



AI Can Write the Proof. Who Checks It? — Leonardo de Moura

September 29, 2026

Machine Learning Street Talk

Podcast Transcript

Revision: deb6a932

Topic headings include retention indicators: ● above average, ● average, ● below average (vs. similar YouTube videos)

Interested in licensing ReScript for your podcast? Get in touch with Tim Scarfe.

Contents

Overview	2
Cold open: the green checkmark can lie ●	3
Cathedral or bazaar: who controls Lean's core? ●	3
Why Lean's core stays small and protected ●	5
The Slack purge, Brandolini's law and the Lean FRO ●	6
The Collatz exploit: two kernels, two bugs ●	7
More kernels, reward hacking and safety by transparency ●	9
Kim Morrison, Claude and the zlib proof ●	12
Can we specify complex systems? ●	14
Specs change: proofs are cheaper to redo with AI ●	15
From Lean 1 to Lean 4 ●	16
Dependent types in plain terms ●	18
Lean 4's extensibility and Mathlib's growth ●	20
Mathlib as infrastructure: Formal Frontiers ●	21
Creativity, abstraction and nut-sniping ●	23
Breadcrumbs, not learning: what AI agents lack ●	24
Competence without comprehension, and verified guardrails ●	26
Is the human still the author? ●	28
AlphaProof, LLMs and why certificates still matter ●	30
What's next for Lean, and its legacy ●	31
How to start learning Lean ●	33
References	35

Overview

This episode is sponsored by Parallel. Parallel, where agents find answers: web search, extraction and deep research APIs built for AI agents. Start free with the Parallel MCP server and \$5 of credits every month: https://parallel.ai/mlst?utm_source=creator

Leonardo de Moura created Lean and co-created Z3. Tim Scarfe talks with him about how Lean escaped its original audience, why dependent types and Mathlib made it useful to working mathematicians, and what happens when formal verification leaves the lab. De Moura explains the small trusted kernel and independent checkers, and gives his account of the recent Collatz incident, in which a purported proof was accepted by both Lean's official kernel and Nanoda, apparently by exploiting a different bug in each.

The conversation then turns to AI-produced proofs and software. Claude agents rebuild zlib in Lean and the result is verified, yet the example exposes the loophole at the centre of the show: a proof only certifies the specification humans chose to write. De Moura describes proof search as a game, separates formal competence from mathematical understanding, and asks what authorship means when models can generate proofs at scale.

The closing argument puts human responsibility upstream: choosing definitions, judging abstractions, curating Mathlib, inspecting certificates, and deciding which problems deserve effort. He also reflects on Lean's future beyond its founder and gives a practical starting point for new users.

Cold open: the green checkmark can lie ●

Leonardo de Moura

00:00:00

This proof, I think, was a disproof. The Collatz conjecture was submitted as with a Lean proof. And the claim is it's accepted by the official kernel that comes with Lean. Wow. I mean, this is horrible. We strongly believe it was built by AI, exploits one bug in the official kernel, and this completely different bug in nanoda. Don't have patience for doing this kind of low level implementation, but AI has the patience for doing that. I think this is going to be a game changer going to better models. They keep doing more amazing things, but they keep doing stupid things at the same time. Boris Alexeev submits a proof to its 1,200,000 lines of Lean, the formal proof. Nobody wants to check the proof line by line. I'm Leo de Moura. I'm the creator of Lean, chief architect and cofounder of the Lean FRO, a nonprofit behind Lean, and also senior principal scientist at AWS.¹²

Cathedral or bazaar: who controls Lean's core? ●

Tim Scarfe

00:00:59

At some point, you actually ring fenced off like a a core part of the system, and you said, okay. This is locked. I'm in control of this. And there's interesting tension, isn't there? That's you know, a lot of creativity sometimes happens in the community. So there's different levels of adaptation. So there's the core thing. You know, for example, in Quake 3, the shooter, there are these bounce bugs. And an an an entire community evolved around speed running and bounce bugs and stuff like that. And similarly, in in in Lean now, there there's there's the core system and there and there's Mathlib and there's this entire community, being creative in different directions. The core thing adapts much slower and is and is controlled by yourself. But how do you think about that? Do do you kind of feel that you you do control an aspect of Lean, or does it have a life of its own?³

¹Lean — tool Leo introduces himself as the creator of Lean, the system at the centre of the interview.

²Lean FRO — organization Leo introduces himself as chief architect and co-founder of the Lean FRO, the nonprofit behind Lean.

³Mathlib — tool Tim contrasts Lean's tightly controlled core with the large community library built on top of it.

Leonardo de Moura

00:01:40

Mean Okay. Actually, this is a great question. This is a great question. No. No. I don't control the core system anymore. I mean, we have the FRO. There's a nonprofit. We have 20 people there. Each one of them are owning I I like I like this model. The developers in the core, they own parts of the system. It's their responsibility. They have ownership if they're going to take external pull requests or not. They they are the decision makers. But for the first 10 years, basically, I own a bunch of the system, Sebastian Ullrich, my colleague and friends that we built Lean 4 together. He own other parts of the system. But we believe I I strongly believe in the cathedral model for open source. Cathedral is like there's a central entity that decides what goes into the code base or not. This is bazaar where it's more open people. Everybody can contribute. And there was a lot of tension if the community community, the Lean community. Many people want sometimes to contribute to the core. Contributing to a library is different from contributing to the core system. The library many case, suppose that you're building a library for binary trees is self contained. Right? The changes you make there, can own that without impacting the rest of the system. But the core system, all the pieces are interconnected. They depends on each other. Another thing like a library a binary tree library may be missing some functions, maybe missing some theorems about these functions, but it's still useful. But a system full of holes is not very useful. Right? I mean, if there there are many pieces missing, many crashes, it's a more interconnected system. That's why I always insisted for the cathedral model for developing the core and creates tensions. But one one cool thing about that's that's amazingly extensible to enable people to add their own extensions without changing the core, and this became really popular with the community. I mean, many people implement today extensions to Lean without coordinating with us. They implement it's actually similar to the example you gave. They implement domain specific languages that hide the complexity of focus on a domain specific language for a specific domain, like distributed protocols. You can specify distributed protocols. You have a language that's optimized for that, hides all the complexity, and this can be done without even talking to us. Sometimes they demo things to me. They never met me before they they implemented this beautiful extension. Right? Yeah. I think it's great. It was also my side effects.

Why Lean's core stays small and protected ●

Tim Scarfe

00:04:55

And can you give me an example of that? So so that's you know when when when solving abstract problems, sometimes there are very that there are ingenious and creative approaches. And do do some of those really surprise you when you just see what people do?

Leonardo de Moura

00:05:10

Yes. Yes. I mean, I remember last year, Ilya Sergey is a professor in Singapore. He was with his student, Vova, and they demoed something to me. I said, why why these folks are demoing this stuff? This is not Lean. And it was really cool to demo. At the end, they said, no. No. This is Lean. This is an extension we implemented on top of Lean. I was really surprised. It was for software verification. They have a language that's much easier for people to verify software, especially if it's imperative software. Yeah. I was really blown. Yeah. They blew me away, I mean, with the demo.⁴

Tim Scarfe

00:05:52

It's part of the reason that you were so kind of protective over the core. I mean, if you think about it, the reputation of Lean is paramount because this is something that is gonna be used to verify software and mathematics and so on. So there's that. But it reminds me a bit of, you know, MATLAB or maybe Mathematica, has insanely optimized core libraries for doing linear algebra and so on, but someone could come along and they, with great power comes responsibility that they could just make a mess of it. They could do some operations and it's no longer vectorized and it's really slow and perhaps in Lean they could do something that actually affects the integrity of what they're doing. Is that part of it? So you wanna put a little bit more control

Leonardo de Moura

00:06:33

Yeah. There is that. I mean, there's also people adding something that's half-baked. They they it's like a lot of people like to have fun implementing software. Software is I like to say actually, one of my academic brothers. He says software is only fun if the software doesn't have to work. If it has to work, the fun disappears very quickly. And first issue, bugs like you mentioned, people introducing bugs. Second issue, someone submits a feature that's not fully im-

⁴Ilya Sergey — person Leo recalls a demo by Ilya Sergey and his student of a software-verification language built as an extension on top of Lean.

plemented. There are many holes. It looks like it works, but it doesn't. That's the thing. People may lock the design. They add a feature. It's complete. It's correct, but put constraints on how we can optimize the system or and now we have this feature. It may become popular and prevents us from doing other things that's more important because it's there. People use it, we can't easily remove it, and we're stuck with that. Another issue, it's super important to prioritize the developments. Going chaotic, implementing random features, and this happens if you start merging random PRs. It's really bad. It doesn't really scale. I think prioritization is super important, and this model where you keep merging random PRs does not work. This is a strong belief I have.

The Slack purge, Brandolini's law and the Lean FRO •

Tim Scarfe

00:08:07

So you're you're a depth-first search kind of guy. And when you build something successful that involves lots of other people, you have to become more like breadth-first search. And because there are so many things that are coming in every single day. And I remember there was a bit in Kevin's book where he said at some point, you just kicked a load of people off the Slack channel. And you said, look, I just need to I just need to focus now because it's just it was getting ridiculous.⁵

Leonardo de Moura

00:08:28

Yes. Oh, yes. Yeah. Yeah. I forgot about this story of the kicking. Yeah. Because at the points at that point, wow. I really dislike people that keep throwing ideas. I mean, they don't even they're having fun. I mean, it's different. I mean, for me, I was trying to build something that works. Some people are there in this, like, channel having fun throwing ideas on the wall. Right? I mean, it's cheap. I mean, you can throw ideas on the wall, and the other side has to keep telling you, no. This doesn't work. It's Something that's, for me, one frustrating scenario. You throw an idea. You you didn't spend 5 minutes thinking about it. Then I find a hole. I send back to you. Then you polish a little bit, send back. Now it takes 10 minutes to find the hole the next hole. It keeps taking exponentially longer to find the holes on the half-baked idea. For the person coming up with them, it's fun. I mean, but it's not fun from your receiver. Right? You have to do your work now. You spend time telling people why this is not a good idea, why it's not the time you should do it. Yeah. There was a point, yes, I kicked out. I mean, stayed

⁵The Proof in the Code — book Tim recalls a passage in Kevin Hartnett's book about Leo removing people from the Lean Slack.

only the core team that was really contributing to the project. Only people that were writing useful codes, the projects stayed.

Tim Scarfe

00:10:03

Yeah. It must be even worse now with generative AI. I mean, so what what you're pointing to there, it's called Brandolini's law, which is that it takes exponentially more work to debunk bullshit than create it.⁶

Leonardo de Moura

00:10:12

Well, fortunately, now we have the FRO. Right? The Lean FRO. We I have many colleagues, many people that's helped me, and my life is actually better now. I mean, I have for example, we had a serious bug this week. Joachim Breitner, he took care of. It's not just the bug. You have to take care of the community, explain to the community, communicate. He did all that to me. In the past, for me, would be a nightmare to have to do so many people, the amount of the noise, you know Joachim did that. And and we have many other people at the Lean FRO that help coordinating the community, communicating the road map. For me is much easier now. I mean, I I without the FRO, we we we would not be able to run the Lean projects now.⁷

The Collatz exploit: two kernels, two bugs ●

Tim Scarfe

00:11:12

Was the bug this week something to do with the Collatz?

Leonardo de Moura

00:11:15

Yes.

Tim Scarfe

00:11:16

Yes. I was gonna ask you about that. May maybe can can you set that up?

⁶Brandolini's law — other Tim names the principle that refuting nonsense takes far more effort than producing it, in the context of AI-generated ideas.

⁷Joachim Breitner — person Leo credits Joachim Breitner with handling the recent serious bug and communicating about it to the community.

Leonardo de Moura

00:11:19

Yeah. Yeah. Yeah. Basically, Lean is we have the notion of a small trusted kernel. Lean's massive, has zillions of bugs. Right? But to trust our results in Lean, you only need to trust the kernel that's much smaller. Right? And we also have external kernels. Right? The sequence of events I mean, you know, Joachim did all the digging. I mean, I'm glad he did. The sequence of events was like 1 week ago or or 10 days ago, a bug was reported in one of the main external checkers. We have many kernels. One of the main ones, external is called nanoda, is implemented in Rust. This bug was fixed immediately by the nanoda developer. And a few days later, this proof of the the Collatz conjecture this proof, I think, was a disproof of the Collatz conjecture was submitted with a Lean proof. And the claim is it's accepted by the official kernel that comes with Lean and by the main external kernel, nanoda. Right? It would be yeah. Wow. I mean, this is horrible. Right? Then the first reaction we had is he said, nanoda was not accepting the proof. Then Joachim said, okay. This guy is exaggerating. nanoda does not accept. But then he said, oh, nanoda from 10 days ago does not was accepting the Proof front. And then, actually, do it should it the exploit and we build we strongly believe it was built by AI, exploits one bug in the official kernel, and this completely different bug in nanoda. I fixed the bug in the official kernel, and I found strange the example because there was a bunch of weird terms to create a hash collision that was completely irrelevant. I said, why did they put this stuff there? It's to pass nanoda I mean, to to exploit the bug in nanoda. Then Joachim asked whether the LLM look at they asked the LLM to look at the PR that was submitted to nanoda. They said the bug reporters said that was a coincidence, but they could not discuss the the possibility the LLM by itself⁸⁹¹⁰¹¹¹²

Tim Scarfe

00:14:09

look

⁸nanoda_lib — tool An external Lean kernel written in Rust that Leo says was also accepted by the purported Collatz disproof.

⁹nanoda_lib pull request 22 — tool The nanoda_lib fix pull request ('more struct and inductive checks'), merged the day before the Lean kernel issue was filed; it fits the earlier nanoda fix Leo describes.

¹⁰CollatzLean — tool Repository of the purported Collatz disproof that, in Leo's account, passed Lean's official kernel and nanoda.

¹¹Lean issue 14576 — tool Upstream issue titled 'Kernel accepts wrong-structure projections, allowing an axiom-free proof of False', which corresponds to the official-kernel bug Leo describes.

¹²Lean pull request 14577 — tool Leo's fix for the official kernel ('missing check at kernel inductive declaration'), matching the fix he mentions.

Leonardo de Moura

00:14:10

at the nanoda repository, saw the PR and crafted these exploits. Yeah. This we have to before coming here, I was talking to Joachim and Sebastian on how to improve, what we can do to improve to make it even more secure because this is going to keep happening. People are AIs are really good exploiting and soundness, bugs in the in the kernels. We we're throwing ideas like, oh, we should have bounties on people for people building kernels that are bulletproof. They get some cash if the kernel cannot be attacked for more than x months or another idea is to read to simplify the kernel to make it simpler to verify. Another thing is to support Mario Carneiro is working on trying to prove the kernel is correct, trying to assist him or to accelerate the developments of his proof. I mean but yeah. But coming here, I was talking to Joachim about how to to improve the situation in this for avoiding this kind of thing from happening again. But there are some concrete steps, like one concrete step that Rob did. We have a tool called comparator. That's for that's basically comparator exports the proof because some people can use metaprogramming, Lean extensibility features we discusses to inject false proofs. You can even call C function corrupts the memory, for example. But you can you have comparator that we will export the proofs. We check them in a sandbox. And one lesson is to for comparator always to download the latest version of nanoda. Right? And so we don't have to manually upgrade nanoda. Someone submits the bug fix to nanoda, it will be used immediately by comparator. And comparator would have rejected, right, if you had done that. And we are doing that. Yeah. But we have lots to do. We want to prove more things about the kernel, about the code generator. That's another place we have to improve. Yeah. But this was I could not imagine the situation without the FRO to help us. Yeah. And I¹³

More kernels, reward hacking and safety by transparency



Leonardo de Moura

00:14:10

at the nanoda repository, saw the PR and crafted these exploits. Yeah. This we have to before coming here, I was talking to Joachim and Sebastian on how to improve, what we can do to improve to make it even more secure because this is going to keep happening. People are AIs are really good exploiting and soundness, bugs in the in the kernels. We we're throwing ideas like, oh, we should have bounties on people for people building kernels that are bulletproof. They

¹³Lean comparator — tool The tool Leo describes that exports proofs and checks them in a sandbox, guarding against metaprogramming tricks and kernel bugs.

get some cash if the kernel cannot be attacked for more than x months or another idea is to read to simplify the kernel to make it simpler to verify. Another thing is to support Mario Carneiro is working on trying to prove the kernel is correct, trying to assist him or to accelerate the developments of his proof. I mean but yeah. But coming here, I was talking to Joachim about how to improve the situation in this for avoiding this kind of thing from happening again. But there are some concrete steps, like one concrete step that Rob did. We have a tool called comparator. That's for that's basically comparator exports the proof because some people can use metaprogramming, Lean extensibility features we discusses to inject false proofs. You can even call C function corrupts the memory, for example. But you can you have comparator that we will export the proofs. We check them in a sandbox. And one lesson is to for comparator always to download the latest version of nanoda. Right? And so we don't have to manually upgrade nanoda. Someone submits the bug fix to nanoda, it will be used immediately by comparator. And comparator would have rejected, right, if you had done that. And we are doing that. Yeah. But we have lots to do. We want to prove more things about the kernel, about the code generator. That's another place we have to improve. Yeah. But this was I could not imagine the situation without the FRO to help us. Yeah. And I¹⁴

Tim Scarfe

00:16:44

wonder whether it's a similar thing, but there was an incident where the, you know, GPT 6, let's say, hacked Hugging Face recently and the models are becoming, reward hacky, reward seeking. And I don't know whether you think, I mean, maybe you implied this was a similar situation, but, you know, there was a green check basically. So it it looks like this proof was valid. And sometimes these these proofs are quite inscrutable. They're difficult for humans to read. So in in principle, the LLM could have done reward hacking and rather than kind of doing the spirit of what we wanted to do, it it didn't. And what OpenAI did was they changed their guardrails so that, you know, when the model generates some code, they don't just look at the code in isolation, they they look at the entire trajectory, the reasoning trace, why did this code come to be? So they're trying to sort of like disentangle the, the intention behind this thing versus the intention of the developer who did the prompt. So do you see a similar thing here that we might we might need to have almost separate adversarial systems kind of evaluating what's happened?

¹⁴Lean comparator — tool The tool Leo describes that exports proofs and checks them in a sandbox, guarding against metaprogramming tricks and kernel bugs.

Leonardo de Moura

00:17:44

Well, having multiple kernels is a way to accomplish that. Right? If have independent kernels implemented by different people using different programming languages, it strengthens the situation. Right? The fact that we have the official kernel and nanoda, we want to have more. I mean, the one from Mario, this is called Lean4Lean, is is going to be special because he's proving it should be correct. This way, will be not just another kernel, but a kernel that has been proved correct. That will be a big deal when it arrives. I mean, we can use Lean4Lean today, but it's not fully proved. It proved many parts, not the parts that has to exploit. If he had to prove it, he would had found the bug before it was exploited. So having multiple kernels, having some of them prove it's correct makes the story really bulletproof. Then there is the next level you may ask, okay. You prove the kernel correct, but you're compiling with some compiler. The compiler maybe has a bug. Right? And you prove it's correct, but what's running is not what you prove it. Right? Then then the next step, let's prove the compiler correct. Then they may say, oh, the hardware may have a bug. Right? So say, oh, well, okay. Let's ask Intel to publish formal specs. That's how they are saying this is the official formal specification for the microprocessor, same for AMD and so on. Yeah. The movement here is always reducing what you have to trust. Right? And this is going to be important. Right? If one day we want to have massive results, that's we don't have to spend time checking manually for exploits. It's really important that we can fully trust these mechanical proofs. Right?¹⁵

Tim Scarfe

00:19:44

I wonder whether this is an example of you because obviously, you're an open source guy. Yeah. But I I mean, let's look at the you know, there's, there's an AI benchmark called ARC-AGI-3. And what they do is is they have, like a semi private dataset and a private dataset. And, of course, what they're trying to do is is to stop the models from, you know, kind of hacking hacking and data leaking from these data sets. So when they test it on the private data set they know that the models haven't really seen that before. Do you think this is one exception that you might make to your philosophy that you know possibly we might need to have a separate independent private kernel that couldn't possibly have been hacked. So we've always got the joker in the pack that we can test it with the private kernel.¹⁶

¹⁵Lean4Lean — tool Mario Carneiro's kernel, which Leo says is being proved correct and would be more than just another independent kernel.

¹⁶ARC-AGI-3 — tool Tim uses the benchmark's semi-private and private datasets as an analogy for keeping a private kernel.

Leonardo de Moura

00:20:23

I I don't like safety by obfuscation by hiding. I think we should be always be transparent. I mean, people because otherwise, I mean, you're saying, look. I have a really good kernel here that's I'm not showing anybody that checks the results. I I think for transparency, for trust, you have to be fully transparent with these kernels. Right? We can improve the situation by proving these kernels correct, by adding more kernels implemented by different people, but everything transparent. I mean, I I I it goes against everything our community believes to safety by secrecy.

Tim Scarfe

00:21:05

Speaking of checking where your answers come

Leonardo de Moura

00:21:07

from,

Tim Scarfe

00:21:08

today's episode is sponsored by Parallel, where agents find answers. Most engineers today, they closely follow new model releases, but they don't pay attention to their agents' most important tool, web search. Parallel develops enterprise grade infrastructure for agents to retrieve high quality context from the web. Their core products are a suite of APIs for retrieving high quality information from the web with Pareto optimal quality, cost, and speed. So whether you're working on voice agents that need 200 millisecond latency, chatbots that balance speed, depth, and quality, or long horizon agents to do thorough overnight research for you, Parallel is the single platform for all of your agentic research. Get started for free at parallel.ai/mlst Apparently, Kim Morrison used a Claude agent to produce a Lean implementation of the compression format zlib and and prove some specific properties of it. So what exactly did they manage to prove?

Kim Morrison, Claude and the zlib proof ●

Tim Scarfe

00:21:08

today's episode is sponsored by Parallel, where agents find answers. Most engineers today, they closely follow new model releases, but they don't pay attention to their agents' most important tool, web search. Parallel develops enterprise grade infrastructure for agents to retrieve high quality context from the web. Their core products are a suite of APIs for retrieving high quality information

from the web with Pareto optimal quality, cost, and speed. So whether you're working on voice agents that need 200 millisecond latency, chatbots that balance speed, depth, and quality, or long horizon agents to do thorough overnight research for you, Parallel is the single platform for all of your agentic research. Get started for free at parallel.ai/mlst Apparently, Kim Morrison used a Claude agent to produce a Lean implementation of the compression format zlib and and prove some specific properties of it. So what exactly did they manage to prove?¹⁷

Leonardo de Moura

00:22:12

Oh, yeah. Yeah. This when Kim started working on this project, I I thought, wow. This is it felt hopeless. No way. I I I thought it was at the beginning of the year. I to me, I I thought AI is not good enough for that. I mean, we'll not be able to succeed at that. But the AI managed to translate from C to Lean, the code, then managed to keep fixing the Lean implementation till it passed the test suite that comes with the C version. And then it proves that for any compression level, for any data, if you compress the data and then decompress, you get the original data back. This felt completely out of reach for AI at the beginning of the year, but AI managed to do it and didn't stop there. Kim asked them to optimize the codes, especially for this kind of program. Lean right now is not the kind of language for array manipulation. Lean Lean has been optimized for programs that are manipulating trees because I mean, you're you're manipulating terms. Right? You're manipulating trees internally. You have the syntax trees, the expression trees. Lean is really good for this kind of program, but not for arrays. It's very in my mind, this will never be competitive. Right? But turns out that Kim asked the AI to keep trying really you can go wild, keep optimizing the code. You you you have to keep proving the the property. You cannot break the property. And it beats our implementation in Rust. And Rust is a really efficient language for array manipulation. Then for us is is really unexpected. Right? I mean, first, the sign imagine now if we improve our array manipulation for Lean, how far we can go. There's so many possibilities. Right? We can ask now the AI. Suppose that we give the semantics of x86, the microprocessor, the the assembly. You can ask, I think very soon, we're going to be able to ask the AI to prove writing assembly, exploiting all these fancy instructions, but proving you you have to prove the property. Right? We don't have patience for doing this kind of low level implementation, but AI has the patience for doing that. I think this is going to be a game changer. Right? Last year, I thought this would be impossible, but now it's in reach. I mean, this experiment shows there is a path there. These AIs will keep keep getting better and better at these kinds of task because there's a strong

¹⁷lean-zip — tool Kim Morrison's Lean zlib implementation, produced with a Claude agent, which Tim and Leo discuss as an AI-written program with proved properties.

signal telling them you're in the right path. Right? You have to still prove the theorem. Right? You you you can't cheat. Yeah. I think the sky is the limit.

Can we specify complex systems? ●

Tim Scarfe

00:25:25

Yeah. It's so tantalizing. I'm fascinated by the notion of specification driven software engineering. And we've had similar kind of periods in AI, we had expert systems for example, and we had the concept of a knowledge engineer or an ontologist. And there are certain types of things that we are, we just understand we can be computer scientists and we can reduce them down to their essence. But a lot of software engineering in my opinion, because you know, the reason we have agile is the world is very complicated. And what we do is we take small steps towards acquiring a specification. And and, you know, after thousands of iterations, we have those moments and we sort of compress and collapse some of it down. So I I suppose, do we have a chicken and egg problem that when we're working with very complex systems, we can't really yet specify or if we can, we can only specify parts of it.

Leonardo de Moura

00:26:17

Yeah. Yeah. Yeah. Yeah. I agree that it's very common sometimes. You know, some properties the system should have, but not all of them. You're still engaging with the users about which properties the system should have. Believe it or not, this happens with Lean all the time. The kernel is clear what the kernel should do, but the whole system, there are many components. I mean, there are many for example, these tactics I mentioned, the moves. What a move should do, we implement something the users is really common for the users saying, oh, no. No. I want more or I want less. No. It's doing too much. I I want more control. I want this flag. I want to have a way to to control where the proof is going more it's a back and forth. The spec there is not really clear. Right? Yeah. I think you're right in many cases, the spec's not clear, but it's important to talk to start documenting the properties you know they should hold, talk to your customers about them. Another thing to keep in mind is that a high level program in a very high level programming language like Lean without any optimization can be viewed as as a specification. Right? You can say, look, this is what I want to compute. It's not super efficient, but this is what I want to compute. And we can ask the AI optimize this function for me. I implement the simplest possible, right, idea, right, without any fancy data structures, clever algorithms. It does what I want. Now it's it's AI's job to optimize it.

Specs change: proofs are cheaper to redo with AI ●

Tim Scarfe

00:28:04

Yes. I suppose that specification acquisition engineering can be made. We can streamline that process. We can make it interactive, have lots of feedback and so on. But if we did move to a model where we were specification-first, and we have globally distributed deployed software, would we then have a specification updating problem? So we have some software deployed on the previous specification, and then, like, over here, we're doing some development and some other folks are updating the specification. How do you think that kind of system could work?

Leonardo de Moura

00:28:37

Oh, yeah. For sure. I mean, when you update the specifications, the code that was generated may be invalidated. The proofs that the code meets specification need to be invalidated. In the past, formal verification, this was a big deal. Right? People would say, oh my gosh. I mean, he spent all this time proving the implementation is correct. Now these folks want to add this fancy optimization. Now I have to redo many of these proofs. It's not rocket science, but it's a lot of work. But turns out AI is really good at updating these artifacts for us. Right? But before AI, this was a really big deal. Right? Especially for the safety critical systems where formal verification was used in the past, they were for application domains where the spec was very clear, like cryptography the specs, very clear. Right? They find a micro kernel. People put lots of thoughts in the micro kernel before they even started verifying the projects. All these famous verification projects like CompCert, seL4, people put a lot of thoughts on the spec. Right? They in some cases, they had to updates along the way and was painful as hell to do it. But now because we have AI to update the proofs, the artifacts, is way less painful. Right? I think there's still I I can see in the near future, right, AI is too expensive. Right? People would say, wow. I'm updating the spec. Now I have to waste zillions of tokens updating everything. Right?¹⁸¹⁹

Tim Scarfe

00:30:23

It is it is really exciting, though, because if I remember correctly, Kevin said in his book that I think Z3, that actually solved an incredible number of bugs in

¹⁸CompCert — tool Leo cites CompCert as a famous verification project where people thought hard about the specification.

¹⁹seL4 — tool Leo cites seL4 alongside CompCert as a famous verification project with a carefully considered specification.

Windows 7 before it even got released. So it was it was life saving. But, but we we you know, most of the time this kind of technology is used for very high stakes software. But do do you agree though in principle that in the future just everyone will be doing this? Just every developer will will will be, you know, building this kind of software?²⁰

Leonardo de Moura

00:30:49

They may not know they're using formal verification, but we'll be there in their pipelines. Right? They are going to be writing properties, and the properties are going to be verified. The code is going to be updated. I mean, there will be proof showing the codes still computes the same thing. People submit PRs. They're going to get proofs whether the PR is going to is violating some of the properties in their test suites or not statically. Yes. I think even if they don't know, they're using formal verification. They will it will be there.

From Lean 1 to Lean 4 ●

Tim Scarfe

00:31:27

So tell me about the evolution of Lean. So, the first prototype, 2013, and then, you know, Lean 2 2015, Lean 3 2017, and then Lean 4 took a very, very long time to to build. Tell me about that whole story.

Leonardo de Moura

00:31:39

Oh, wow. Okay. Lean the first Lean is even called 0.1. I learned by doing it. I read many papers about dependent type theory. I I used some of the other systems. But I've always feel it gets different perspective when you implement something. There's so many nuances, details that you only now say, oh, wow. This this is really important, I mean. And and Lean 1, I also tried to make it very small. The kernel was was too small and was unusable. Maybe only Jeremy, I think, Floris, one of his students also uses Lean 1, but they were heroes. It was a horrible system. I mean, it was more like a learning an artifact I use to learn dependent type theory. Then Lean 2 was for real, but I think Lean 2 had, like, maybe between 50 and 100 users. Very small. And we one learning a a actually was when I was visiting MIT. Adam Chlipala is a professor there. He he told me, look. You need a real tactic framework. The tactic framework is the collection of moves. And one thing he want is that this move should be extensible. Right? So you should be able to add new moves as a user. And this was the main learning from Lean 2 to

²⁰Z3 — tool Tim recalls a claim that Z3 found many bugs in Windows 7 before release; Leo later calls Z3 a black box that AI cannot guide, unlike Lean.

3. We added a new framework for writing these moves. The users could write it, and they could write in Lean itself. We did not have to use a different programming language. And this became very popular. And then Mathlib started. Things started growing very rapidly. And it was clear that Lean 3 had limitations. I mean, in terms of scalability, the system was the tactics you could extend yourself, but that was it. Some people managed to do linters. The Mathlib community figured out how to hack Lean, to write linters in Lean. But we want to make it fully extensible. And in 2018, Sebastian Ullrich and I said, we are going to implement the whole thing. At that point Detail. I'm not a programming language person. I'm automated reasoning person, formal verification person. And Lean was basically my first programming language I ever implemented. And there were the syntax was ugly. I mean, there was many weird things in the implementation, and we'd say, okay. Now we know how to do it. Let's do it properly in Lean 4. Then we start Lean is going to be implemented in Lean, was much harder than we expected, Took us from beginning of 2018 to the end of 2020 to get Lean compiling itself, but was really exciting to get there. And I don't think we sufficiently explained dependent type theory. So maybe you can do that. But there's this relationship between the expressability space of the problem descriptions and how sophisticated the kernel needs to be. And I think early on you were quite bullish about you want the kernel to be kind of like quite small and optimized and you had to admit more complexity that Yes. Yes. But this was in the design phase. I mean, it was before even Lean 0.1 started. I started implementing Lean 0.1, it was dependent type theory. Right? It was always all Lean versions are based on dependent type theory, but was more like when we were designing, deciding what's what to implement, what's at the design level, this discussion about dependent type theory. To explain to the audience what dependent type theory is, I think one example is to say, can have in dependent type theory. You can have a structure like that has 2 fields, x and y . They are supposed integers. And you can have a third field that says the type of this field is x is greater than y . It's a proof. You you have to fill this field with a proof that's evidence that x is greater than y .²¹²²

²¹Adam Chlipala — person Leo credits Adam Chlipala at MIT with the advice that Lean needed a real tactic framework.

²²The Lean 4 theorem prover and programming language — paper Paper describing Lean 4, the rewrite in which Lean is implemented in Lean itself, a story Leo tells here.

Dependent types in plain terms •

Leonardo de Moura

00:31:39

Oh, wow. Okay. Lean the first Lean is even called 0.1. I learned by doing it. I read many papers about dependent type theory. I used some of the other systems. But I've always feel it gets different perspective when you implement something. There's so many nuances, details that you only now say, oh, wow. This this is really important, I mean. And and Lean 1, I also tried to make it very small. The kernel was too small and was unusable. Maybe only Jeremy, I think, Floris, one of his students also uses Lean 1, but they were heroes. It was a horrible system. I mean, it was more like a learning an artifact I use to learn dependent type theory. Then Lean 2 was for real, but I think Lean 2 had, like, maybe between 50 and 100 users. Very small. And we one learning a actually was when I was visiting MIT. Adam Chlipala is a professor there. He he told me, look. You need a real tactic framework. The tactic framework is the collection of moves. And one thing he want is that this move should be extensible. Right? So you should be able to add new moves as a user. And this was the main learning from Lean 2 to 3. We added a a new framework for writing these moves. The users could write it, and they could write in Lean itself. We did not have to use a different programming language. And this became very popular. And then Mathlib started. Things started growing very rapidly. And it was clear that Lean 3 had limitations. I mean, in terms of scalability, the system was the tactics you could extend yourself, but that was it. Some people managed to do linters. The Mathlib community figured out how to hack Lean, to write linters in Lean. But we want to make it fully extensible. And in 2018, Sebastian Ullrich and I said, we are going to implement the whole thing. At that point Detail. I'm not a programming language person. I'm automated reasoning person, formal verification person. And Lean was basically my first programming language I ever implemented. And there were the syntax was ugly. I mean, there was many weird things in the implementation, and we'd say, okay. Now we know how to to do it. Let's do it properly in Lean 4. Then we start Lean is going to be implemented in Lean, was much harder than we expected, Took us from beginning of 2018 to the end of 2020 to get Lean compiling itself, but was really exciting to get there. And I don't think we sufficiently explained dependent type theory. So maybe you can do that. But there's this relationship between the expressability space of the problem descriptions and how sophisticated the kernel needs to be. And I think early on you were quite bullish about you want the kernel to be kind of like quite small and optimized and you had to admit more complexity that Yes. Yes. But this was in the design phase. I mean, it was before even Lean 0.1 started. I started implementing Lean 0.1, it was dependent type theory. Right? It was always all Lean versions are based

on dependent type theory, but was more like when we were designing, deciding what's what to implement, what's at the design level, this discussion about dependent type theory. To explain to the audience what dependent type theory is, I think one example is to say, can have in dependent type theory. You can have a structure like that has 2 fields, x and y . They are supposed integers. And you can have a third field that says the type of this field is x is greater than y . It's a proof. You you have to fill this field with a proof that's evidence that x is greater than y .^{23 24}

Tim Scarfe

00:36:11

You

Leonardo de Moura

00:36:12

can view it as an invariance. It's it's called dependent because of the type of something can depend on the value of something else. You're depending on the the type of this x this third field depends on the value of x and y . Right? That's why it's called dependent. You can simulate invariants. In functions, you can have, like, a function that takes x and y . And the third arguments, y should not be 0. You have to to call this function, you have to provide evidence that y is not 0. It's almost like a precondition. That that's the beauty of the dependent type theory. A simple idea, you can capture preconditions. You can simulate post conditions. You can have invariants. You don't need to invent all these concepts. Everything is really minimalistic, but it complicates your type theory because you have these dependencies. You have to manage them, but it was worthwhile. I mean, it was worthwhile.

Tim Scarfe

00:37:17

Yes. So so so the kernel had to be significantly more sophisticated to to support that. What can what can you do in in Lean 4 that you can't do in Lean 3?

²³Adam Chlipala — person Leo credits Adam Chlipala at MIT with the advice that Lean needed a real tactic framework.

²⁴The Lean 4 theorem prover and programming language — paper Paper describing Lean 4, the rewrite in which Lean is implemented in Lean itself, a story Leo tells here.

Lean 4's extensibility and Mathlib's growth •

Leonardo de Moura

00:37:26

Lean 4 is fully extensible. Right? I mean, everything's implemented in Lean. You have access to all the internal data you when you're writing Lean your Lean declarations, you can in the same file, you can write code that access the internal representations of everything. Many the the first AI research labs groups, they for them was a big deal. They could extract training data, right, from the internals of Lean. They could write Lean code that extracts information about all the proofs that are in Mathlib. They could manipulate these objects. This was a big deal for them. All these extensions I I told you about, like, like Velvet, Veil, they are all using this extensibility we have in Lean.²⁵²⁶

Tim Scarfe

00:38:21

And I saw from from your your slides you presented this week. So I I think that we've now reached the inflection point where there is as much Lean 4 code in Mathlib as there was Lean 3 code.

Leonardo de Moura

00:38:32

Oh, yeah. Way more. Way more. Way Yeah. Way more. Mathlib 3 stop at 1,100,000. In Mathlib 4, we have 2,400,000.

Tim Scarfe

00:38:42

One interesting thing though for me is I I think Kevin Buzzard said something along the lines of it will just keep growing forever. So it'll it'll it's it's always gonna be at I think did he say it's always gonna be at naught percent? And I think that's quite instructive about the nature of mathematics because it's something that just keeps evolving in many, many different directions. So I suppose it's tempting to think of mathematics as being this kind of this distilled thing that we just discover and and there it is. But is is that is that not the way it is?²⁷

²⁵Veil — tool The other extension Leo names, a Lean framework for verifying transition systems such as distributed protocols.

²⁶Velvet — tool One of the extensions Leo names as built on Lean 4's extensibility.

²⁷Kevin Buzzard — person Tim recalls Kevin Buzzard's remark that the formalised share of mathematics will stay small as mathematics keeps growing.

Leonardo de Moura

00:39:12

No. That's going to keep growing forever? Yes. I mean, there will be always new math being invented, new structures, defined, new mathematical objects. There will be always new things. There's a debate whether everything should be in Mathlib or not. At the beginning, for sure, at the beginning, it was amazing to have everything in a single library because the library you have many concepts in mathematics. For example, you have rings. There's only one ring in the Lean Mathematical library. The library's coherence I mean, it's really common in other programming languages to have many implementations of the same thing. And things are incompatible. If you want to prove things about your code, it's important to have this uniform view. And Mathlib also they also another great thing they decided. We we we will always shoot for the more general, the most general description of something that was also great. But there are limitations about the size. Right? Mathlib is 2,400,000 lines. Imagine if it's 50,000,000 lines. Now it's starting to get the size of the Linux kernel. Right? I mean or big projects we have that's like Firefox, size of these projects have teams of people just maintaining the build system. I mean, everything is more complicated. Sometimes they have their own version specialized version control system to handle the complexity. Now there's a lot of discussion on how to get the mathematical library. Maybe Mathlib is going to be a collection of libraries. You have the main ones like the standard library for math and many satellite ones, all of them maintained, all of them coherence, but with different leaderships. I mean, you don't have the same group of maintainers for each one of them. All these discussions are happening. I think they're healthy. Yeah. But people predict if you want to do if you have to have all mainstream math, the library has to be 100,000,000 lines. I mean, if you want to be able to formalize arbitrary research-level mathematics papers, the library has to be 100,000,000. Some people in the community have these estimates. I mean

Mathlib as infrastructure: Formal Frontiers ●

Tim Scarfe

00:41:44

There is one of these things though that some knowledge is more distilled to another knowledge. So rings for example, you know, you learn about them at university. So so this this is the the distillation process has been going on for hundreds of years. Yes. So it I would say it's high up the abstraction mountain. We can kind of, you

Leonardo de Moura

00:42:01

know,

Tim Scarfe

00:42:01

maximally compress it. But then there are all of these strands of discovery, just like when we write software. So the reason we refactor software is because when we started writing the code, we haven't yet understood the domain abstractly enough. And now we do. So we refactor, we distill, we compress. So is it a similar thing in Mathlib that that some parts of it are highly evolved and other parts of it are quite adaptable and we're still figuring things out? How do you kind of Yes. Yes. Yes. Both things have it's exactly

Leonardo de Moura

00:42:28

like software. You should you're thinking the right way. It's exact there are things that people thought a lot, and it's crystal clear how it should be, and there are things that are in flux. Right? But one cool property we want to have and there's a project now from the Mathlib initiative called Formal Frontiers. They want humans plus AI to formalize to fill all the gaps in Mathlib and enable any research-level math paper to be people to be able to formalize the paper. To put in perspective for software for software developers, this is equivalence, like, you want to develop some piece of software. And today, people use hundreds of crates, right, if they're developing Rust. But imagine that you half of these crates were missing, and you would say, oh my gosh. Now I have to spend time building the infrastructure for building my application. It feels like, you say, like something that should take 1 month, takes years because you have to build all the infrastructure. Same thing for math. If you try to formalize a paper, people in the community will tell you is doable or not. Because if there are so many components missing, it's going to be a gigantic task. Right? But if all the components are there, you only need to formalize your paper, your contribution, not the background's material. It's like, if you have all crates there, the rest is easy. It can focus only on my application. I don't need to build a JSON library. I don't need to build B-trees. I don't need you you get it. Right? So what I mean? But this is exactly like software.²⁸

²⁸Mathlib Initiative — organization Leo describes the initiative's Formal Frontiers project, which aims to help humans and AI fill Mathlib's gaps.

Creativity, abstraction and nut-sniping •

Tim Scarfe

00:44:15

I've got a theory that creativity is about deep understanding. So when you understand something deeply, you can kind of extend that lineage. And there's a really interesting thought that sometimes, you know language models, they, they can acquire all of these abstractions. I love using the example of linguistics. So they can, skin, sentences like an onion, even ones that we can't skin and they explain it to you using concepts from PhD linguistics textbooks. And it's a similar thing across the board. They know about rings and but I have a theory that even though they have the abstraction in a sense they don't understand it because you know, the path that led to rings, you know, it's like a big evolutionary tree. Understanding is about knowing the difference between what's possible and what's not possible, counterfactuals, how could it be different and so on. So if you start with something that is very, very distilled and you don't understand how it came about, then it's difficult to be creative. So what we see with language models is they start quite low down the abstraction mountain. So so these these are still a form of understanding, but they're deep concrete understandings. They're still quite fractured. They're not very factored, but they have more degrees of freedom. So when you look at this proof from the new GPT model, disproving this, you know, unit distance problem, What you see is insane verbosity. It's just kind of going around. It's doing quite surprising things. So isn't it interesting that from a creativity point of view, what we think is most important. So these really deep factored abstractions might not actually be how the AI models are creative.²⁹

Leonardo de Moura

00:45:58

Really hard to tell. I mean, I I keep thinking about this. They surprise me in both directions, sometimes doing amazing things. We had a bug, not not in the trust this is a simpler bug in the top level that's not doesn't need to be trusted. But nonetheless, this is very complex piece of software there. I think very few developers, Lean developers understand these parts. They I understood perfectly. Perfect. The explanation was spot on. I was blown away, I said, how come they understand so much about internal implementation? Right? So I was blown away. But on the other hand, sometimes it makes idiotic mistakes. I mean and it's not it it I I I don't see the behavior changing. I I keep going to better models. They keep doing more amazing things, but they keep doing stupid things at the same time. But we do stupid things too. Right? So maybe

²⁹A new result on unit distances — other Tim points to a proof from a new GPT model disproving a unit-distance conjecture, and Leo later notes its 1.2-million-line Lean formalisation.

they are. Today, I feel like they are really good for nut, tasks that we call nut-sniping. Yeah. Like, if you have a hole in a you have a statement, you want a proof, If you don't need to create a new theory, you are just combining the pieces that exist, they're a bit feels like they're unbeatable by humans. They are way better. You ask to implement micro optimizations to a piece of code. Also, they are amazing doing that. But if you ask them to come up with a new technique for implementing something, they fail miserably. I I feel like maybe I'm not a machine learning expert. Maybe you tell me one conjecture I have is, like, these things are like you don't know. They are, like, waking waking up. I mean, with no only if the training you show them a problem, and they go for it. I mean, they read all the literature during the training. They were during the the training, they saw all the literature. They have a bias towards exist solutions that people use. If you can solve this problem using a collection of tricks from the literature, they will do it. They will crush it. But if you need a new trick, it feels like they fall short. Of course, in the future, you may have AI agents that keep updating their weights and learning. You tell me. I mean, what what does this match your understanding? I'm not an expert.

Breadcrumbs, not learning: what AI agents lack ●

Tim Scarfe

00:48:46

I'm I'm as excited as you are, but there is this weird thing that they have lots and lots of intelligence, but they still clearly need to be directed. So, the best case scenario is you're solving a specific problem and you can hill climb towards it, but you still need to give it a kind of aspect ratio. I use the analogy of shining a flashlight into embedding space. The flashlight has a beam. So as an expert, what you're doing is you're saying, okay, well, this perspective is quite interesting. You know, here are some sample seeded solutions. Here are some tricks that you can use. And here is a clear optimization objective. And in that case, it's miraculous.

Leonardo de Moura

00:49:22

Yes.

Tim Scarfe

00:49:22

But there's still some human taste and expertise required. But the thing that seems to be missing for me is it doesn't have powers of abstraction. So for example and this might actually be a good thing from a creativity point of view because what it's doing is it's just finding a path through these low level concrete

fractured representations and it's getting to the solution. But what we need to do after it's done that adaptation and after it's found a path it needs to reflect on it in a meta sense and it needs to say how can I compress this? Now maybe I'm being unfair, maybe harnesses do this to a certain extent but when harnesses have a memory system it's still spaghetti. It's still the human supervisor that says okay well you're doing it now but you need to optimize it you need you know maybe if you think about it in this different way boom If they could do that on their own, that would be incredible. Yes. And maybe they are to some extent.

Leonardo de Moura

00:50:14

It's very limited. I feel like, you know, machine learning more than I do, but it to me feels like they wake up. They leave breadcrumbs for the next one that we wake up. It just reads all the breadcrumbs and do the next step. They're not really the memory are like breadcrumbs for the next one. They're not changing their weights in any meaningful way. It's almost like and there's so much you can do. You keep leaving bread breadcrumbs for the next instance that we wake up and reach all the breadcrumbs and take the next action. But eventually, the the the diversion, that's where the humans help a lot, keeping them in the right direction. Right? I mean, does this match your

Tim Scarfe

00:51:03

I I have a a feeling that the reason why we can do this abstraction is because we are we're in the world and we could have done different differently. You know, we we have agency, we can, know, towards doing a goals, we can actually kind of subject to the constraints that we are, you know, that we have applied to us. We can try different things. An example of this is if I hire a video editor. So I've been evolving video editing skills for 5 years. And I could say to the video editor, okay, I'm going to optimize your process because I've learned all of these important lessons. So, always use a tripod, don't even bother holding it by hand because you're gonna have lots of problems with that. So, let's just cut, we're gonna prevent that problem from even happening. And I can lay a load of constraints and they would have competence without comprehension. Right? But they would always be pushing back at me. And they would say, oh, you know, I wanna try this and I wanna try that because they don't know why those constraints are. I understand it because I understand the space of possibilities and I know why those constraints came to be.

Leonardo de Moura

00:52:03

You know, I I agree. It's like you learn by experience. Your experience taught you that. And they are learning from your experience. They do not experience themselves. Yeah. For example, SMT. I mean, you you mentioned Z3. If you ask questions about SMT implementation through large language models, you're going to get exactly the textbook explanations. But the large language model never tries them. It is repeating the learnings from people that wrote the papers. Right? It's exact that's exactly what you said. They're repeating. They're not they're not trying experimenting themselves. Yeah. The this part for sure is missing. But it feels in reach. I mean, I I I don't see an obstacle here. People will try that, right, to to improve the large language models in this direction. Right?

Competence without comprehension, and verified guardrails



Tim Scarfe

00:53:03

I mean, I'm so torn on it because competence without comprehension is not necessarily a bad thing because we're now writing software which is intelligent. So in the olden days we wrote software which does did exactly what it was supposed to, here's what you do. And now there's some flexibility. So we place some constraints in. And then within those constraints, there's some adaptation and we can control that as an engineering problem. But these constraints are really good. You know, like for example, there are certain things that we've just figured out and that's why they've survived in our cultural lexicon, certain algorithms, certain types of thinking. And frankly, it would be a waste of time to make the AI rediscover those things from scratch.

Leonardo de Moura

00:53:44

So Yes. Yeah. Yes. Learning by doing it gives you a new perspective. Right? Making AI can get new perspectives by learning by doing itself. Right? It's not relying on what's what's widgets, but learning by doing itself, maybe updating their own their weights.

Tim Scarfe

00:54:06

That that's conceivable. Yes. And and maybe it's just a matter of computation.

Leonardo de Moura

00:54:11

Yes.

Tim Scarfe

00:54:11

And maybe if we did have a society of agents that were actually doing weight adaptation.

Leonardo de Moura

00:54:16

But

Tim Scarfe

00:54:17

it would be quite difficult and possibly dangerous in some ways because the good thing about having big fat models that are trained every 6 months and you know on a platform is can put safety guardrails on there and at least there's only one model to check. Mean just imagine this is something that interests me as well is the extent to which the platform owners should be responsible versus individual engineers responsibility, right? Because if I make a sophisticated agentic system, it's already got to the point now that I don't read, I don't think anyone should be deploying Fable-class models in production. You know so maybe during the development and the R and D phase you need more intelligence and then the engineers place constraints in and then it should get to the point where it's halfway between a normal software application and you know so then maybe we we would have a lower intelligence model in production. But how much responsibility do you think the engineers have for kind of degenerate behavior?

Leonardo de Moura

00:55:17

Well, the main concern is you are talking about a scenario where the application has AI embedded insights. Right? If you have this scenario, I think it's super important to have the guardrails. Right? To have and even better if your guardrails are verified using a system like Lean. You're proving properties about the guardrails. Some people are talking about having verified sandboxes in the future. Yeah. I would love to see that. In my mind, I always use AI to develop software that does not contain AI. Lean does not contain AI. Right? This is a much safer scenario. Right? And many people are using like that. They build applications, but the applications themselves do not have AI embedded in them. It is a much safer scenario. Right? So you are fighting against bugs in the application, not putting guardrails around this AI.

Is the human still the author? ●

Tim Scarfe

00:56:22

If I get, you know, AI to generate some Lean and then Lean proves it, is the human still the author? And will humans play a meaningful part of this in 5 years time? Really hard to tell. Right? I mean, I couldn't predict. 1 year ago, I could not have predicted what happened so far. Right? I mean, I cannot even imagine in 5 years. It feels like it's accelerating. It's moving way faster than we expected. Sometimes people ask me, do you fear being becoming obsolete? I would feel like we leave.

Leonardo de Moura

00:57:00

I mean, developing software, it would be feel like mission accomplishes. I don't have to do it myself anymore. Maybe I can spend my time only with implementing softwares that do not have to work. I would not be the end of the world. I hope one thing I want to become I I think we are always going to be in the loop because we have to keep always the specification saying what the software should do will be always our responsibility. We're the ones that are benefiting from the AI. We're the ones that's saying, this is what we want. We will be there somewhere even if it's just at the specification level, stating what we want. Of course, people can create scenarios where the AI is super intelligent. We have robots, and we talk to the robots what we want, and they they go in natural language and they are obedient. They do They do exactly what we asked. But we're still the the ones telling them what to do. And so I I I feel like we are always going to be some role for for us in the future, But I I can see the exciting parts. I mean, how many as a developer most developers I know have many ideas, but they only implement a small subset of them because time is finite. I mean, they they don't have time to implement all their ideas. And and they see many of my colleagues, they they see the ones that have the most the ones that are most probable to work. It's like you're you're playing a game saying the efforts versus the chance is going to work. And many of the ideas you discard, you see, this very low probability is going to work. I mean but with AI, we can try more ideas. Right? I give the example of Joachim. I mean, he's not vibe coding in the sense people do, but for trying ideas, he he sends the AI different missions and sees the ones that are worthwhile his time to implement them. This is already a big win. Right? I mean

Tim Scarfe

00:59:16

Yeah. It does make me think about because I guess it's so difficult to see the future because we're just starting to find out what the what the meaningful boundary is. But I think that humans do have this kind of deeper understanding. And that means that we have better instincts for what are interesting problems. And we're also better at this distillation. So I mean, Mathlib, example, if it becomes really cheap to produce theorems in the future, maybe the human role would be more about kind of distillation and structure.

Leonardo de Moura

00:59:43

Yeah. Yeah. I I can see Mathlib in the near future, people will use the AI to maintain the proofs, to construct new proofs, proofs that are boring, that nobody cares. But the key ones, they will still write by hands because they want them to be beautiful or have some specific structure that's good for teaching, for communicating ideas. Coming up with the the road map also is going to be a human activity. What is in Mathlib? What is not? How do we define the rings or and other mathematical structures? These are all going to be done by humans. There are lots of exploration happening. For example, one cool idea was and I've seen people find this idea. Let's say I start building the whole mathematical library. What's going to happen? Right? It can be garbage. But how do we control this process? Johan Commelin, he's the one of the main people behind this Formal Frontiers. The idea is to have one way to control the process is to have key theorems that are super simple to state but needs a lot of math to be proven. Because if the AI managed to to to construct a proof of this key theorem, it's doing something worthwhile there. One example is Fermat's Last Theorem. I mean, you you can write if the natural you only need the natural numbers to state the theorem. It's very simple, but you need a lot of math to prove it. You can say, like, this is a milestone for the AI. You can build your crazy library, but you have to get there. I mean, you have to show me you got there, and you can have many of these milestones that are easy to state, but really hard to prove. And if the AI managed to build a library that allows you to prove that maybe it's worth our time looking what is there. Right?³⁰

³⁰Fermat's Last Theorem formalisation — other Leo proposes Fermat's Last Theorem as an example of a simple-to-state milestone for AI-built formal libraries.

AlphaProof, LLMs and why certificates still matter •

Tim Scarfe

01:01:44

Yeah. One of one of the age-old debates in AI, I guess, is kind of pure connectionism versus neurosymbolic. And I think now we are in the neurosymbolic age and these agentic harnesses they are incredible. Think they're the reason why AI works so well. There are still folks who think that connectionism only is really good. But just to put a bit of flavor on this. So we've long had the intuition that adding structure and constraints to neural networks makes them better and this was the intuition that Google had in 2024 when they won silver at the IMO using AlphaProof which auto-formalized Lean code and generated these Lean proofs. But there's been a bit of an interesting development which is that, last year they won gold and they were just using a pure, I think an agentic LLM or just a pure, can't actually remember, but this is interesting, right? So these models, they're just neural networks. They don't have any hard constraints. And an example of this is when they play chess, apparently they're getting better. So they generate valid moves about 91% of the time, which frankly, given it's just a kind of, spaghetti neural network is very impressive that it's generating these valid moves so far. But how do you see this going forward? So do you think that folks like Google will move back to Lean in some way? Or do you think the direction of travel is that as we scale these things up that they're kind of internalizing some kind of these constraints?³¹

Leonardo de Moura

01:03:11

Okay.

Tim Scarfe

01:03:11

I think

Leonardo de Moura

01:03:12

there are 2 ways to to to answer your question. One is whether techniques like AlphaProof, it follows the Monte Carlo tree search. It's like a hybrid technique. You can maybe even say it's a truly neurosymbolic technique for proving whether this pays off or not. Right? And the other one is that, do we need formal verification at all? I think the answer to the second question is yes. I mean, even if we have a model that generates right answers 99.99999% of the time, we use we still want certificate. Just to give an example of how proofs can get massive, the this

³¹AlphaProof — paper Tim cites AlphaProof, which auto-formalised Lean code for its IMO silver result, in the neurosymbolic versus pure-LLM debate.

unit-distance conjecture that OpenAI proved false. Boris Alexeev submits a proof to is 1,200,000 lines of Lean, the formal proof. Nobody wants to check the proof line by line. Right? It's great to have a certificate. I think having certificates that are machine-checkable, this is excellent. Having moves that's if you are playing with the AlphaZero architecture, the moves or a a large language model is saying, boom, boom, boom. These are the moves to do it. You still have the moves. Right? It's just how you produce the moves differently. I think having the moves in the game, viewing math as a game, this is value valuable. It will stay. Right? Because the AI will synthesize a formal proof that's a sequence of moves independently of the fact if it's a LLM agent agent control controlled by LLM or is a Monte Carlo tree search combined with some kind of neural network. The question why it pays off for not is still open. I mean, I everybody's impressed. I mean, the community with GPT 5.6 and and Claude Fable, they, I think, outperform all these systems that are using Monte Carlo tree search. But then we have to ask the question, the costs. Right? Which one is cheaper? I was talking with some colleagues that are doing these experiments during the conference this week, and they told me it's not clear because each move of these systems using the hybrid approach of Monte Carlo tree search with the moves are much cheap, but they're making way more moves because the moves are dumber. The LLM is more expensive, but it gets there faster. Right? It's so it's too debatable. I can imagine in the future that for some domains, this hybrid approach works and the LM works. There are also scenarios where it goes we improve so much the hardware that's is so cheap that nobody bothers with hybrid approaches anymore. Everybody's using just large language models or something like a large language model. This part is still huge debates whether it pays off or not.³²

What's next for Lean, and its legacy •

Tim Scarfe

01:06:38

What what is what is the future of Lean? So I guess the the first part of the question is you've distilled Lean 4 so much. And do you kind of, I mean, what do you imagine the next version would look like? And also what do you imagine Lean's legacy will be after you're gone? Because you've built something now, which is, it's a phenomenon. It's gonna be around long after after you're gone. And and in sense, it it'll just continue to to to evolve. How do you see that process on further? Yeah.

³²AlphaZero — paper Leo compares proof search to game moves in an AlphaZero-style architecture versus an LLM agent.

Leonardo de Moura

01:07:09

Well, there is still lots of work to be done. Right? I mean, one thing that we want to improve a lot, Lean is a programming language. Right? We want Lean to generate code that's competitive with Rust. Right? Especially for array-manipulating programs. There will be a lot of investments there. A lot of investments in Lean as a platform for software verification. One thing is to verify some library or the other things to verify full systems. Right? I mean, that's much bigger. We want to invest there, make sure this Lean scales. We want to make sure that we have been keeping up with Mathlib growth, but Mathlib, the growth is by humans. Just to give you perspective, Mathlib from 2025, and now Mathlib grew 44%, but the compilation time reduced by 10%. This way, we are making Lean faster and faster. But with AI now, writing math, we have to make we have to step up. I mean, AI the library can grow way faster than we can keep adding improvements. We will need AI to help us to improve the performance and keep up with the size of the library. So scalability will be a big issue. How do we support a library that's 100,000,000 lines of code? This is an open question. The scalability, Lean as a programming language, as a software verification platform and then improving trust. We want to have a verified compiler. The code you generate from Lean. People don't understand that the trusted code base for Lean, if you only want to prove theorems, it's much smaller. You only care about the kernel. If you only care about proving theorems, but if you care about suppose that you wrote a program in Lean, you prove a property about it and you generate executable. Today, the trusted code base includes the compiler. That's much bigger than the kernel. And we want to prove this compiler. We want to remove in the future, we want to have a compiler that has been verified and you do not have to trust us that this compiler was implemented correctly. Not because we are malicious, but because we make mistakes, I think, like everybody. Reducing the trust code base will be an activity, right, from now on. We want to keep reducing. AIs will keep getting better and better at finding vulnerabilities, exploits. We want to keep improving that, making sure that Lean is a platform, I mean, for developing software and math about legacy. I I can even if I die today, Lean will continue. I mean, the Lean FRO is amazing. I think the the folks there can take care of everything. Even yeah. They don't need me. I'm always trying to help, but the project would continue even if I die.

Tim Scarfe

01:10:07

Mean And and back in 2013, I mean, even in your wildest dreams, did you imagine that it would be anything like it is today?

Leonardo de Moura

01:10:15

No. No. No. Even in my wildest dreams, I never imagined that Terence Tao would be using people like Terence Tao would be using Lean. Right? And giving talks about that. Never imagined that. I thought felt like we are building a platform for verified safety-critical systems, and they would be the proofs would be done by hands. Right? I mean, it's it's fun. Some people claim I'm lucky. Maybe I am moving. Because making Lean interactive was great for AI. Z3 is a black box. The AI has very cannot influence, cannot guide Z3, but it can guide Lean. Right? I was really lucky. So many amazing people like Jeremy, Mitch and Mario. I mean, I found him in an obscure conference. Sebastian Ullrich, Joachim Breitner, and so many Kim Morrison, so so many. All these folks at the FRO. I mean, yeah. All the colleagues. I mean, yeah, it's it's incredible. Kevin showing Kevin Buzzard showing up. Mean, Kevin Hartnett taking the interest, writing articles.³³

Tim Scarfe

01:11:36

Yeah. It was yeah. Maybe I'm the luckiest person in the world. And I watched a Computerphile video yesterday, there was a wonderful chap. He was using a tactic for tautology. Was it tautology detection reduction? But anyway, point is that it was quite intimidating. So, like there's a lot of tutorial videos and people are going through a million miles an hour and they're doing this and they're doing this and what would you say to people that the initial shock is quite intimidating using Lean?

How to start learning Lean •

Tim Scarfe

01:11:36

Yeah. It was yeah. Maybe I'm the luckiest person in the world. And I watched a Computerphile video yesterday, there was a wonderful chap. He was using a tactic for tautology. Was it tautology detection reduction? But anyway, point is that it was quite intimidating. So, like there's a lot of tutorial videos and people are going through a million miles an hour and they're doing this and they're doing this and what would you say to people that the initial shock is quite intimidating using Lean?³⁴

³³Terence Tao — person Leo says he never imagined mathematicians such as Terence Tao would use Lean.

³⁴Computerphile — organization Tim mentions a Computerphile video on Lean that he found intimidating for beginners.

Leonardo de Moura

01:12:12

Well, for software developers, I would recommend start using Lean as a programming language. Just write codes. Ignore the proofs. Don't try proofs at the beginning. Start writing codes and playing if it's executing. Right? Then when you're comfortable with the Lean syntax, you go to the next step. You start trying to prove simple properties about your codes. Another thing that I tell everybody, have your AI agents around. They know lots and lot. They know everything about Lean. They know Lean better than I do. I finally doubt that. No. They can explain. I mean, they can explain extensions that they do not develop. Lean is extensible. They know about all extensions, and they read the documentation. They will read the documentation. Very few users read the documentation. The AI reads it and can explain to you. You can tell the AI your backgrounds. For example, software developer is vague. If you have, for example, a software developer that has been using Haskell or Lisp, it's going to be much easier to learn Lean because they are all functional programming languages. But suppose you can tell the AI, I'm a C sharp developer or Java developer. The AI now will calibrate the explanations for customize to your backgrounds, and they're really effective at teaching you Lean step by step. You you if you open a video that I can see that's very overwhelming with many tactics that's how like moves in the game in the game of proving theorems, You can ask the AI, explain to me. This is the background I have. Tell me what I should know before I even try to understand this example. It's really good at that.

Tim Scarfe

01:14:10

Leo, it's been a pleasure and an honor having you on the show. Yes. Thank you so much for joining us.

Leonardo de Moura

01:14:14

Thank you.

References

[1] **Lean**

<https://lean-lang.org/>

tool Leo introduces himself as the creator of Lean, the system at the centre of the interview.

[2] **Lean FRO**

<https://lean-lang.org/fro/>

organization Leo introduces himself as chief architect and co-founder of the Lean FRO, the nonprofit behind Lean.

[3] **Mathlib**

<https://github.com/leanprover-community/mathlib4>

tool Tim contrasts Lean's tightly controlled core with the large community library built on top of it.

[4] **Ilya Sergey**

<https://ilyasergey.net/>

person Leo recalls a demo by Ilya Sergey and his student of a software-verification language built as an extension on top of Lean.

[5] **The Proof in the Code**

<https://us.macmillan.com/books/9780374620059/theproofinthecode/>

book Tim recalls a passage in Kevin Hartnett's book about Leo removing people from the Lean Slack.

[6] **Brandolini's law**

https://en.wikipedia.org/wiki/Brandolini%27s_law

other Tim names the principle that refuting nonsense takes far more effort than producing it, in the context of AI-generated ideas.

[7] **Joachim Breitner**

<https://www.joachim-breitner.de/>

person Leo credits Joachim Breitner with handling the recent serious bug and communicating about it to the community.

[8] **nanoda_lib**

https://github.com/ammkrn/nanoda_lib

tool An external Lean kernel written in Rust that Leo says was also accepted by the purported Collatz disproof.

[9] **nanoda_lib pull request 22**

https://github.com/ammkrn/nanoda_lib/pull/22

tool The nanoda_lib fix pull request ('more struct and inductive checks'), merged the day before the Lean kernel issue was filed; it fits the earlier nanoda fix Leo

describes.

[10] **CollatzLean**

<https://github.com/xrchz/CollatzLean/blob/a79357462a33d2a6babd4cf6c8d8bcd25425d65README.md>

tool Repository of the purported Collatz disproof that, in Leo's account, passed Lean's official kernel and nanoda.

[11] **Lean issue 14576**

<https://github.com/leanprover/lean4/issues/14576>

tool Upstream issue titled 'Kernel accepts wrong-structure projections, allowing an axiom-free proof of False', which corresponds to the official-kernel bug Leo describes.

[12] **Lean pull request 14577**

<https://github.com/leanprover/lean4/pull/14577>

tool Leo's fix for the official kernel ('missing check at kernel inductive declaration'), matching the fix he mentions.

[13] **Lean comparator**

<https://github.com/leanprover/comparator>

tool The tool Leo describes that exports proofs and checks them in a sandbox, guarding against metaprogramming tricks and kernel bugs.

[14] **Lean4Lean**

<https://github.com/digama0/lean4lean>

tool Mario Carneiro's kernel, which Leo says is being proved correct and would be more than just another independent kernel.

[15] **ARC-AGI-3**

<https://arcprize.org/arc-agi/3>

tool Tim uses the benchmark's semi-private and private datasets as an analogy for keeping a private kernel.

[16] **lean-zip**

<https://github.com/kim-em/lean-zip>

tool Kim Morrison's Lean zlib implementation, produced with a Claude agent, which Tim and Leo discuss as an AI-written program with proved properties.

[17] **CompCert**

<https://compcert.org/>

tool Leo cites CompCert as a famous verification project where people thought hard about the specification.

[18] **seL4**

<https://www.sel4.org/>

tool Leo cites seL4 alongside CompCert as a famous verification project with a

carefully considered specification.

[19] **Z3**

<https://github.com/Z3Prover/z3>

tool Tim recalls a claim that Z3 found many bugs in Windows 7 before release; Leo later calls Z3 a black box that AI cannot guide, unlike Lean.

[20] **Adam Chlipala**

<https://adam.chlipala.net/>

person Leo credits Adam Chlipala at MIT with the advice that Lean needed a real tactic framework.

[21] **The Lean 4 theorem prover and programming language**

https://doi.org/10.1007/978-3-030-79876-5_37

paper Paper describing Lean 4, the rewrite in which Lean is implemented in Lean itself, a story Leo tells here.

[22] **Veil**

<https://github.com/verse-lab/veil>

tool The other extension Leo names, a Lean framework for verifying transition systems such as distributed protocols.

[23] **Velvet**

<https://github.com/verse-lab/velvet>

tool One of the extensions Leo names as built on Lean 4's extensibility.

[24] **Kevin Buzzard**

<https://www.ma.imperial.ac.uk/~buzzard/>

person Tim recalls Kevin Buzzard's remark that the formalised share of mathematics will stay small as mathematics keeps growing.

[25] **Mathlib Initiative**

<https://mathlib-initiative.org/about/>

organization Leo describes the initiative's Formal Frontiers project, which aims to help humans and AI fill Mathlib's gaps.

[26] **A new result on unit distances**

<https://openai.com/index/model-disproves-discrete-geometry-conjecture/>

other Tim points to a proof from a new GPT model disproving a unit-distance conjecture, and Leo later notes its 1.2-million-line Lean formalisation.

[27] **Fermat's Last Theorem formalisation**

<https://imperialcollegelondon.github.io/FLT/>

other Leo proposes Fermat's Last Theorem as an example of a simple-to-state milestone for AI-built formal libraries.

[28] **AlphaProof**

<https://doi.org/10.1038/s41586-025-09833-y>

paper Tim cites AlphaProof, which auto-formalised Lean code for its IMO silver result, in the neurosymbolic versus pure-LLM debate.

[29] AlphaZero

<https://arxiv.org/abs/1712.01815>

paper Leo compares proof search to game moves in an AlphaZero-style architecture versus an LLM agent.

[30] Terence Tao

<https://terrytao.wordpress.com/>

person Leo says he never imagined mathematicians such as Terence Tao would use Lean.

[31] Computerphile

<https://www.youtube.com/@Computerphile>

organization Tim mentions a Computerphile video on Lean that he found intimidating for beginners.