural.

Andrei Ungureanu: Mhm.

01:21:51

Claudiu Leoveanu-Condrei: So basically a natural number in in in in lean and type theory itself is just something that uses these two rules. That's it. That that's what defines a natural number. Compare this with

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: uh

Andrei Ungureanu: So in this like system you don't need to define infinity for instance right because you have those two

Claudiu Leoveanu-Condrei: Yeah, exactly. But everything is going to be finite,

Andrei Ungureanu: rules.

Claudiu Leoveanu-Condrei: right? You you will never reach infinity. There's no such thing as infinity.

Andrei Ungureanu: Right. Right.

Claudiu Leoveanu-Condrei: Yeah. So that that's a distinction.

Andrei Ungureanu: Yes.

Claudiu Leoveanu-Condrei: You can see it in the naturals. So then I obviously uh had this question. So the proof by induction works because the type is the inductive construction. That's the whole point which is I again I don't think I understand this fully but

Andrei Ungureanu: Let's

Claudiu Leoveanu-Condrei: I checked and I and I asked and I asked L and AI is this like true and I asked different AI and they said yes but I again it's like I feel like a cheat sheet because I got to this conclusion

01:22:54

Andrei Ungureanu: go.

Claudiu Leoveanu-Condrei: but I don't understand it fully. So to me it's like okay so basically when you're defining a type and any type is defined like this if you look at not if we look at I think even list has it uh you see every single type has inductive in front and and this to me when I saw this it was like okay this this means that if they define types using inductive then this

Andrei Ungureanu: Yes.

Claudiu Leoveanu-Condrei: keyword is important this means something right because it's everywhere it's it's a it abstracts every type and then it was like for me because I I I thought about this difference then I thought okay then the rules right the rules

Andrei Ungureanu: Thank you.

Claudiu Leoveanu-Condrei: are basically uh what gives you the induction because how do you apply right the the natural number right there has to be like the rule bake in and I think the inductive keyword is just it's just like um like a I wouldn't call it tactic because I'm not sure that's the right word, but it's like it's like you're saying well use this definition of the type as a blueprint for proving it.

01:24:10

Claudiu Leoveanu-Condrei: Something like this. I don't know if this makes sense. It's like saying look at this boolean type and use it as a recipe to

Andrei Ungureanu: Right.

Claudiu Leoveanu-Condrei: prove itself. And that's what the active

Andrei Ungureanu: Yeah. What I understand here is like it it uh relates uh

Claudiu Leoveanu-Condrei: does.

Andrei Ungureanu: to the set theory and what you had have there written the universe already exists. You basically build your your own universe in type theory, right?

Claudiu Leoveanu-Condrei: Uh I'm not sure I understood exactly.

Andrei Ungureanu: Because in Seth theory for instance uh you you wrote there like right that the universe already exists.

Claudiu Leoveanu-Condrei: Mhm.

Andrei Ungureanu: So you you have to take it for granted uh so to say. But in this uh type theory,

Claudiu Leoveanu-Condrei: Yeah. In this you cannot Yeah. You cannot do

Andrei Ungureanu: yeah,

Claudiu Leoveanu-Condrei: that.

Andrei Ungureanu: you have to prove like uh everything

Claudiu Leoveanu-Condrei: Exactly. Yes. That's also what I understood,

Andrei Ungureanu: basically

Claudiu Leoveanu-Condrei: but I'm not sure this is correct.

01:25:23

Claudiu Leoveanu-Condrei: It's just uh it feels correct to me, but I'm not I don't Maybe they will show us at some point like exactly what is inductive and how it works because I I would be really curious like I mean that's clearly an important uh word uh keyword uh because it's used in every type definition.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: Cool. Then um here I had uh again some questions because of this because to me is it's not intuitive to yeah to so it's again the problem of abstraction right um I'm pretty comfortable with uh with um so for instance in symbolic the framework that I develop I I constantly use functional

Andrei Ungureanu: All

Claudiu Leoveanu-Condrei: programming so my abstractions are decorators and meta decorators so I'm using yeah that that's what I was So I'm comfortable with that but with thinking about types and it's like something it's always something like that. I have questions like okay but like how do you do that right? So for instance I asked myself well when we say that every natural is literally a finite tree right because it's literally finite by definition is finite or a finite power of

01:26:33

Andrei Ungureanu: right.

Claudiu Leoveanu-Condrei: of consecutive uh successor applications does that mean it's stored that way right I mean how do you store that uh and this is how I got to that huge number right so I asked myself okay so it's like are you telling me that if if if we have two to the power 500 112 are you actually having somewhere like like sack of sack of zero and then sack of sack of zero sack of sack of zero the I don't know how many times right so let's see how many times like that that many times is there somewhere like this uh thing and yes apparently it is uh so

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: basically what do I have here how does Lynn know the chain length of sack for right I mean this this all these questions related but then at runtime we get into so basically when you're when you're saying okay I have two to the power 512 well

Andrei Ungureanu: All

Claudiu Leoveanu-Condrei: at runtime when you compile it and all of this it gets represented as the GMP lib um as I s as I told you so it's it's basically this library well at logical level it is indeed this many sock power because that's what this language uses right as an inductive proof or whatever like

01:27:50

Andrei Ungureanu: right.

Claudiu Leoveanu-Condrei: that's how it's represented and at proof level. So basically when you're trying to prove things with it then it's it has this which is this is yet another um rabbit hole this recursor uh in this case is like not rag uh so basically and I and I put more here so basically this recursor is and I think I have it I mean let's go a bit down. So how does induction work without asserting it as an aum? So again for me is like I mean when you say when you define induction like that like I mean what is the difference between that and as and and an action right because an action you just state it like the law of the excluded me right it's it's not P or P fine you just stated something and you say that's true but

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: but when you say inductive that stuff how is that different from this that I don't understand concept conceptually, you know what I mean? Because to me it also reads like an axiom.

01:29:00

Claudiu Leoveanu-Condrei: It's like you're saying that this is boolean and I don't get the proof.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: You you know what I mean? It's to me it's it's like an axiom.

Andrei Ungureanu: Or

Claudiu Leoveanu-Condrei: I look at that stuff right now and it's and and I read it like I'm reading uh not P or P. That's what's in my mind. But it's clearly not that. It's not an axiom. It's the the the whole inductive block is the proof of that block. So then I have to I have to understand how is exactly is it right the proof and I don't get it. Uh and apparently it has to do with this recursor. So this recursor is like a again I don't want to say tactic but is uh for instance I have it open here. I I will share anyway that this file. So it it's here for instance and and you can see it's when I put it right it went straight to piano axioms. So as you can see if you read this so the piano aums are a consequence of this definition right the induction principle generated by uh naturals is the one demanded by the axiom of induction.

01:30:13

Claudiu Leoveanu-Condrei: So it's an axiom you see. So basically when they call that inductive thing I assume that that's an axum and this is

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: his definition which I don't get I don't get this I mean sure I I I see that it's about zero and stock I get this that is the successor and of zero and so on and so forth but I don't get like what is the motive like what do you mean is like this is

like a type now what are we talking about what is a motive I don't get it so I hope they will teach this at some

Andrei Ungureanu: motive motive isn't it

Claudiu Leoveanu-Condrei: Sorry.

Andrei Ungureanu: uh defined anywhere you

Claudiu Leoveanu-Condrei: If if you hover if you hover over this, it's basically saying that it's a type from natural, sorry, is a mapping from natural to sort U. And I will show you sort U is another rabbit hole I went to. So sort is uh it's not it has nothing to do with sorting by the way. Uh so there's no sorting or anything like that.

01:31:15

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: Sort is it's literally universe. That's the universe. And sort u is basically in this case is just saying that the natural goes to uh sort U which is a universal type of um yeah it's a universal type. basically it's it's it's it's none of the types like boolean natural is one that abstracts all the types this is how I understand this but I don't know exactly like this um um this formulation like all of this I don't get it because you will see below this u is creating a universe and this is basically now this is a pattern over types right now so this abstract types that's what's happening here but I I don't understand it yet, but apparently this is why this is why um you can use it in proofs because it's handled by this recursor and this recursor apparently is just a way to define the axum of induction which is that which I just opened before and um that is basically enough. I don't know how they did this, but apparently I believe that's enough to prove to to create any type in type theory.

01:32:43

Claudiu Leoveanu-Condrei: That's why it's so important. So basically the recursion the recursor sorry the recursor which is the it has something to do with the axum of

induction is enough to create the boolean type is enough to create the natural the integer the whatever the list the you name it that's enough I don't understand how yet at least I don't get it how you you Okay.

Andrei Ungureanu: Yeah,

Claudiu Leoveanu-Condrei: Yeah, it's a bit it's a bit mindbending.

Andrei Ungureanu: I mean I don't understand Hi there.

Claudiu Leoveanu-Condrei: Yeah. But apparently this is so induction is enough to define types like boolean

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: and integers and all of this. That's what what what this is saying. And it's using a pretty basic formulation, right? So it's using it it's it's it's abstracting the natural over types. So that's what this is what they did here. This is how I read this. So this there are no numbers here. There's not zero and ones and anything. These are just types.

01:33:54

Andrei Ungureanu: Thank

Claudiu Leoveanu-Condrei: It's like saying the type zero or like the type it's like capital true,

Andrei Ungureanu: you.

Claudiu Leoveanu-Condrei: right? Or how we discussed in the beginning. It's it's like this is just a an abstract pattern that is enough to reproduce basically naturals and all of that and I don't understand how I mean I yet I don't understand how but it's very interesting. Uh so anyways I list I put the link to depth just in case. Uh then uh I just have here yeah I evaluated this bananas numbers. Uh then we have uh now we get to the type thing. So basically what is a type right? So you can check so you can always check right if you look on the right. So if you check type zero you see that type zero is of type one. If you do type one you see that is of type two. If you do type three is of type if type two is of type three. So what the hell does it mean?

01:34:58

Claudiu Leoveanu-Condrei: Like it's always plus one. So uh basically even they said it there in the book is that the underlying

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: foundation has an infinite hierarchy of types. So basically what this means I don't know if you if you know the the Russell's paradox of the of the set theory I don't know maybe a version of it you you you heard it as the Hilber hotel. It's the kind of the same thing if you heard the Hilbert Hotel like when you Yeah.

Andrei Ungureanu: Oh, no.

Claudiu Leoveanu-Condrei: when you put basically the um like the

Andrei Ungureanu: No.

Claudiu Leoveanu-Condrei: so you are like a manager at the hotel and then then you then you get a bunch of guests, right? And the guests are infinite. They come an infinite number of guests. Then you book them into uh all the rooms uh possible rooms that you might have from one to infinity. Right? Then uh tomorrow the next day there comes yet another uh another bunch of like uh people which is yet another infinity and where the hell do you put these guys right?

01:36:04

Claudiu Leoveanu-Condrei: So the way the manager solves it because he has the infinity as a friend is

Andrei Ungureanu: Not

Claudiu Leoveanu-Condrei: to move all the people in odd or even rooms right and then all the other rooms become empty. Uh so and he puts these people into into the other rooms and then the next bus comes in and let's say they are just the prime numbers or whatever like and you do this kind of shenanigans over and over again.

Andrei Ungureanu: Yeah, I knew something similar like when you have to go uh a

Claudiu Leoveanu-Condrei: So

Andrei Ungureanu: distance and first you have to go half of the distance and first you have to half of the half.

Claudiu Leoveanu-Condrei: yeah the Zenos paradox. Yeah.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: The the Ailles and the tortoise paradox. Yeah. The Zeno paradox. Yeah.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: So basically again the Russell paradox is in set theory and there is an equivalent if type in type theory because they are connected.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: So um it's just a way of saying that creating this tower of types prevents self-referential type membership loops.

01:37:04

Claudiu Leoveanu-Condrei: So basically no universe can contains itself. It's like the set that cannot contain itself and so on. So you cannot have right basically Russell asked what is the set of all sets and this led to paradoxes and you have the same thing in in type theory like what is the universe that contains all the universes right so you have to solve this problem because if you don't solve the problem then your system is not consistent you have always this this tug of war between um a system being complete versus consist consistent. I don't know if you ever heard about this but the mathematics is is a tug of war between a consistent versus a complete system and godel if you ever heard of godel so godel prove

Andrei Ungureanu: Wait.

Claudiu Leoveanu-Condrei: that you mathematics is not a complete system so you have to agree upon do you want your mathematics to be complete or consistent so obviously everyone said consistency right so then you know it's incomplete By definition, by the way you define the axioms and all of it, you know from the the definition that your mathematics, so the mathematics that we're using now that we know as as society, as as humanity, as as homo sapiens, we already know this is incomplete by definition.

01:38:28

Claudiu Leoveanu-Condrei: So there has to be a universe somewhere out there where there are weird things about math uh that we don't know about. uh and we have to accept that because otherwise if we don't accept that then our mathematics would not be consistent which is a huge pain right because if your mathematics is not consistent then you have contradictions right and you cannot afford that then your mathematics is no longer pure so it's like a tradeoff so I anyway I I put this here just like like a detour into paradoxes and stuff like that uh then I had this like huge um yeah I basically asked myself what what's the what's the relationship between sort and type like and I went here like I don't basically all of this from up until here is just like as trying to get to this point like how how should I even think about this so if you look at sort zero right so if you you can do check oh sorry not here but let's do it here Yeah, you see check s zero. So if you check s zero and you look on the right, it's basically saying um it's of the type prop and sorry it's prop and it's and it's of the type, right?

01:39:52

Claudiu Leoveanu-Condrei: So if we do check sort one, it's saying that well sort one is type and it has type one. Do you remember where we saw this? We saw it here, right? So if I do check type, you see this these things are the same. You see? So what what the hell? Why why do I need to why do I have source one and why do I have type?

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: And why what do I have check prop right and source zero? What the hell is this? Uh and I asked myself this basically why do I need to have both in mind? And then uh basically I started to go into into reading about it and it's like okay so sort zero is just prop. So source zero is basically the universe of propositions. That's that what that's what it means. Okay. It's it's everything at like true and like we saw the the capital through and the capital false and all of that. It's like all that infinite of statements of that kind.

01:41:03

Claudiu Leoveanu-Condrei: Then at source one you get types, right? So so is the universe of small data types. So types is is like boolean and integer and not and all the others that we've seen above, right? And this goes on forever and forever. This is like okay so what is sort two? Well, by definition is a type of a higher degree than sort one. So it's like imagine you abstract over types, right? Again, you see where the abstraction starts to literally melt your brain because what do you mean you abstract over types? So basically you have function over types. It's like you're saying you're you're studying functions that their inputs are types like natural and boolean and that's it and and you just find patterns in that, right? And then you can go even further. You can say sore three is it's it's an even bigger universe of types of types of types,

Andrei Ungureanu: Thank you.

Claudiu Leoveanu-Condrei: right? And this keeps going forever. That's the hierarchy. That's the um that's why they need to do this to avoid that paradox, right?

01:42:17

Claudiu Leoveanu-Condrei: So you always have to construct a higher type, a higher universe.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: That's why you have to do it because otherwise if you're not doing this, you're running into the paradox.

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: And that's what I was saying. You're no longer consistent at the level of the type, right? Which you want that. You want to be consistent. So, and then we read the we reach the sortu which we've seen in the inductive thing. And that's I assume is just like um yeah it's it's the I don't know how to call it. It's like the super level. I don't know how what it is exactly but it's like the the higher the highest abstraction. It's like the universe of types or the universe of I don't know how to call it but something like that I assume. Um but even this doesn't have its own universe because again no

universe can contain its own universe right so what's the type of sort of view well it's sort of view plus one right it's always plus one so then uh I had this thing like why is sort so to me it was a very confusing name and it comes from many sorted logic that's why sort sort means sorts like how any sorts of ways or something like that.

01:43:36

Andrei Ungureanu: Oh,

Claudiu Leoveanu-Condrei: So it's like kinds categories stuff stuff like this is how you should think about like categories

Andrei Ungureanu: okay.

Claudiu Leoveanu-Condrei: kinds. Um, and basically what I wrote here is that a variable of type

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: bool can't be confused with one of type nat even though both and bull live in the same sort right so they both live in the same universe the natural and the boolean but you don't confuse them right and this is because uh this is like one nuance that is different between type and sort

Andrei Ungureanu: Yes.

Claudiu Leoveanu-Condrei: so basically both not and boolean are sorts but and they are also types but you cannot distinguish them between based on types sorry you you can distinguish them them based on types but not based on sorts right because based on the sorts they are just like the same thing it's like a it's like the way I think about it's like a inheritance it's like natural and boolean by being types they are just instantiation of that specific sort in in That's it's like you pick something from the

01:44:41

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: universe and you say this is boolean and this is natural. You're making it specific.

Andrei Ungureanu: Yes. Okay.

Claudiu Leoveanu-Condrei: So that that's how at least how how I'm thinking about it.

Andrei Ungureanu: Understand.

Claudiu Leoveanu-Condrei: Then they had this in the book. Um so basically reading down so on the on the column a sort again a a sort is a universe. A sort and universe is the same thing. uh they just I don't know why they keep the name sort but whatever a sort and a universe is the same thing but if you if you read on the column then um each universe has a type and each type contains a term in that specific dimension I don't know how to call it and if you did on the uh on the x-axis then you get the bigger universes right so as you move to the right you get higher and higher and higher abstractions and and stuff like that. So obviously the first one is the easiest or let's say maybe the the second one is the easiest for us because if you look at the second one right at the universe where you have type so source one you get and we move now on the column then the type is capital bool and the term or the value right is true or false.

01:46:07

Claudiu Leoveanu-Condrei: uh uh lower case true and false, right?

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: But if we move to the left and we reach the prop universe or the universe of propositions, then we see the difference. You see now basically we this is how I answer myself the question like what is the difference between true and true is because you see they live in different universes. You can see it here. So lowerase true is just the value or the term of a boolean type in the un in the in the universe one. And if you look at the prop or capital true just just the type. So basically true and bull they are equivalent at the type level. They are the same type. No not the same type sorry they are they mean the same thing at and their own level and then the term is like yeah it's true intro again. I I I I I've told you about this intro and that MK. I think they are the same thing. Um and then if we go up the ladder,

01:47:06

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: so type one or the universe 2, then you get this weird uh again you get this weird things like you you get a natural you get a mapping from a natural to a type. This is what it's saying basically. Um and this is this is an example a term of this form. It's a function n that takes f of n and I don't remember exactly what was f but f is like let me maybe define it entirely to not cry okay I don't have that but it was something um yeah finite natural numbers that's what it means so basically you take a function of n and you and you return a finite natural number n. That's what's happening here. And then you can go even more in abstraction, right? And and what's happening here is that you take a type and you return a higher order type. So it's like you're taking this finite natural number and you're returning something else. I don't even know how to think about this. I'm I'm being honest.

01:48:23

Claudiu Leoveanu-Condrei: I don't I don't understand. is too abstract for my brain because at this point you see there what

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: is this man they're just function they're just mappings that's what it is you're studying mappings from this point maybe from this point on right from this from here on so the right you're just studying mapping abstract mappings between things so I guess it's category theory right they do that so then uh I I had this like super super big confusion like I checked the sort of zero and I was and I've seen that it's of uh uh of of of type and I was like this is basically why all of this like why this confusion I was like wait what so sort of zero is prop like what why is that and yeah it's basically what I wrote here is that the type of prop is type and the type of source zero is also type so they both have the same type but source zero is a universe of all props. So are we saying that the universe's type is the type of one of its elements?

01:49:30

Claudiu Leoveanu-Condrei: I don't I know it's maybe confusing the way I phrase this but that this was my question because I didn't get it. So um no the answer is not or you basically resolve that if you say that source zero is prop and that's the case. So I proved here again that sort zero is prop by reflexivity and that sort one is type. So basically I proved this uh table these two things from the table just you know for me to be clear that this is the expectation that this should be exactly what it is.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: So again think about sort zero as the universe in which prop is one instance of right but it could be something else.

Andrei Ungureanu: Excellent.

Claudiu Leoveanu-Condrei: It could be that there's a different prop or something else that is or or maybe a different uh the way I I was thinking about it is like why don't we write zero right so so why why don't you write like this why why don't you write zero like this but you write it like this this is how I'm thinking about it so basically sort is the equivalent of zero but when you program you call it zero

01:50:42

Andrei Ungureanu: Okay, I understand I

Claudiu Leoveanu-Condrei: This is what's in my mind right now.

Andrei Ungureanu: think

Claudiu Leoveanu-Condrei: This is how I this distinguished. So it's just so prop is just a is just the writing zero like this instead of zero. That that's what it is. They are specific instantiation of of these universes because technically from what I read this is the correct way to phrase it like sorts. They are just universes.

Andrei Ungureanu: Can you have different instantiations or that is the only one

Claudiu Leoveanu-Condrei: But imagine no,

Andrei Ungureanu: possible?

Claudiu Leoveanu-Condrei: this is the only one possible. It's just like giving it a name. You know, what do you like coding with? Would you like coding with having all the time sort zero or like this nice prop? Right? That's how I think about it. It's like they just made these names to be more readable or more, you know what I mean? It's like just how the f*** do you code with only sort one and zero?

01:51:42

Claudiu Leoveanu-Condrei: That doesn't make any sense.

Andrei Ungureanu: Yeah, definitely. Yes, basically.

Claudiu Leoveanu-Condrei: Yeah. Exactly. So that's at least what's in my mind. Uh so then I had here like yeah since we're here quick check why is this of type type and obviously because the function type must be at least as big as its biggest piece right because again don't forget this is a mapping right so this is a

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: function or mapping more correct and if you look on the right right it has like uh the type is type right so then the question is why is the type type.

Andrei Ungureanu: Yes.

Claudiu Leoveanu-Condrei: Well, the type is type because the type of the prop is type, right? So, if we check the if we check prop, it's type. You see? So, basically, it's taking the highest one all the time. So, if we would have something like prop to type, guess what? It's type one, right? If we had type to type one is type two and so on.

01:52:47

Claudiu Leoveanu-Condrei: It's always plus one. Then we had and this is the last part of it. What is the polymorphism? Uh this is just again the list, right? So a list if you look to the right is defined at list dot curly brackets op. And that u is the universe. Again this is just abstractions to me uh is just a way of saying that the list can take any kind of type. Right? So you can say list of booleans and then the sorry you can say you can say list of

alpha right and alpha can be because it's a type variable right again we we can have type variables you can make alpha whatever you want you can make alpha I don't know let's say springles so then you have a list of sprinkles right because alpha is the type spring and these are just two ways of defining um uh universe. So I define Q and Q prime um and this is from the book as you can see they are using alpha is of type U. So every time you do type U or this basically type and some letter like this.

01:54:05

Claudiu Leoveanu-Condrei: Uh so you have to define the universe first. So the universe is a keyword you define it as being you and then you say well I have a mapping which takes alpha which is of the type of that universe again a universe think about the universe as whether it's prop it's type right what universe so you're basically abstracting over universes now you're abstracting over all the types that exist in lean this is what it's

Andrei Ungureanu: So you are setting over the columns,

Claudiu Leoveanu-Condrei: Exactly.

Andrei Ungureanu: right?

Claudiu Leoveanu-Condrei: Yes. No, sorry about uh you're abstracting over the row the columns. You're correct. Yeah. So to the right X- axis.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: So if you meant from here to the right,

Andrei Ungureanu: But yeah.

Claudiu Leoveanu-Condrei: right?

Andrei Ungureanu: Yeah. On the column.

Claudiu Leoveanu-Condrei: Yeah. So yes,

Andrei Ungureanu: But where's universe V here defined?

Claudiu Leoveanu-Condrei: you here.

Andrei Ungureanu: No, not you. V. Because you have

Claudiu Leoveanu-Condrei: Ah that that's what I so if you try to do this it's not

01:55:04

Andrei Ungureanu: to

Claudiu Leoveanu-Condrei: allowed if you read the error u is already being declared right which this this should be telling you that this is the same formulation right these two things are the same thing just that if I'm closing this if I comment this out this this works right so this is the same thing this

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: one from the bottom is syntactic sugar of these two. So instead of having two lines, you have one but you have to do it like this like with with dot curly brackets you

Andrei Ungureanu: Okay.

Claudiu Leoveanu-Condrei: and if you look at the list this is how they even show it in le right. So here this is the notation. So this is how they show it. So they like this syntactic sugar. So probably when you code things and you see this right it's it's the equivalent of this. That's how I I'm thinking about it. So they preferred this stuff from the bottom. So and because of that I had to use another variable right because u was taken because of here and then I made it V. So but these two are the same and again I proved it here right.

01:56:14

Claudiu Leoveanu-Condrei: So I said uh yeah is Q equal with Q prime and I and I again by reflexivity that they are the same thing. So yeah, this was uh this was all

Andrei Ungureanu: Mind uh mind blowing mind melting.

Claudiu Leoveanu-Condrei: Yeah, it is it is a bit it's a it's a it's it's very deep rabbit holes. You can easily get um Yeah, you you can easily get into deep rabbit holes and keep keep asking at least this is how I learn. So I always learn I ask stupid questions like you see like why sort maybe that question doesn't make any goddamn sense right but I ask

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: myself that I asked myself why sort and then I have to answer it and I don't understand maybe like what do you mean why sort right what kind

of question is that so I keep asking myself stupid question like this how is that's how I learn I I have to ask questions. If I if I don't ask question then I know I'm not learning anything.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: But when I start asking question it's like I don't stop.

01:57:30

Claudiu Leoveanu-Condrei: It's like but why is this like that and why is this like this and wait a second I I I think I saw

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: this. It's like that but wait what? Why is like that? So it keeps going. It never stops.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: So yeah it's fun. It's it's very cool. I mean I don't know. I would love I would have loved to learn this kind of stuff when I was in high school. That would have been amazing honestly to have this kind

Andrei Ungureanu: Yeah,

Claudiu Leoveanu-Condrei: of

Andrei Ungureanu: maybe instead of philosophy or something or maybe you both learned it in

Claudiu Leoveanu-Condrei: ah yeah that that would been cool.

Andrei Ungureanu: philosophy.

Claudiu Leoveanu-Condrei: Yeah. You you have a philosophy class and you do lean. That would be cool. your your professor is teaching you first order logic in lean that philosophy class man that that would be a cool philosophy club honestly but yeah uh I know it's so even splitting into two sub chapters is like it took us two hours so

01:58:40

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: it's a lot it's a lot to unpack But,

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: uh, yeah, I mean, let's see how how it goes. I'm

Andrei Ungureanu: Yeah, it was super super cool presentation. I mean, you went through a lot of I wouldn't even have thought of

Claudiu Leoveanu-Condrei: I'm

Andrei Ungureanu: like 80% of the stuff you you went through to besides the actual course. I mean, so you're you're a good teacher.

Claudiu Leoveanu-Condrei: a course in simple type theory. Yeah,

Andrei Ungureanu: Sorry.

Claudiu Leoveanu-Condrei: but I but I'm but I'm like a a crash course in simp in simple IP how

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: they said it in the book that but yeah I mean this is how I learn and I and I knew that it's going to be like this u I yeah I always go rabbit holes and I have

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: to stop at some point I really have to stop because I don't first of all I cannot find the answer and that's my mind I have I have that unfortunate security mindset you know when you find the when you want to find the bug you find it you just you just have to spend enough time you know and you find the bug uh so so that that that's that's

01:59:52

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: how my mind works it's like I need to go there do I get this and it's

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: just sometime you don't get it or there's not enough time so I have to stop and the stopping part is very hard that's just very hard because it's it's

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: interesting you it's like okay so what is this what do you mean the inductive that thing with the induction that that literally broke me. I don't I don't remember how much I spent on it, but it's just like are you saying like the inductive like they're using this again? And I'm not even sure that's the case. But yeah, I mean this is this is how I learn and I keep grinding and all all these questions are literally in the order that I they came up. I didn't change the order even though probably it would have been better to change the order to be on the right I know moving up where it would have been the discussion about it but yeah nonlinear thinking cool then yeah we have to probably revisit I don't think um the last time we we decided which is going to be the next chapters but now I'm afraid because I've seen how what two sub chapters did to us.

02:01:12

Claudiu Leoveanu-Condrei: So I don't know what the hell is.

Andrei Ungureanu: Oh, awesome.

Claudiu Leoveanu-Condrei: So I think it was up until name name spaces or something like that. So I'm not actually sure we should be doing that even though it makes sense. I mean to to split them like this it makes sense. Um, I agree with GPD, but uh I'm not sure like um yeah, there's a lot here. So,

Andrei Ungureanu: Yeah,

Claudiu Leoveanu-Condrei: anyways,

Andrei Ungureanu: I know that uh way of learning because You you have some material and you

Claudiu Leoveanu-Condrei: we

Andrei Ungureanu: go through it uh one time and you kind of grasp general aspect and you go through another time and you go into deeper and you go uh the third time and

you know everything clicks uh and I'm not sure that this is applicable here because you know it already is

Claudiu Leoveanu-Condrei: Mhm.

Andrei Ungureanu: too uh abstract uh and we're just getting started. Uh I'm thinking that if you have uh like holes in understanding uh at this point uh it won't make much sense uh further on.

02:02:31

Andrei Ungureanu: That's my u like intuition about this right

Claudiu Leoveanu-Condrei: Wait,

Andrei Ungureanu: now like

Claudiu Leoveanu-Condrei: when you say it doesn't make much sense further on, what do you mean?

Andrei Ungureanu: to um go forward without really addressing like some questions like you had before or like having

Claudiu Leoveanu-Condrei: Ah, okay. I see. So it's like to not miss anything as you move,

Andrei Ungureanu: um Yeah,

Claudiu Leoveanu-Condrei: right? Like

Andrei Ungureanu: I mean so actually maybe that's the point here which is uh

Claudiu Leoveanu-Condrei: foundation.

Andrei Ungureanu: which part is the foundation and which part is like the rabbit hole which maybe not that important uh for for this uh

Claudiu Leoveanu-Condrei: Oh, I agree. Yeah. But I I I cannot stop that.

Andrei Ungureanu: post run through.

Claudiu Leoveanu-Condrei: That's the point. Like my mind doesn't stop.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: It's like what is that? That that's it. I I cannot stop. But I agree. I mean uh to me it's like if I don't understand it's like I'm I I hate that feeling, you know, of not understanding.

02:03:38

Claudiu Leoveanu-Condrei: It's like I I keep reading and I not I'm not getting it. It's like what is this? What do you mean? These are the same thing. So again, that that's for me is part of the learning. It's just like how I break things apart. Um but yeah it has to be caps of course I mean it cannot go on forever right but I feel like the foundation is is super important like getting all this like um I

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: I mean I think the whole chapter 2 is part of the foundation because it introducing like d like you know like the the types like double and all of this boolean and how it how they works and of that.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: So I assume the entire chapter 2 is like uh I mean it's called dependent I theory. So it's like the it's literally the backbone of lean. So my assumption is that by the end of chapter two, we kind of know exactly like you know what to expect when you when you when you when you take a when you look over a lean program like you you get a gist for okay so this is about maybe a higher abstract typing formatting.

02:04:54

Claudiu Leoveanu-Condrei: So you get you kind of know already how to read right u the types and all of that.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: So that's my assumption and anyway my my point is not to be like a lean experts. So my point of of learning this is not to become a lean expert. My point of this is to understand the language um good enough that I can control an agent right that can write code for me because I cannot do it right now because I don't understand anything to me it's just like a a wall of code I don't get it so how how do I debug something that I don't understand I cannot yeah like I don't

Andrei Ungureanu: An agenda right in

Claudiu Leoveanu-Condrei: know you to code like a proof or whatever,

Andrei Ungureanu: code.

Claudiu Leoveanu-Condrei: right? Let's say you have a weekend project, right? And you want to learn I don't know like to prove something like I don't know like the Schroinger equation. I don't know what you want something you want to prove something right and you do it with an agent.

02:05:59

Claudiu Leoveanu-Condrei: I mean everyone is using coding agents today but in my opinion you you know you have to know how to code right to use agents because if you don't know what to code then I don't know how you I mean the code is working but is either trash

Andrei Ungureanu: Yeah, you end up V

Claudiu Leoveanu-Condrei: or yeah you you vibe code.

Andrei Ungureanu: coding.

Claudiu Leoveanu-Condrei: So I I want to understand, you know, what I'm vi coding. If I'm if I'm going to vibe coding limb, I want to understand lim. I want to understand, okay, but what is it exactly?

Andrei Ungureanu: Yes.

Claudiu Leoveanu-Condrei: So if I'm I don't know if I'm thinking about the proof, right? Or or it's not even that. It's literally understand. So you're going in China and you don't speak Chinese, right? What the hell are you doing? If you if you if you don't speak Chinese and you don't have a translator or something that translates, you're lost. It's it's a it's it's still an experience, right? I mean, you still get to enjoy the city.

02:06:54

Claudiu Leoveanu-Condrei: You still get to walk around, see the park, smell the air, see the sea, whatever. It's still nice, right? But I don't get it. I mean, I don't get the culture. I

cannot speak to people. I don't get anything. So, I see V code in the same way. It's like going into China and not speaking Chinese and not having a chance. Yeah, maybe nice. Maybe I get something but overall what do I ultimately get not much so if I if I want to v code with you know like so or do some more serious work I mean I'm interested in in in in software

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: verification and all of this so I have ideas in my in my head but I can go right now and start to vibe code with claude or whatever right but I don't get it. I mean,

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: let's say I'm getting a proof back and it says proof checked. Fine. I don't get it. I cannot even read it or like, you know, end to end.

02:08:00

Claudiu Leoveanu-Condrei: So to me, this is like a a way of understanding the language and the like what can you do with it and how exactly are you doing quantifiers and and how exactly are you doing math with a computer? Literally this is what it is, right? So if you if you know math with pen and paper, then this is like with computers. So I want to I want to learn how to do math with computers because why would I use that pen and paper? I don't like it. Computers are our slaves. They should work for us.

Andrei Ungureanu: Actually, I liked it. Uh, I had a project. I uh worked on some uh defy protocols and on blockchain and I had to like go through the paper and um there's like the general paper with a bunch of formulas and then the code that didn't really translate to what the paper was saying. Uh, and I had to do this by hand and I I actually had fun doing it and uh you know going

02:09:06

Claudiu Leoveanu-Condrei: But it's nice, right?

Andrei Ungureanu: through Yeah.

Claudiu Leoveanu-Condrei: When you put the mat on the machine and it's working, if that feeling,

Andrei Ungureanu: Yeah, of course.

Claudiu Leoveanu-Condrei: it's like

Andrei Ungureanu: That was like a goal for me here to like maybe

Claudiu Leoveanu-Condrei: ah.

Andrei Ungureanu: uh try to do actually you can do two things, right? because you have the paper and you like can uh try to prove the paper itself

Claudiu Leoveanu-Condrei: Mhm.

Andrei Ungureanu: and you try and you can uh try to prove that the code that mirrors the paper is correct. And so you have many

Claudiu Leoveanu-Condrei: Yeah. Yeah. That's also my Yeah, that's also my goal. I mean,

Andrei Ungureanu: like

Claudiu Leoveanu-Condrei: I want to I want to take software and prove that it's correct. And probably I would be doing it with an agent or I I'm I'm going to do it with cloud code or whatever like some some something that does code that writes probably better than me, but I still want to be like in the control, right?

02:10:02

Claudiu Leoveanu-Condrei: I still want to understand how exactly am I doing this? I mean, how would I approach it? Like, how should I, you know, I have to understand lean? I I don't see how that's possible. I don't think I don't think it's possible.

Andrei Ungureanu: Yes.

Claudiu Leoveanu-Condrei: So, it's the same. And I don't know if you saw this, but they started um signal uh lean 4 They started to

Andrei Ungureanu: Are you share or you're not sharing anymore?

Claudiu Leoveanu-Condrei: work.

Andrei Ungureanu: Not sure.

Claudiu Leoveanu-Condrei: Am I not?

Andrei Ungureanu: Oh, no. It disappeared sharing for some

Claudiu Leoveanu-Condrei: Oh, okay. Sorry. Uh I I didn't know.

Andrei Ungureanu: reason.

Claudiu Leoveanu-Condrei: So they started to look into formalizing signal formalizing signal lin 4. Oh come on it was recent. You know the signal app.

Andrei Ungureanu: Oh, the signal. I was about to ask.

Claudiu Leoveanu-Condrei: Yeah.

Andrei Ungureanu: Yeah, I

Claudiu Leoveanu-Condrei: So they they want to verify the uh they want to verify that

Andrei Ungureanu: know.

02:11:17

Claudiu Leoveanu-Condrei: signal is correct and they started a project.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: Oh, there you go. This is the one.

Andrei Ungureanu: Oh, by defender.

Claudiu Leoveanu-Condrei: Uh yeah, he obviously he's he's he's involved in all of this, but like where is like signal signal shot? Maybe this. Oh yeah, this is the one. So this stuff. So basically they they uh they are doing right now cryptographic proofs of the signal protocol which is man it's amazing. So this kind of stuff right I mean something like this this is cool. I want to know about this stuff, right? To do this kind of stuff. It's cool and especially now that you can do it. But yeah, this is this is my goal at least what I'm looking into. So let's see how how it goes. Uh the other thing which I by the way I forgot to say at the beginning. I wanted to say at the beginning but I forgot but now I have to say it at the end.

02:12:30

Claudiu Leoveanu-Condrei: So it is what it is. Um I have to move it from Thursday because I cannot join uh anymore Thursday. So it has to be any other day uh but not Tuesday. So I'm considering you are here right now. So I assume you are you are the most interested party. Uh let's the two of us decide when when it should be because Yeah. I mean Tuesday.

Andrei Ungureanu: So you cannot do it uh Thursday and uh Tuesday.

Claudiu Leoveanu-Condrei: Yeah. So uh and Friday works.

Andrei Ungureanu: Friday is like probably No,

Claudiu Leoveanu-Condrei: Yeah. But

Andrei Ungureanu: I was about to say cuz you know Friday usually have some other stuff

Claudiu Leoveanu-Condrei: I know I know I know.

Andrei Ungureanu: maybe through the weekend.

Claudiu Leoveanu-Condrei: So ideally not. Yeah. Ideally not.

Andrei Ungureanu: So Monday,

Claudiu Leoveanu-Condrei: But uh Monday or Wednesday?

Andrei Ungureanu: right?

Claudiu Leoveanu-Condrei: Yes. Monday works for me if it's fine with you.

Andrei Ungureanu: Yeah. Yeah. Let's do Monday. Monday it's okay.

02:13:25

Claudiu Leoveanu-Condrei: It's a good start of the week. I like it.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: Okay then perfect.

Andrei Ungureanu: Yeah, the weekend maybe go through

Claudiu Leoveanu-Condrei: Uh then same hours,

Andrei Ungureanu: stuff. Oh yeah, one thing now that we're mentioning uh I'm going on vacation uh between uh 18th of May

Claudiu Leoveanu-Condrei: right?

Andrei Ungureanu: and uh 30 of May. Uh so that means uh I mean I I'll try to join uh not sure I I will be able to but I will

Claudiu Leoveanu-Condrei: But you you said 18 and

Andrei Ungureanu: definitely Yeah.

Claudiu Leoveanu-Condrei: 31.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: Okay.

Andrei Ungureanu: So to those two

Claudiu Leoveanu-Condrei: So this basically Okay. Yeah. I mean it's fine.

Andrei Ungureanu: weeks

Claudiu Leoveanu-Condrei: Don't don't worry. It's like um we can take a a pause or whatever. Uh it's fine with me. I'm I'm not we should not rush it obviously. So or or we can read or I don't know.

Andrei Ungureanu: Yeah,

Claudiu Leoveanu-Condrei: But obvious take your time.

Andrei Ungureanu: maybe like uh the previous um lectures

02:14:29

Claudiu Leoveanu-Condrei: Mhm. I mean technically so if it's in 18 so now it's like what 30 of

Andrei Ungureanu: and

Claudiu Leoveanu-Condrei: April so I Okay. So I I don't think four is reasonable, right?

Andrei Ungureanu: Uh

Claudiu Leoveanu-Condrei: Because it's it's too So fourth uh fourth would be

Andrei Ungureanu: what?

Claudiu Leoveanu-Condrei: uh the next Monday.

Andrei Ungureanu: Ah okay. Next one they write. Uh

Claudiu Leoveanu-Condrei: So that's hardcore.

Andrei Ungureanu: yeah.

Claudiu Leoveanu-Condrei: So I assume then it's going to be 11 and then on 18 you leave.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: So up until 11 then we can I

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: mean we can we can try to do all of this. I don't know how you feel about it, but it's basic.

Andrei Ungureanu: That's what uh JPT said that we can

Claudiu Leoveanu-Condrei: No, no, no, no. CH GPT said that it's basically up until I

Andrei Ungureanu: very

Claudiu Leoveanu-Condrei: think um let me see. So, not that. Wait, what? Where was that?

Andrei Ungureanu: Yeah, it was here on the road.

02:15:49

Claudiu Leoveanu-Condrei: Oh here so basically it is session two. Yeah. So I I was right. So 2.7. So basically up until name name spaces and then uh these two are like I don't know how long they are but yeah.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: Okay. So basically it it is 2.3 to 2.7 is what GPT recommended which is anyway I think it's a lot.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: So it's a lot here. So we can go for the plants. I'm fine with that.

Andrei Ungureanu: uh actually I don't know it's it's uh too soon for me I

Claudiu Leoveanu-Condrei: You feel like it's you feel like it's too much

Andrei Ungureanu: think uh I'm not sure um but

Claudiu Leoveanu-Condrei: or

Andrei Ungureanu: if uh JB that is doable maybe. Although you're kind of brushing a lot

Claudiu Leoveanu-Condrei: I mean I'm honestly telling you this scares me.

Andrei Ungureanu: here.

Claudiu Leoveanu-Condrei: I don't know what the hell is this, man. It's like what the hell is this? So I I guess I I don't know.

02:16:59

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: I don't know how it's it's going to go. Uh, I mean I was looking obviously I was scrolling over over this. So we we get into more complicated things. So we define stuff like you know more complicated things. So I I don't know. I'm not sure how it's going to go but I'm fine with it. So we can go with with what it proposed and see how much we cover.

Andrei Ungureanu: Yeah. Yeah. Let's do that.

Claudiu Leoveanu-Condrei: If we cannot cover everything then I guess that's fine. It's still it's still fine. But yeah.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: Okay. So then next time let's let's do it like this. Let's do so next time would be then uh and I will send the invite starting then 11.

Andrei Ungureanu: Yeah. Yeah. It's

Claudiu Leoveanu-Condrei: Okay cool.

Andrei Ungureanu: okay.

Claudiu Leoveanu-Condrei: Then uh yeah talk to you. See you later. Then I will I will put this on the in case you want to take the file.

Andrei Ungureanu: Yeah.

Claudiu Leoveanu-Condrei: I will put it on on GitHub.

Andrei Ungureanu: Mhm.

Claudiu Leoveanu-Condrei: Okay, man. Well, cool.

Andrei Ungureanu: Keep in touch then.

Claudiu Leoveanu-Condrei: Thank you. Keep in touch.

Andrei Ungureanu: Yeah. Yeah.

Claudiu Leoveanu-Condrei: Take care.

Andrei Ungureanu: Thank you YouTube.

Claudiu Leoveanu-Condrei: Bye-bye.

Andrei Ungureanu: Bye. Bye. Bye.

Transcription ended after 02:18:26

This editable transcript was computer generated and might contain errors. People can also change the text after it was created.