Lean 4: Bridging Formal Mathematics and Software Verification

36th International Conference on Computer Aided Verification

July 25th 2024

Leo de Moura

Senior Principal Applied Scientist – AWS

Chief Architect – Lean FRO

What is LEAN ?

An efficient pure functional **programming language** and a **proof assistant.**

Lean is **implemented in Lean**, and is very **extensible.**

Main applications:

- Formal Mathematics
- Software/Hardware verification
- AI for Mathematics and code synthesis
- Education

Small trusted kernel, and many external type/proof checkers.

Soonho Kong and I started the project at Microsoft Research in 2013.

What is LEAN?

Lean is based on dependent type theory

```
def f (n m : Nat) (xs : BitVec n) (ys : BitVec m) : BitVec (m+n+m) :=
  ys ++ xs ++ ys

structure FinNat (n : Nat) where
  val : Nat
  bounded : val < n

def a : FinNat 3 := {
  val := 0
  bounded := by decide
}
```

What is LEAN?

Lean is based on dependent type theory

```
def f (n m : Nat) (xs : BitVec n) (ys : BitVec m) : BitVec (m+n+m) :=
  ys ++ xs ++ ys

structure FinNat (n : Nat) where
  val : Nat
  bounded : val < n

def a : FinNat 3 := {
  val := 4
  bounded := by decide
}
```

```
tactic 'decide' proved that the proposition
  4 < 3
is false Lean 4
```

LEAN is and IDE for formal methods

Lean is a development environment for formal methods.

Proofs and definitions are machine checkable.

The math community using Lean is growing rapidly. They love the system.

Lean is used in several software verification projects at AWS since 2023.

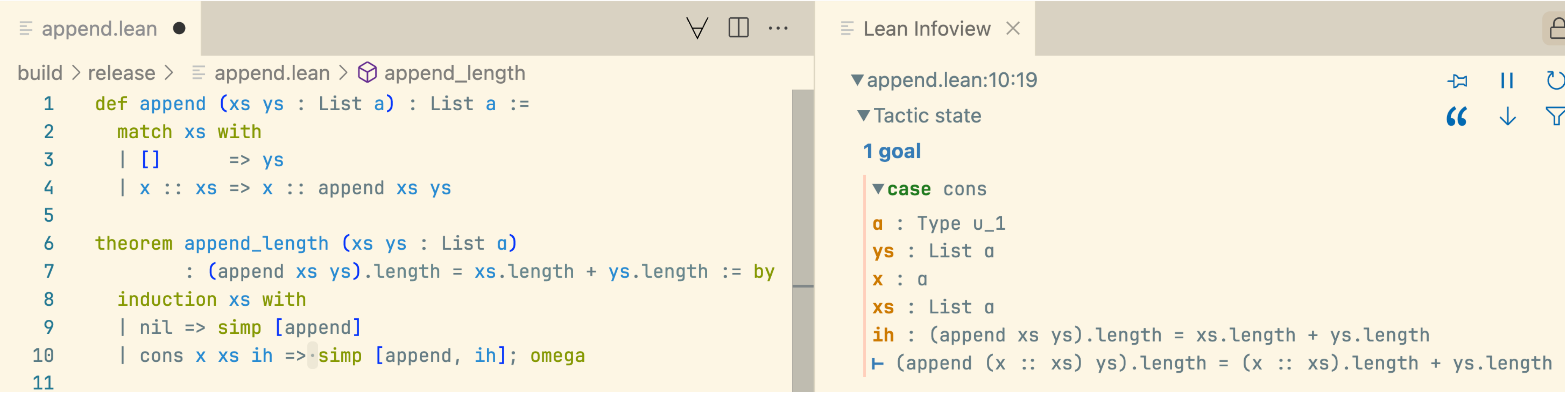
```lean
theorem euclid_exists_infinite_primes (n : ℕ) : ∃ p, n ≤ p ∧ Prime p :=
  let p := minFac (factorial n + 1)
  have f1 : (factorial n + 1) ≠ 1 :=
    ne_of_gt $ succ_lt_succ' $ factorial_pos _
  have pp : Prime p :=
    min_fac_prime f1
  have np : n ≤ p := le_of_not_ge fun h =>
    have h₁ : p ∣ factorial n := dvd_factorial (min_fac_pos _) h
    have h₂ : p ∣ 1 := (Nat.dvd_add_iff_right h₁).2 (min_fac_dvd _)
    pp.not_dvd_one h₂
  Exists.intro p
```

LEAN and formal mathematics

Mathlib > RingTheory > Finiteness.lean

```lean
/-- **Nakayama's Lemma**. Atiyah-Macdonald 2.5, Eisenbud 4.7, Matsumura 2.2,
[Stacks 00DV](https://stacks.math.columbia.edu/tag/00DV) -/
theorem exists_sub_one_mem_and_smul_eq_zero_of_fg_of_le_smul {R : Type _} [CommRing R] {M : Type _}
    [AddCommGroup M] [Module R M] (I : Ideal R) (N : Submodule R M) (hn : N.FG) (hin : N ≤ I • N) :
    ∃ r : R, r - 1 ∈ I ∧ ∀ n ∈ N, r • n = (0 : M) := by
  rw [fg_def] at hn
  rcases hn with ⟨s, hfs, hs⟩
  have : ∃ r : R, r - 1 ∈ I ∧ N ≤ (I • span R s).comap (LinearMap.lsmul R M r) ∧ s ⊆ N := by
    refine' ⟨1, _, _, _⟩
    · rw [sub_self]
      exact I.zero_mem
    · rw [hs]
      intro n hn
      rw [mem_comap]
      change (1 : R) • n ∈ I • N
      rw [one_smul]
      exact hin hn
    · rw [← span_le, hs]
```

LEAN has a rich user-interface



Should we trust LEAN ?

Lean has a small trusted proof checker.

Do I need to trust the checker?

No, you can export your proof, and use external checkers. There are checkers implemented in Haskell, Scala, Rust, **Lean**, etc.

You can implement your own checker.

LEAN enables decentralized collaboration

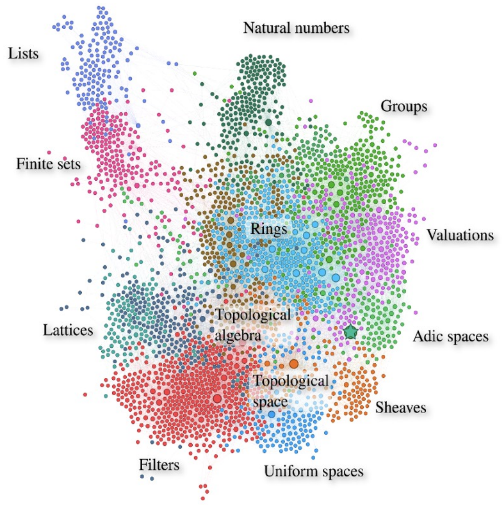
Meta-programming

Users extend Lean using Lean itself

Proof automation

Visualization tools

Custom notation



Formal Proofs

You don't need to trust me to use my proofs.

You don't need to trust my proof automation to use it.

Hack without fear.

LEAN enables decentralized collaboration

Meta-programming

Users extend Lean using Lean itself

Proof automation

Visualization tools

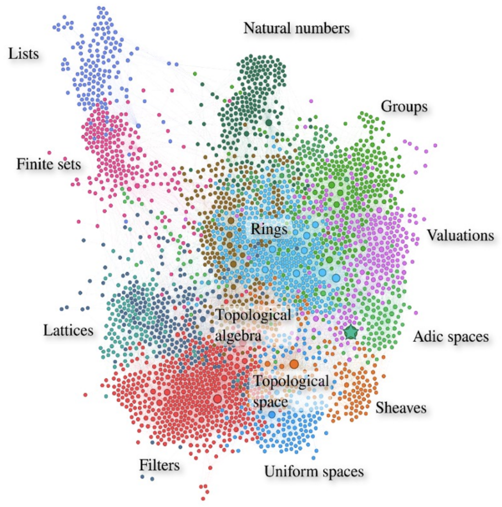
Custom notation

Formal Proofs

You don't need to trust me to use my proofs.

You don't need to trust my proof automation to use it.

Hack without fear.



Takeaway: formal proofs address the “Trust Bottleneck”

LEAN Community

starts Mathlib – 2017

Documentation   Mathlib.AlgebraicGeometry.PrimeSpectrum.Maximal   Search   Google site search

Main definitions

- `MaximalSpectrum R`: The maximal spectrum of a commutative ring `R`, i.e., the set of all maximal ideals of `R`.

Implementation notes

The Zariski topology on the maximal spectrum is defined as the subspace topology induced by the natural inclusion into the prime spectrum to avoid API duplication for zero loci.

```
theorem MaximalSpectrum.toPrimeSpectrum_range
    {R : Type u} [CommRing R] :
  Set.range MaximalSpectrum.toPrimeSpectrum =
   {x : PrimeSpectrum R | IsClosed {x}}
```

source

```
instance MaximalSpectrum.zariskiTopology
    {R : Type u} [CommRing R] :
  TopologicalSpace (MaximalSpectrum R)
```

source

The Zariski topology on the maximal spectrum of a commutative ring is defined as the subspace topology induced by the natural inclusion into the prime spectrum.

▶ Equations

return to top

source

▶ Imports
▶ Imported by

MaximalSpectrum.toPrimeSpectrum_range
MaximalSpectrum.zariskiTopology
MaximalSpectrum.instT1Space
MaximalSpectrum.toPrimeSpectrum_continuous

The Lean Mathematical Library

The mathlib Community*

Abstract

This paper describes mathlib, a community-driven effort to build a unified library of mathematics formalized in the Lean proof assistant. Among proof assistant libraries, it is distinguished by its dependently typed foundations, focus on classical mathematics, extensive hierarchy of structures, use of large- and small-scale automation, and distributed organization. We explain the architecture and design decisions of the library and the social organization that has led to its development.

Lean 4 development starts in 2018

Sebastian Ullrich and I start Lean 4 in 2018

Main goal: make the system much more extensible and address many limitations

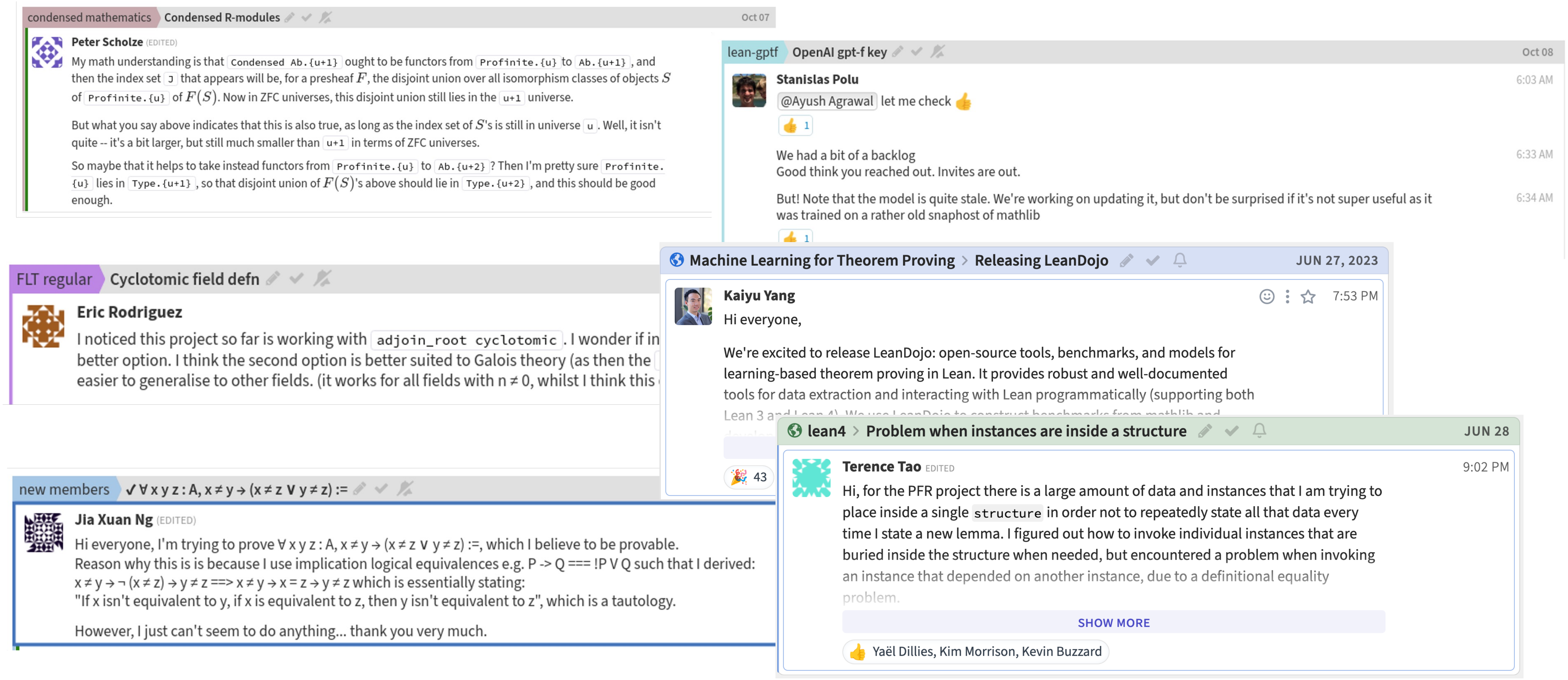
Lean is now a general-purpose and efficient programming language

Implemented in 120+ kLoC of Lean!

Opened up parser and elaborator for complex notations, embedded languages, …

It is not backward compatible with Lean 3

The Lean Zulip Channel – https://leanprover.zulipchat.com



condensed mathematics > Condensed R-modules — Oct 07

Peter Scholze (EDITED)

My math understanding is that `Condensed Ab.{u+1}` ought to be functors from `Profinite.{u}` to `Ab.{u+1}`, and then the index set `J` that appears will be, for a presheaf F, the disjoint union over all isomorphism classes of objects S of `Profinite.{u}` of $F(S)$. Now in ZFC universes, this disjoint union still lies in the `u+1` universe.

But what you say above indicates that this is also true, as long as the index set of S's is still in universe `u`. Well, it isn't quite -- it's a bit larger, but still much smaller than `u+1` in terms of ZFC universes.

So maybe that it helps to take instead functors from `Profinite.{u}` to `Ab.{u+2}`? Then I'm pretty sure `Profinite.{u}` lies in `Type.{u+1}`, so that disjoint union of $F(S)$'s above should lie in `Type.{u+2}`, and this should be good enough.

lean-gptf > OpenAI gpt-f key — Oct 08

Stanislas Polu — 6:03 AM

@Ayush Agrawal let me check 👍

👍 1

We had a bit of a backlog — 6:33 AM
Good think you reached out. Invites are out.

But! Note that the model is quite stale. We're working on updating it, but don't be surprised if it's not super useful as it was trained on a rather old snaphost of mathlib — 6:34 AM

👍 1

FLT regular > Cyclotomic field defn

Eric Rodriguez

I noticed this project so far is working with `adjoin_root cyclotomic`. I wonder if in
better option. I think the second option is better suited to Galois theory (as then the
easier to generalise to other fields. (it works for all fields with n ≠ 0, whilst I think this

Machine Learning for Theorem Proving > Releasing LeanDojo — JUN 27, 2023

Kaiyu Yang — 7:53 PM

Hi everyone,

We're excited to release LeanDojo: open-source tools, benchmarks, and models for learning-based theorem proving in Lean. It provides robust and well-documented tools for data extraction and interacting with Lean programmatically (supporting both

🎉 43

new members > ✓ ∀ x y z : A, x ≠ y → (x ≠ z ∨ y ≠ z) :=

Jia Xuan Ng (EDITED)

Hi everyone, I'm trying to prove ∀ x y z : A, x ≠ y → (x ≠ z ∨ y ≠ z) :=, which I believe to be provable.
Reason why this is is because I use implication logical equivalences e.g. P -> Q === !P V Q such that I derived:
x ≠ y → ¬ (x ≠ z) → y ≠ z ==> x ≠ y → x = z → y ≠ z which is essentially stating:
"If x isn't equivalent to y, if x is equivalent to z, then y isn't equivalent to z", which is a tautology.

However, I just can't seem to do anything... thank you very much.

lean4 > Problem when instances are inside a structure — JUN 28

Terence Tao EDITED — 9:02 PM

Hi, for the PFR project there is a large amount of data and instances that I am trying to place inside a single `structure` in order not to repeatedly state all that data every time I state a new lemma. I figured out how to invoke individual instances that are buried inside the structure when needed, but encountered a problem when invoking an instance that depended on another instance, due to a definitional equality problem.

SHOW MORE

👍 Yaël Dillies, Kim Morrison, Kevin Buzzard

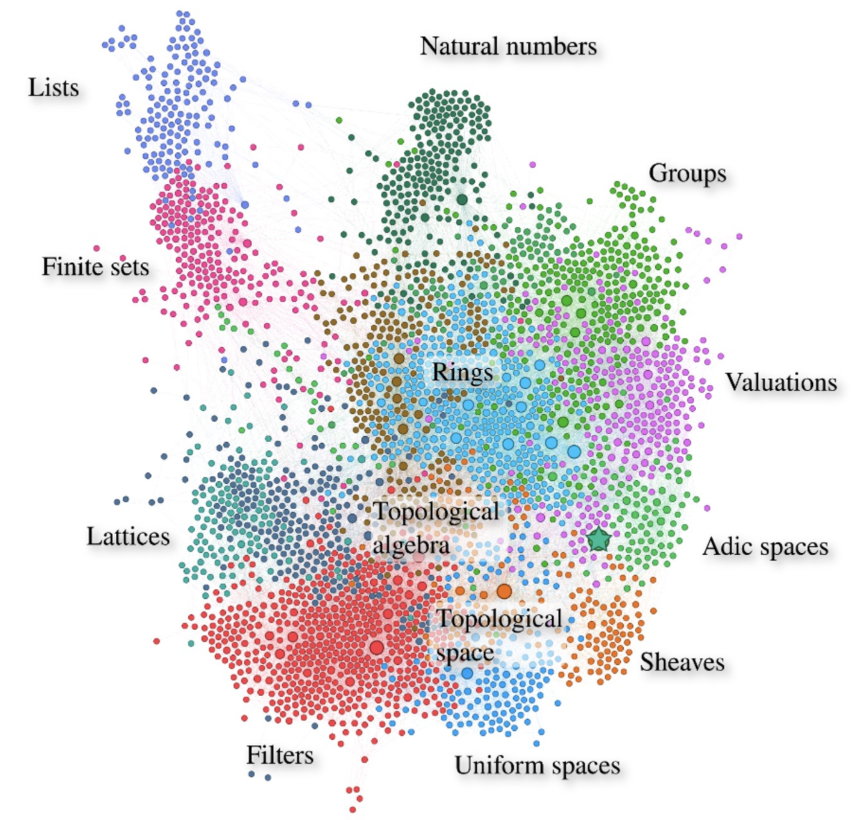
Lean perfectoid spaces – 2019

by Kevin Buzzard, Johan Commelin, and Patrick Massot

What is it about?

We explained Peter Scholze's definition of perfectoid spaces to computers, using the Lean theorem prover, mainly developed at Microsoft Research by Leonardo de Moura. Building on earlier work by many people, starting from first principles, we arrived at

```
-- We fix a prime number p
parameter (p : primes)

/-- A perfectoid ring is a Huber ring that is complete, uniform,
that has a pseudo-uniformizer whose p-th power divides p in the power bounded subring,
and such that Frobenius is a surjection on the reduction modulo p.-/
structure perfectoid_ring (R : Type) [Huber_ring R] extends Tate_ring R : Prop :=
(complete  : is_complete_hausdorff R)
(uniform   : is_uniform R)
(ramified  : ∃ ϖ : pseudo_uniformizer R, ϖ^p ∣ p in R°)
(Frobenius : surjective (Frob R°/p))

/-
CLVRS ("complete locally valued ringed space") is a category
whose objects are topological spaces with a sheaf of complete topological rings
and an equivalence class of valuation on each stalk, whose support is the unique
maximal ideal of the stalk; in Wedhorn's notes this category is called 𝒱.
A perfectoid space is an object of CLVRS which is locally isomorphic to Spa(A) with
A a perfectoid ring. Note however that CLVRS is a full subcategory of the category
`PreValuedRingedSpace` of topological spaces equipped with a presheaf of topological
rings and a valuation on each stalk, so the isomorphism can be checked in
PreValuedRingedSpace instead, which is what we do.
-/

/-- Condition for an object of CLVRS to be perfectoid: every point should have an open
neighbourhood isomorphic to Spa(A) for some perfectoid ring A.-/
def is_perfectoid (X : CLVRS) : Prop :=
∀ x : X, ∃ (U : opens X) (A : Huber_pair) [perfectoid_ring A],
  (x ∈ U) ∧ (Spa A ≅ U)

/-- The category of perfectoid spaces.-/
def PerfectoidSpace := {X : CLVRS // is_perfectoid X}

end
```



mathoverflow

Home
Questions
Tags

What are "perfectoid spaces"?

Asked 9 years, 5 months ago Active 1 year, 5 months ago Viewed 49k times

67

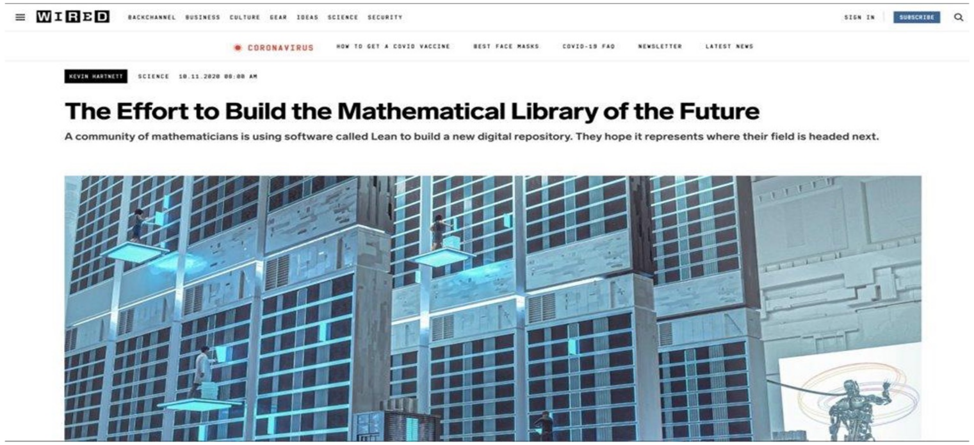
Here is a completely different kind of answer to this question.

A *perfectoid space* is a term of type `PerfectoidSpace` in the Lean theorem prover.

Here's a quote from the source code:

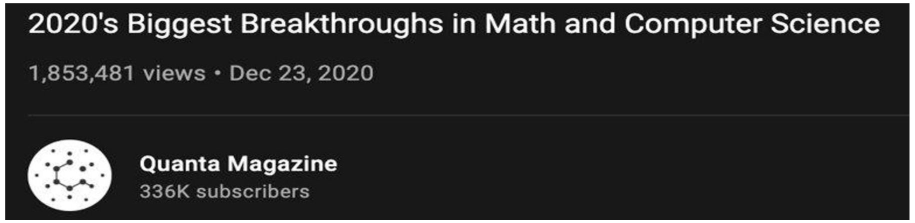
```
structure perfectoid_ring (R : Type) [Huber_ring R] extends Tate_ring R : Prop :=
(complete  : is_complete_hausdorff R)
```

The Lean Mathematical Library goes viral – 2020



The Effort to Build the Mathematical Library of the Future

A community of mathematicians is using software called Lean to build a new digital repository. They hope it represents where their field is headed next.



“You can do 14 hours a day in it and not get tired and feel kind of high the whole day,” Livingston said. “You’re constantly getting positive reinforcement.”

“It will be so cool that it’s worth a big-time investment now,” Macbeth said. “I’m investing time now so that somebody in the future can have that amazing experience.”

Liquid Tensor Experiment

Nov 2020: Peter Scholze posits formalization challenge

> "I spent much of 2019 obsessed with the proof of this theorem, almost getting crazy over it. In the end, we were able to get an argument pinned down on paper, but I think nobody else has dared to look at the details of this, and so I still have some small lingering doubts."

Liquid Tensor Experiment

Nov 2020: Peter Scholze posits formalization challenge

May 2021: Johan Commelin announces completed Lean formalization of crucial intermediary lemma, with only minor corrections

> “[T]his was precisely the kind of oversight I was worried about when I asked for the formal verification. [...] The proof walks a fine line, so if some argument needs constants that are quite a bit different from what I claimed, it might have collapsed.”

nature

Explore content ⌄ Journal information ⌄ Publish with us ⌄ Subscribe

nature > news > article

NEWS | 18 June 2021

Mathematicians welcome computer-assisted proof in ‘grand unification’ theory

Liquid Tensor Experiment

Nov 2020: Peter Scholze posits formalization challenge

May 2021: Johan Commelin announces completed Lean formalization of crucial intermediary lemma, with only minor corrections

July 2022: Completion of the full challenge in Lean

"**The Lean Proof Assistant was really that: an assistant in navigating through the thick jungle that this proof is.** Really, one key problem I had when I was trying to find this proof was that I was essentially unable to keep all the objects in my RAM, and I think the same problem occurs when trying to read the proof." *Peter Scholze*

Abstract Formalities

Johan Commelin's talk: http://www.fields.utoronto.ca/talks/Abstract-Formalities

Abstraction boundaries in Mathematics.

Formal mathematics as a tool for reducing the cognitive load.

Not just from raw proof complexity, but also

discrepancies between statements and proofs, side conditions, unstated assumptions, …

2. Formalization and abstraction boundaries

2.3. Specifications — managing refactors; unexpected gems

Experience from LTE:

1a Wrote down properties of Breen–Deligne resolutions

1b Discovered easier object with similar behaviour

2a Key statements written down without proofs
after stubbing out definitions (example: Ext)

2b Several definitions and lemmas were tweaked

2c After the dust settled, distribute work on the proofs

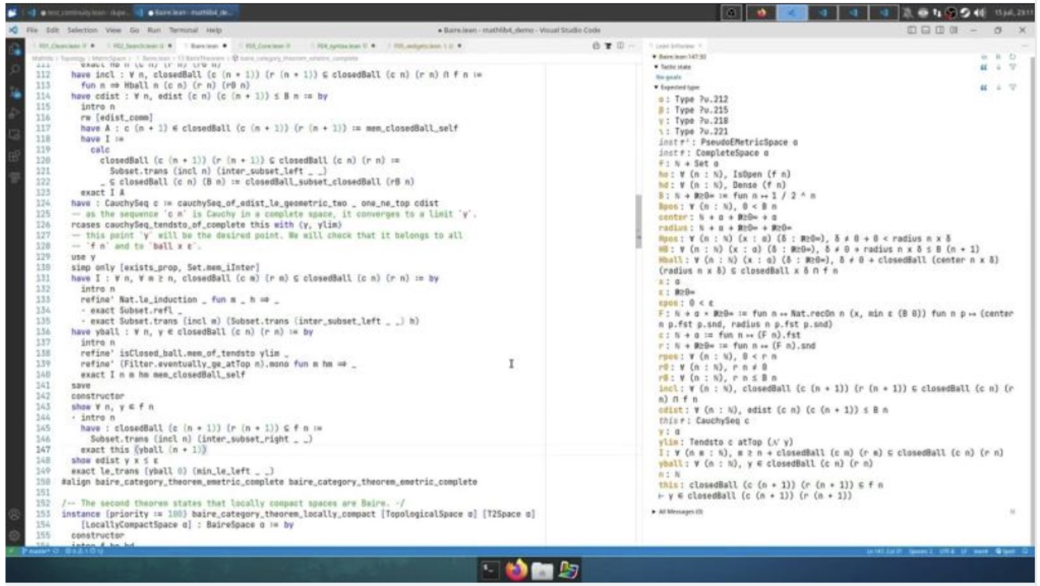
3 Sometimes large proofs or libraries
still had to be refactored (yes, it was painful)

Unexpected win: Johan’s team simplified the proof without fully understanding it.

Mathlib is ported to Lean 4 – 2023

Leonardo de Moura (He/Him) • You
Senior Principal Applied Scientist at AWS, and Chief Architect ...
1mo •

I am thrilled to announce that the Mathlib (https://lnkd.in/gx6eh4aG) port to Lean 4 has been successfully completed this weekend. It is truly remarkable that over 1 million lines of formal mathematics have been successfully migrated. Once again, the community has amazed me and surpassed all my expectations. This achievement also aligns with the 10th anniversary of my initial commit to Lean on July 15, 2013. Patrick Massot has graciously shared a delightful video commemorating this significant milestone, which can be viewed here: https://lnkd.in/gjVr72t8.



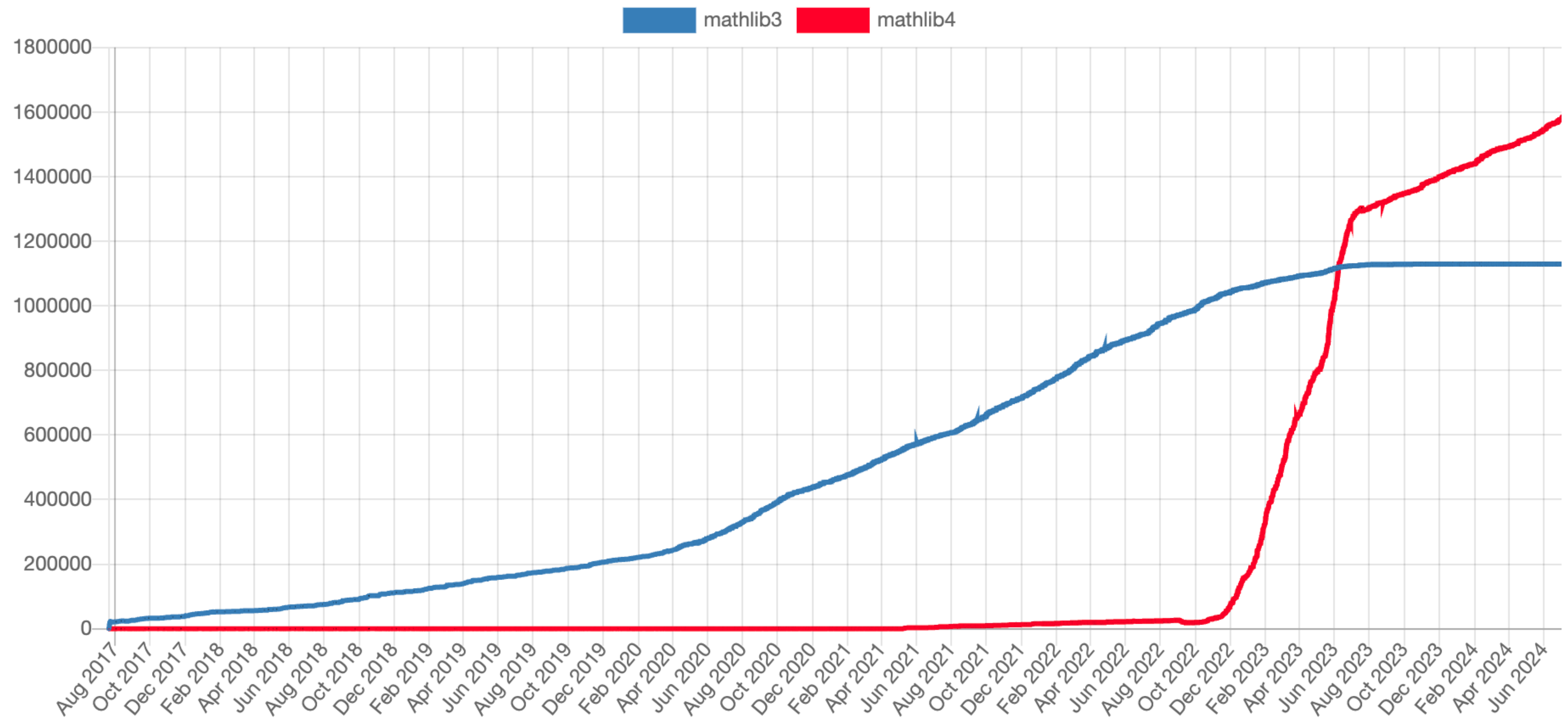
Lean 4 overview for Mathlib users - Patrick Massot
youtube.com

Takeaway: the power of the community

Mathlib statistics

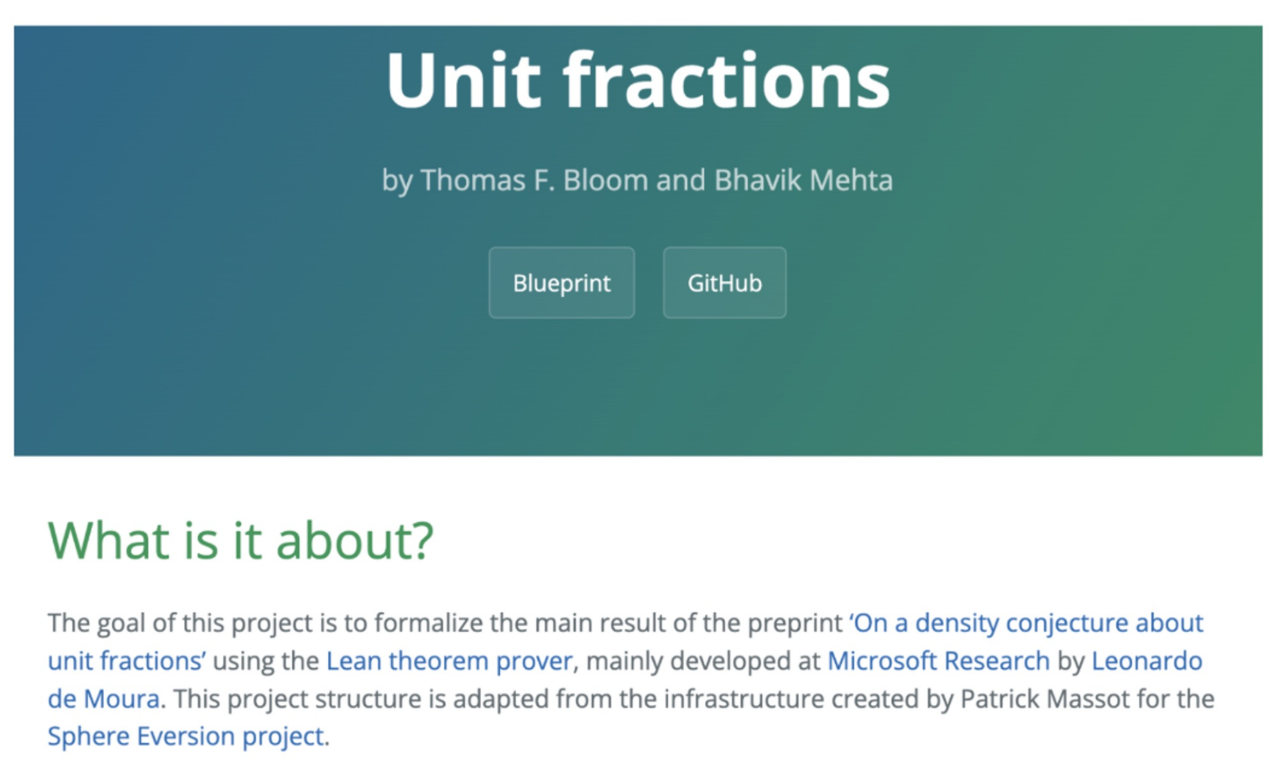
Counts

Definitions	Theorems	Contributors
81788	156453	310



LEAN is impacting how mathematics is done

Thomas' Bloom result: https://b-mehta.github.io/unit-fractions/



Unit fractions

by Thomas F. Bloom and Bhavik Mehta

Blueprint GitHub

What is it about?

The goal of this project is to formalize the main result of the preprint 'On a density conjecture about unit fractions' using the Lean theorem prover, mainly developed at Microsoft Research by Leonardo de Moura. This project structure is adapted from the infrastructure created by Patrick Massot for the Sphere Eversion project.

Timothy Gowers @wtgowers · Jun 13

Very excited that Thomas Bloom and Bhavik Mehta have done this. I think it's the first time that a serious contemporary result in "mainstream" mathematics doesn't have to be checked by a referee, because it has been checked formally. Maybe the sign of things to come ... 1/

Kevin Buzzard @XenaProject · Jun 12

Happy to report that Bloom went on to learn Lean this year and, together with Bhavik Mehta, has now formalised his proof in Lean b-mehta.github.io/unit-fractions/ (including formalising the Hardy-Littlewood circle method), finishing before he got a referee's report for the paper ;-)

Show this thread

2 26 140

LEAN is impacting how mathematics is done

Terence Tao
@tao@mathstodon.xyz

As a consequence of my #Lean4 formalization project I have found a small (but non-trivial) bug in my paper! While in the course of formalizing the arguments in page 6 of arxiv.org/pdf/2310.05328.pdf , I discovered that the expression $\frac{1}{2}\log\frac{n-1}{n-k-1}$ that appears in those arguments actually diverges in the case $n=3, k=2$! Fortunately this is an issue that is only present for small values of n, for which one can argue directly (with a worse constant), so I can fix the argument by changing some of the numerical constants on this page (the arguments here still work fine for $n \geq 8$, and the small n case can be handled by cruder methods).

Enclosed is the specific point where the formalization failed; Lean asked me to establish $0 < n-3$, but the hypothesis I had was only that $n > 2$, and so the "linarith" tactic could not obtain a contradiction from the negation of $0 < n-3$.

I'll add a footnote in the new version to the effect that the argument in the previous version of the paper was slightly incorrect, as was discovered after trying to formalize it in Lean.

```
n : ℕ
s : ℕ → ℝ
h1 : n > 2
h2 : attainable n s
h1' : 2 < ↑n
⊢ 0 < ↑n - 3
```

▾ Messages (1)

▾ prev_bound.lean:222:6

```
linarith failed to find a contradiction
▾ case h
n : ℕ
s : ℕ → ℝ
h1 : n > 2
h2 : attainable n s
h1' : 2 < ↑n
a✝ : 0 ≥ ↑n - 3
⊢ False
```

ALT

2023 was a great year for LEAN

The New York Times

A.I. and Chatbots › | Can A.I Be Fooled? | Testing a Tutorbot | Chatbot Prompts to Try | A.I.'s Literary Skills | What Are the Dangers of A.I.?

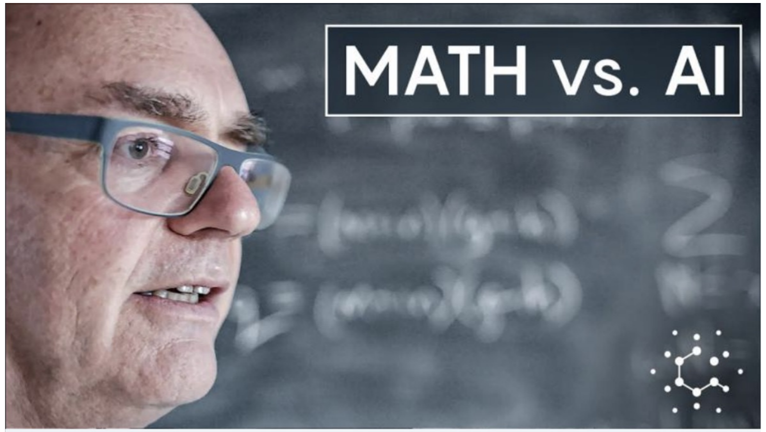
A.I. Is Coming for Mathematics, Too

For thousands of years, mathematicians have adapted to the latest advances in logic and reasoning. Are they ready for artificial intelligence?

Give this article

Terence Tao
@tao@mathstodon.xyz

Leo de Moura surveyed the features and use cases for Lean 4. I knew it primarily as a formal proof assistant, but it also allows for less intuitive applications, such as truly massive mathematical collaborations on which individual contributions do not need to be reviewed or trusted because they are all verified by Lean. Or to give a precise definition of an extremely complex mathematical object, such as a perfectoid space.



When Computers Write Proofs, What's the Point of Mathematicians?
youtube.com

2023 was a great year for LEAN

Daniel J. Bernstein
@djb@cr.yp.to

Formally verified theorems about decoding Goppa codes: cr.yp.to/2023/leangoppa-202307... This is using the Lean theorem prover; I'll try formalizing the same theorems in HOL Light for comparison. This is a step towards full verification of fast software for the McEliece cryptosystem.

Graydon Hoare
@graydon@types.pl

I fairly often find myself in conversations with people who wish Rust had more advanced types. And I always say it's pretty much at its cognitive-load and compatibility induced design limit, and if you want to go further you should try building a newer language. And many people find this answer disappointing because starting a language is a long hard task especially if it's to be a sophisticated one. And so people ask for a candidate project they might join and help instead. And my best suggestion for a while now has been Lean 4. I think it's really about the best thing going in terms of powerful research languages. Just a remarkable achievement on many many axes.

The Lean FRO – 2023

A non-profit organization dedicated to the development of Lean launched in July of 2023.

Missions:

- Address **scalability**, **usability**, and **proof automation** in Lean.
- Support formal mathematics.
- Achieve self-sustainability in 5 years.

Supported by Simons Foundation International, Alfred P. Sloan Foundation, and Richard Merkin

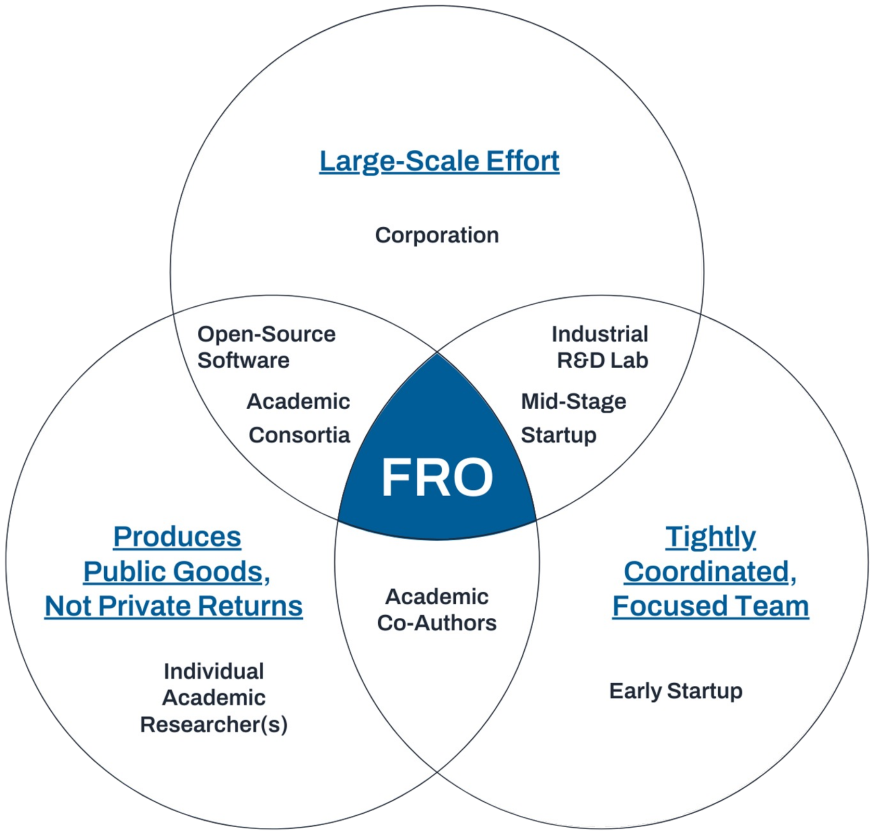
lean-fro.org

Focused Research Organization (FRO)

A new type of nonprofit startup for science developed by Convergent Research.

convergentresearch.org

CONVERGENT RESEARCH
A member of the Schmidt Futures Network



The Lean FRO

Team

Leo de Moura (AWS)
Chief Architect, Co-Founder

Sebastian Ullrich
Head of Engineering, Co-Founder

Corinna Calhoun
Chief Operating Officer

Henrik Böving
Research Software Engineer

Joachim Breitner
Principal Research Software Engineer

David Thrane Christiansen
Senior Research Software Engineer

Johan Commelin
Mathematical Research Engineer

Markus Himmel
Tech Lead

Marc Huisinga
Research Software Engineer

Mac Malone
Research Software Engineer

Kyle Miller
Research Software Engineer

Kim Morrison
Senior Research Software Engineer

Sofia Rodrigues
Research Software Engineer

Board of Directors

Adam Marblestone (Convergent Research)
Leo de Moura (AWS)
Jeremy Avigad (CMU)

Strategic Advisory Board

Simone Severini (AWS)
Alex Kontorovich (Rutgers University)
Lars Bergstrom (Google)
Leo de Moura (AWS)
Corinna Calhoun (Lean FRO)

2024 has been great so far

'A-Team' of Math Proves a Critical Link Between Addition and Sets

12

A team of four prominent mathematicians, including two Fields medalists, proved a conjecture described as a "holy grail of additive combinatorics." Within a month, a loose collaboration verified it with a computer-assisted proof.

INVITED ADDRESS

jmm

AMS Colloquium Lectures: Lecture I

Machine Assisted Proof

Terence Tao, *University of California, Los Angeles*

Introduced by Bryna Kra, *Northwestern University*

AMS AMERICAN MATHEMATICAL SOCIETY

DeepMind has formalized a theoretical result related to AI safety in Lean. Paper: https://lnkd.in/d6GVwFSU
Code: https://lnkd.in/d64ntAj5. ...see more

google-deepmind/
debate

Formalizing stochastic doubly-efficient debate

1 Contributor · 0 Issues · 76 Stars · 9 Forks

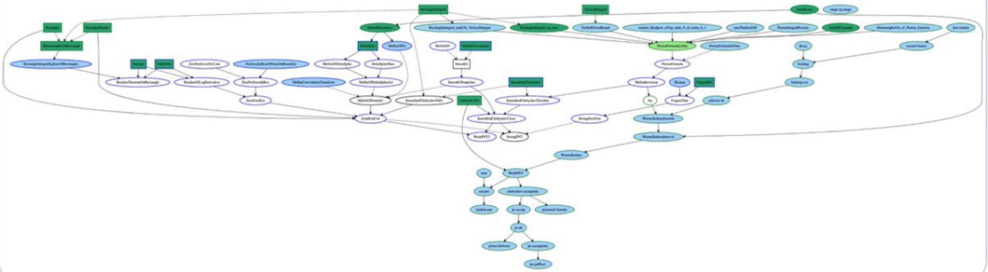
Alex Kontorovich @AlexKontorovich · Feb 1

Terry Tao and I are pleased to announce the "Prime Number Theorem and Beyond" project, which you can find here:

github.com/AlexKontorovic...

...

Show more



2024 has been great so far

SCI AM

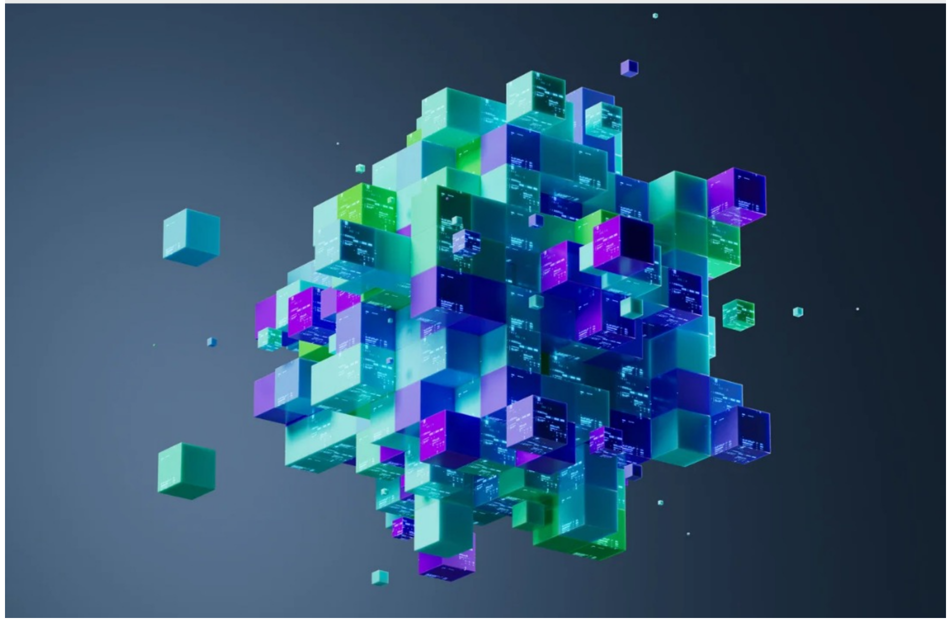
Sign Up for Our Daily Newsletter

JUNE 8, 2024 | 12 MIN READ

AI Will Become Mathematicians' 'Co-Pilot'

Fields Medalist Terence Tao explains how proof checkers and AI programs are dramatically changing mathematics

BY CHRISTOPH DRÖSSER



Anima Anandkumar in • 1st
Sr. Director of AI Research at NVIDIA and Bren Professor at Ca...
2mo •

Launching Lean Co-pilot for AI-human collaboration to write formal mathematical proofs that are 100% accurate. We use LLMs to suggest proof tactics in Lean and also allow humans to int ...see more

lean-dojo/
LeanCopilot

LLMs as Copilots for Theorem Proving in Lean

5	4	4	609	35
Contributors	Issues	Discussions	Stars	Forks

Nethermind
14,667 followers
1w •

+ Follow

Introducing Clear: the complete framework for interactive theorem proving in web3. A formal verification tool for Solidity smart contracts that expresses any on-paper verification into mechanized proofs.

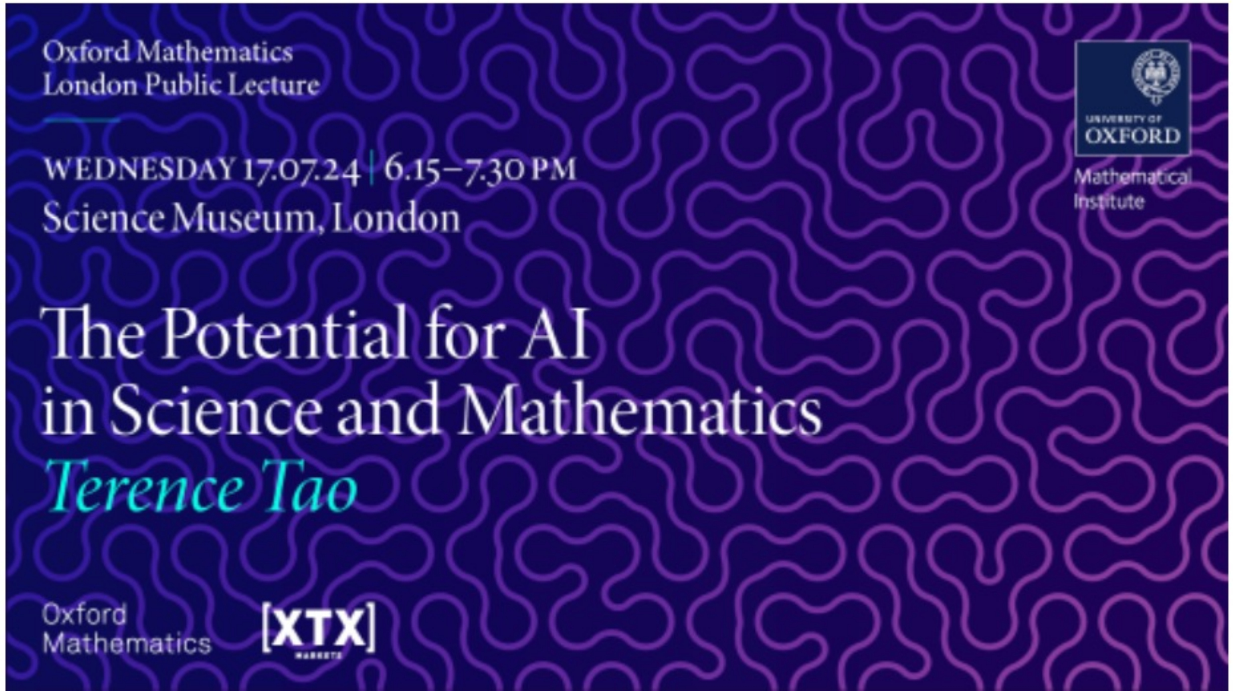
Read more in our blog post: https://lnkd.in/dZctgVWU

Clear under the hood

- Ensures stronger guarantees and greater expressivity (leveraging ITP & hard formal methods)
- Splits the task of verifying contracts into simpler sub-problems, enabling proof reuse and compositionality → achieving scalable verification not previously possible
- Allows extraction into Lean 4, using interactive theorem proving and Mathlib to verify complex smart contracts

To ensure the highest level of certainty in the correctness of our model, we're running our specification against EVM execution conformance tests.

#ethereum #solidity #formalverification #maths #blockchain #security #cryptosecurity #innovation #hacker #yul #evm



- **AI's unpredictability** is not problem for maths because we can use theorem provers to **verify AI-generated content**
- Lean and Mathlib are enabling experts from different backgrounds to **work collaboratively**.
- "*It's like solving a puzzle, like thinking at a different level […]* ***it makes you see the essence of why something really works*** *and gives you this extra level of internal security*" Terence Tao

Only The Beginning

Perfectoid Spaces, Buzzard, Commelin, and Massot 2019

Sphere Eversion, Massot, Nash, and van Doorn, 2020-2022

Liquid Tensor Experiment (LTE), Commelin et al., 2021-2022

Fermat’s Last Theorem for regular primes, Brasca et al., 2021-2023

Unit Fractions, Bloom and Mehta, 2022

Consistency of Quine's New Foundations, Wilshaw and Dillies, 2022-2024

Polynomial Freiman-Ruzsa Conjecture (PFR), Tao and Dillies, 2023

Prime Number Theorem And Beyond, Kontorovich and Tao, 2024-ongoing

Carleson Project, van Doorn, 2024-ongoing

Fermat’s Last Theorem (FLT), Buzzard, 2024-ongoing, **community estimates it will take +1M LoC**

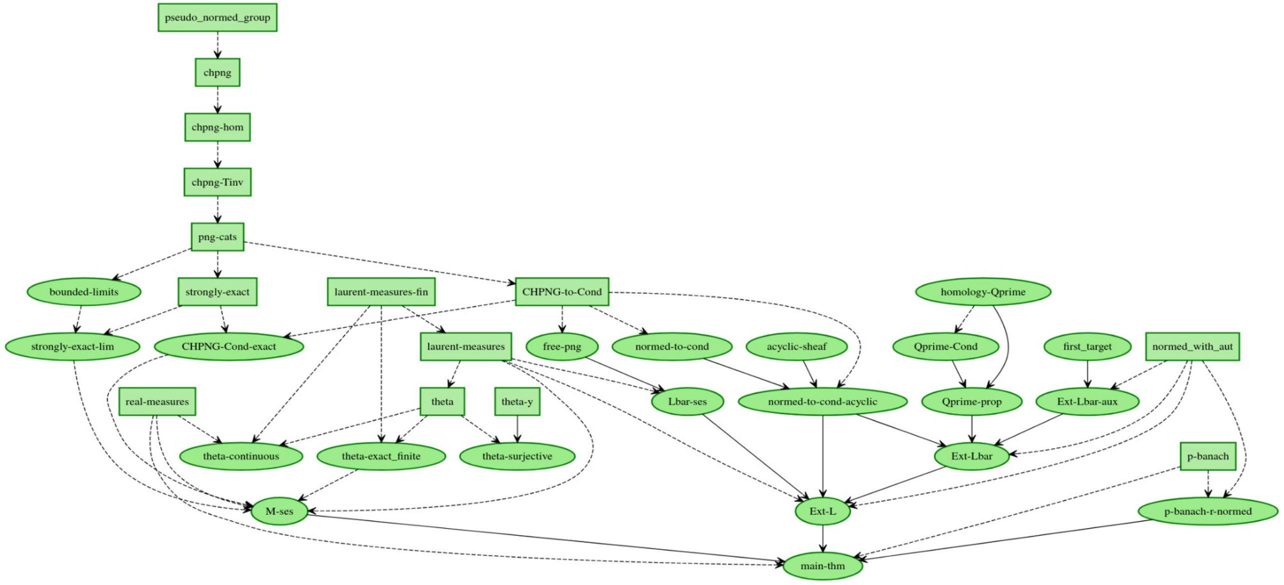
Lean Blueprint

Developed by Patrick Massot

Connects informal and formal mathematics

Used in many Lean projects: LTE, Sphere Eversion, PFR, FLT, Prime Number Theorem and Beyond

https://github.com/PatrickMassot/leanblueprint



Lean Blueprint

PFR

Lemma 8.2. (Constructing good variables, I')✓

One has

$$k \leq \delta + \eta(d[X_1^0; T_1|T_3] - d[X_1^0; X_1]) + \eta(d[X_2^0; T_2|T_3] - d[X_2^0; X_2]).$$

Proof ▾

We apply Lemma 3.23 with $(A, B) = (T_1, T_2)$ there. Since $T_1 + T_2 = T_3$, the conclusion is that

$$\sum_{t_3} \mathbb{P}[T_3 = t_3] d[(T_1|T_3 = t_3); (T_2|T_3 = t_3)]$$
$$\leq 3\mathbb{I}[T_1 : T_2] + 2\mathbb{H}[T_3] - \mathbb{H}[T_1] - \mathbb{H}[T_2].$$

The right-hand side in (1) can be rearranged as

$$2(\mathbb{H}[T_1] + \mathbb{H}[T_2] + \mathbb{H}[T_3]) - 3\mathbb{H}[T_1, T_2]$$
$$= 2(\mathbb{H}[T_1] + \mathbb{H}[T_2] + \mathbb{H}[T_3]) - \mathbb{H}[T_1, T_2] - \mathbb{H}[T_2, T_3] - \mathbb{H}[T_1, T_3] = \delta,$$

using the fact (from Lemma 2.2) that all three terms $\mathbb{H}[T_i, T_j]$ are equal to $\mathbb{H}[T_1, T_2, T_3]$ and hence to each other. We also have

$$\sum_{t_3} P[T_3 = t_3] \big(d[X_1^0; (T_1|T_3 = t_3)] - d[X_1^0; X_1] \big)$$
$$= d[X_1^0; T_1|T_3] - d[X_1^0; X_1]$$

...

Lean Blueprint

PFR

Lemma 8.2. (Constructing good variables, I')✓# L∃∀N

One has

$$k \leq \delta + \eta(d[X_1^0; T_1|T_3] - d[X_1^0; X_1]) + \eta(d[X_2^0; T_2|T_3] - d[X_2^0; X_2]).$$

Proof ▾

We apply Lemma 3.23 with $(A, B) = (T_1, T_2)$ there. Since $T_1 + T_2 = T_3$, the conclusion is that

$$\sum_{t_3} \mathbb{P}[T_3 = t_3] d[(T_1|T_3 = t_3); (T_2|T_3 = t_3)]$$
$$\leq 3\mathbb{I}[T_1 : T_2] + 2\mathbb{H}[T_3] - \mathbb{H}[T_1] - \mathbb{H}[T_2].$$

The right-hand side in (1) can be rearranged as

$$2(\mathbb{H}[T_1] + \mathbb{H}[T_2] + \mathbb{H}[T_3]) - 3\mathbb{H}[T_1, T_2]$$
$$= 2(\mathbb{H}[T_1] + \mathbb{H}[T_2] + \mathbb{H}[T_3]) - \mathbb{H}[T_1, T_2] - \mathbb{H}[T_2, T_3] - \mathbb{H}[T_1, T_3] = \delta,$$

using the fact (from Lemma 2.2) that all three terms $\mathbb{H}[T_i, T_j]$ are equal to $\mathbb{H}[T_1, T_2, T_3]$ and hence to each other. We also have

$$\sum_{t_3} P[T_3 = t_3] \big(d[X_1^0; (T_1|T_3 = t_3)] - d[X_1^0; X_1]\big)$$
$$= d[X_1^0; T_1|T_3] - d[X_1^0; X_1]$$

...

```
theorem construct_good_prelim'                                                    source
        {G : Type u_1} [AddCommGroup G] [Fintype G] [hG : MeasurableSpace G]
        [MeasurableSingletonClass G] [ElementaryAddCommGroup G 2] [MeasurableAdd₂ G] {Ω₀₁ : Type u_2}
        {Ω₀₂ : Type u_3} [MeasureTheory.MeasureSpace Ω₀₁] [MeasureTheory.MeasureSpace Ω₀₂]
        {p : refPackage Ω₀₁ Ω₀₂ G} {Ω : Type u_4} [MeasureTheory.MeasureSpace Ω] {X₁ : Ω → G} {X₂ : Ω → G}
        (h_min : tau_minimizes p X₁ X₂) {Ω' : Type u_5} [MeasureTheory.MeasureSpace Ω']
        [MeasureTheory.IsProbabilityMeasure MeasureTheory.volume] {T₁ : Ω' → G} {T₂ : Ω' → G}
        {T₃ : Ω' → G} (hT : T₁ + T₂ + T₃ = 0) (hT₁ : Measurable T₁) (hT₂ : Measurable T₂)
        (hT₃ : Measurable T₃) :
    d[X₁ # X₂] ≤
     I[T₁ : T₂] + I[T₂ : T₃] + I[T₃ : T₁] +
      p.η * (d[p.X₀₁ # T₁ | T₃] - d[p.X₀₁ # X₁] + (d[p.X₀₂ # T₂ | T₃] - d[p.X₀₂ # X₂]))
```

For any T_1, T_2, T_3 adding up to 0, then k is at most

$$\delta + \eta(d[X_1^0; T_1|T_3] - d[X_1^0; X_1]) + \eta(d[X_2^0; T_2|T_3] - d[X_2^0; X_2])$$

where $\delta = I[T_1 : T_2; \mu] + I[T_2 : T_3; \mu] + I[T_3 : T_1; \mu]$.

Lean Blueprint

PFR

Lemma 8.2. (Constructing good variables, I′) ✓# LƎ∀N

One has

$$k \leq \delta + \eta(d[X_1^0; T_1|T_3] - d[X_1^0; X_1]) + \eta(d[X_2^0; T_2|T_3] - d[X_2^0; X_2]).$$

Proof ▾

We apply Lemma 3.23 with $(A, B) = (T_1, T_2)$ there. Since $T_1 + T_2 = T_3$, the conclusion is that

$$\sum_{t_3} \mathbb{P}[T_3 = t_3] d[(T_1|T_3 = t_3); (T_2|T_3 = t_3)] \leq 3\mathbb{I}[T_1 : T_2] + 2\mathbb{H}[T_3] - \mathbb{H}[T_1] - \mathbb{H}[T_2].$$

The right-hand side in (1) can be rearranged as

$$\begin{aligned} &2(\mathbb{H}[T_1] + \mathbb{H}[T_2] + \mathbb{H}[T_3]) - 3\mathbb{H}[T_1, T_2] \\ &= 2(\mathbb{H}[T_1] + \mathbb{H}[T_2] + \mathbb{H}[T_3]) - \mathbb{H}[T_1, T_2] - \mathbb{H}[T_2, T_3] - \mathbb{H}[T_1, T_3] = \delta, \end{aligned}$$

using the fact (from Lemma 2.2) that all three terms $\mathbb{H}[T_i, T_j]$ are equal to $\mathbb{H}[T_1, T_2, T_3]$ and hence to each other. We also have

$$\begin{aligned} &\sum_{t_3} P[T_3 = t_3]\big(d[X_1^0; (T_1|T_3 = t_3)] - d[X_1^0; X_1]\big) \\ &= d[X_1^0; T_1|T_3] - d[X_1^0; X_1] \end{aligned}$$

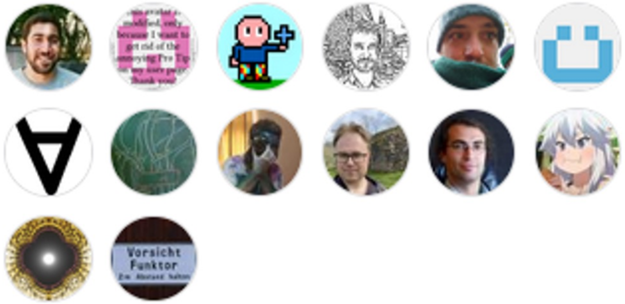
...

```
theorem construct_good_prelim'                                                    source
        {G : Type u_1} [AddCommGroup G] [Fintype G] [hG : MeasurableSpace G]
        [MeasurableSingletonClass G] [ElementaryAddCommGroup G 2] [MeasurableAdd₂ G] {Ω₀₁ : Type u_2}
        {Ω₀₂ : Type u_3} [MeasureTheory.MeasureSpace Ω₀₁] [MeasureTheory.MeasureSpace Ω₀₂]
        {p : refPackage Ω₀₁ Ω₀₂ G} {Ω : Type u_4} [MeasureTheory.MeasureSpace Ω] {X₁ : Ω → G} {X₂ : Ω → G}
        (h_min : tau_minimizes p X₁ X₂) {Ω' : Type u_5} [MeasureTheory.MeasureSpace Ω']
        [MeasureTheory.IsProbabilityMeasure MeasureTheory.volume] {T₁ : Ω' → G} {T₂ : Ω' → G}
        {T₃ : Ω' → G} (hT : T₁ + T₂ + T₃ = 0) (hT₁ : Measurable T₁) (hT₂ : Measurable T₂)
        (hT₃ : Measurable T₃) :
    d[X₁ # X₂] ≤
     I[T₁ : T₂] + I[T₂ : T₃] + I[T₃ : T₁] +
```

```
lemma construct_good_prelim' : k ≤ δ + p.η * c[T₁ | T₃ # T₂ | T₃] := by
  let sum1 : ℝ := (Measure.map T₃ ℙ)[fun t ↦ d[T₁; ℙ[|T₃ ⁻¹' {t}] # T₂; ℙ[|T₃ ⁻¹' {t}]]]
  let sum2 : ℝ := (Measure.map T₃ ℙ)[fun t ↦ d[p.X₀₁; ℙ # T₁; ℙ[|T₃ ⁻¹' {t}]] - d[p.X₀₁ # X₁]]
  let sum3 : ℝ := (Measure.map T₃ ℙ)[fun t ↦ d[p.X₀₂; ℙ # T₂; ℙ[|T₃ ⁻¹' {t}]] - d[p.X₀₂ # X₂]]
  let sum4 : ℝ := (Measure.map T₃ ℙ)[fun t ↦ ψ[T₁; ℙ[|T₃ ⁻¹' {t}] # T₂; ℙ[|T₃ ⁻¹' {t}]]]
  have h2T₃ : T₃ = T₁ + T₂ := by
    calc T₃ = T₁ + T₂ + T₃ - T₃ := by rw [hT, zero_sub]; simp
      _ = T₁ + T₂ := by rw [add_sub_cancel]
  have hP : IsProbabilityMeasure (Measure.map T₃ ℙ) := isProbabilityMeasure_map hT₃.aemeasurable
  -- control sum1 with entropic BSG
  have h1 : sum1 ≤ δ := by
    have h1 : sum1 ≤ 3 * I[T₁ : T₂] + 2 * H[T₃] - H[T₁] - H[T₂] := by
      subst h2T₃; exact ent_bsg hT₁ hT₂
    have h2 : H[⟨T₂, T₃⟩] = H[⟨T₁, T₂⟩]
    · rw [h2T₃, entropy_add_right', entropy_comm] <;> assumption
    have h3 : H[⟨T₁, T₂⟩] = H[⟨T₃, T₁⟩]
    · rw [h2T₃, entropy_add_left, entropy_comm] <;> assumption
    simp_rw [mutualInfo_def] at h1 ⊢; linarith
  -- rewrite sum2 and sum3 as Rusza distances
  have h2 : sum2 = d[p.X₀₁ # T₁ | T₃] - d[p.X₀₁ # X₁] := by
   ...
```

Takeaway:
Formal Proofs enable **Crowd-Sourced Mathematics**

Contributors 29



+ 15 contributors

“The beauty of the system: you do not have to understand the whole proof of FLT in order to contribute. The **blueprint breaks down the proof into many many small lemmas**, and if you can formalise a proof of just one of those lemmas then I am eagerly awaiting your pull request.”

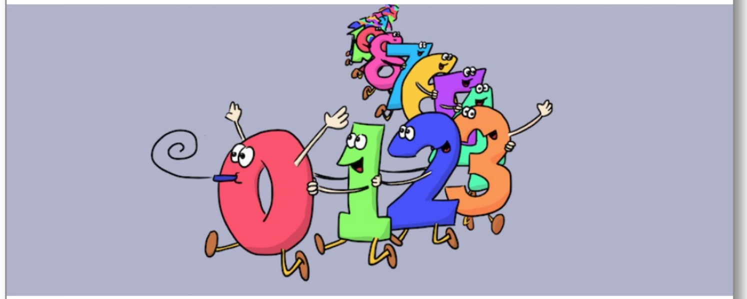
Kevin Buzzard on the FLT Project

Lean Game Server

https://adam.math.hhu.de/

Natural Number Game

The classical introduction game for Lean.



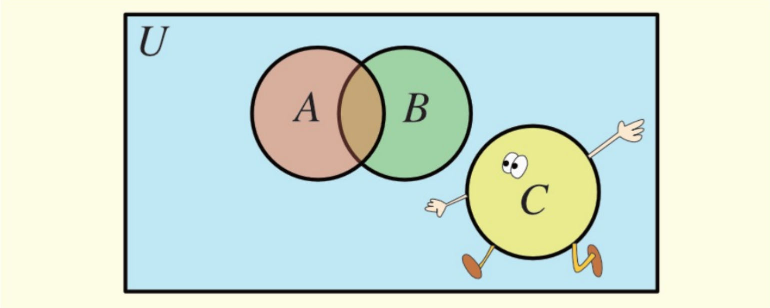
In this game you recreate the natural numbers $\mathbb{N}$ from the Peano axioms, learning the basics about theorem proving in Lean.

This is a good first introduction to Lean!

Prerequisites	
Worlds	9
Levels	79
Language	🇬🇧🇨🇳

Set Theory Game

A game about set theory.



In this game you will learn the basics of theorem proving in Lean by proving theorems about unions, intersections, and complements of sets.

Prerequisites	
Worlds	8
Levels	51
Language	🇬🇧🇪🇸

Takeaway: **Extensibility**

Mathlib is not just math, but many Lean extensions too

Users extend Lean using Lean itself

Users can browse and access Lean data-structures: **Math as data**

We wrote Lean 4 in Lean to ensure the system is very extensible

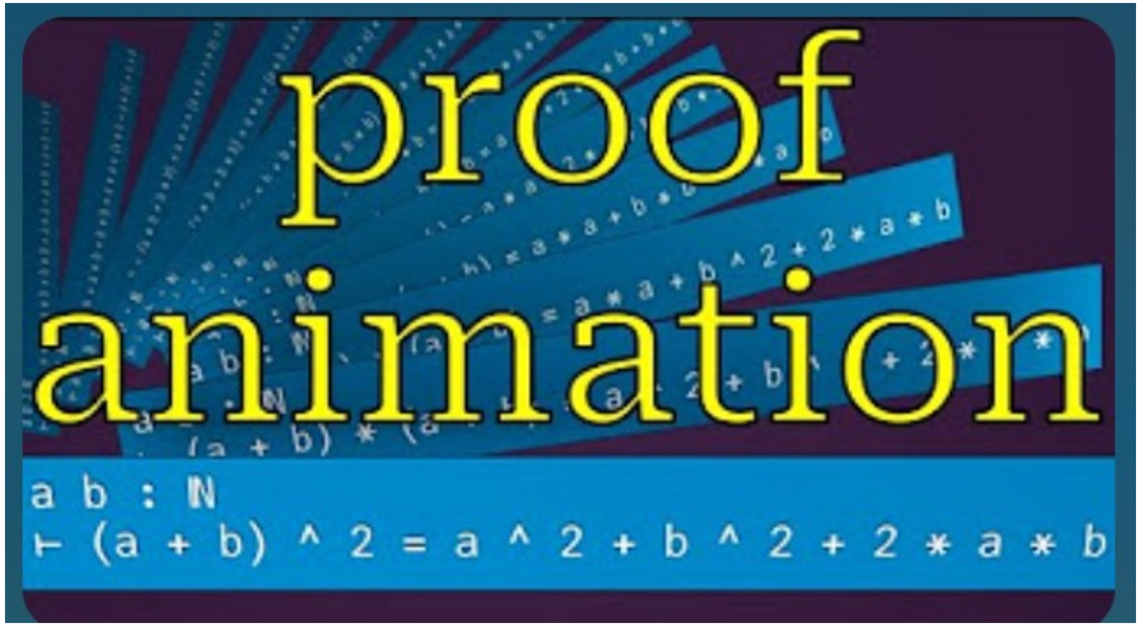
```
elab "ring" : tactic => do
  let g ← getMainTarget
  match g.getAppFnArgs with
  | (`Eq, #[ty, e₁, e₂]) =>
    let ((e₁', p₁), (e₂', p₂)) ← RingM.run ty $ do (← eval e₁, ← eval e₂)
    if ← isDefEq e₁' e₂' then
      let p ← mkEqTrans p₁ (← mkEqSymm p₂)
      ensureHasNoMVars p
      assignExprMVar (← getMainGoal) p
      replaceMainGoal []
    else
      throwError "failed \n{← e₁'.pp}\n{← e₂'.pp}"
  | _ => throwError "failed: not an equality"
```

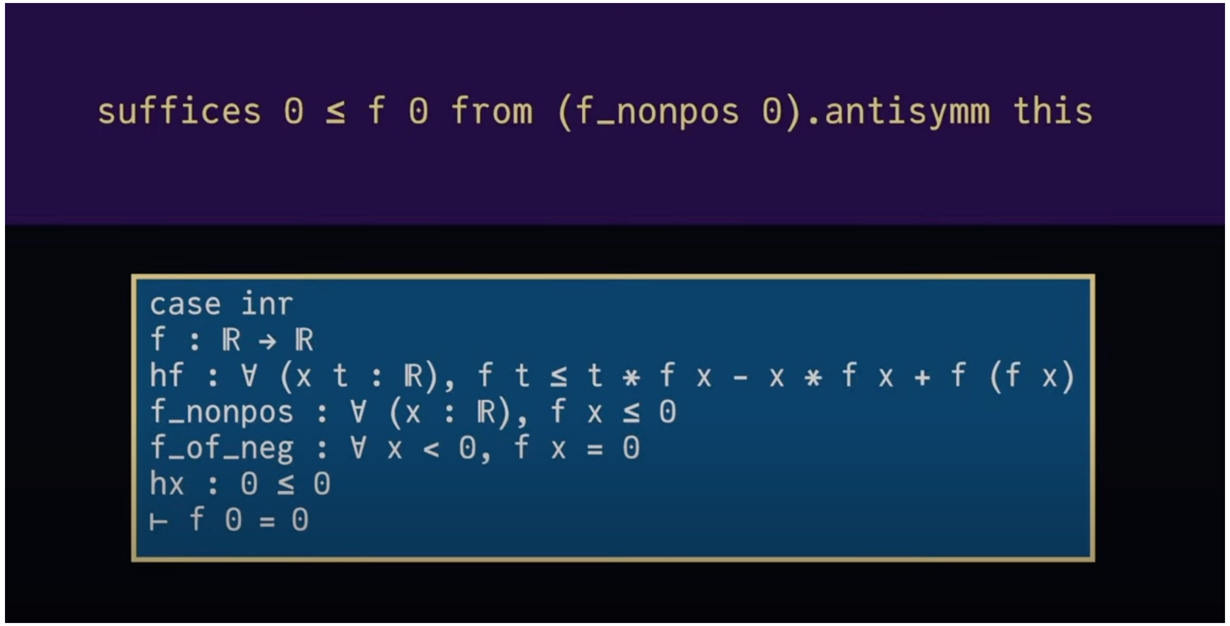
Visualizing Lean 4 Proofs in Blender

David Renshaw - https://github.com/dwrensha/animate-lean-proofs

A tool for turning Lean proofs into Blender animations

YouTube video





SciLean: Scientific Computing Assistant

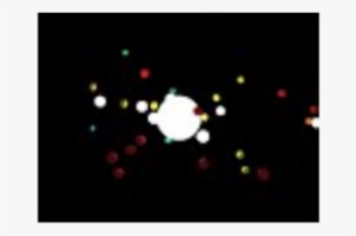
Tomáš Skřivan - https://github.com/lecopivo/SciLean

Framework for scientific computing such as solving differential equations, optimization or machine learning written in Lean

lean4 **Lean 4 as a scripting language in Houdini** May 15

Tomas Skrivan EDITED 8:02 AM

Some more fun with Hamiltonian systems:
https://www.youtube.com/watch?v=qcE9hFPgYkg&ab_channel=Lecopivo

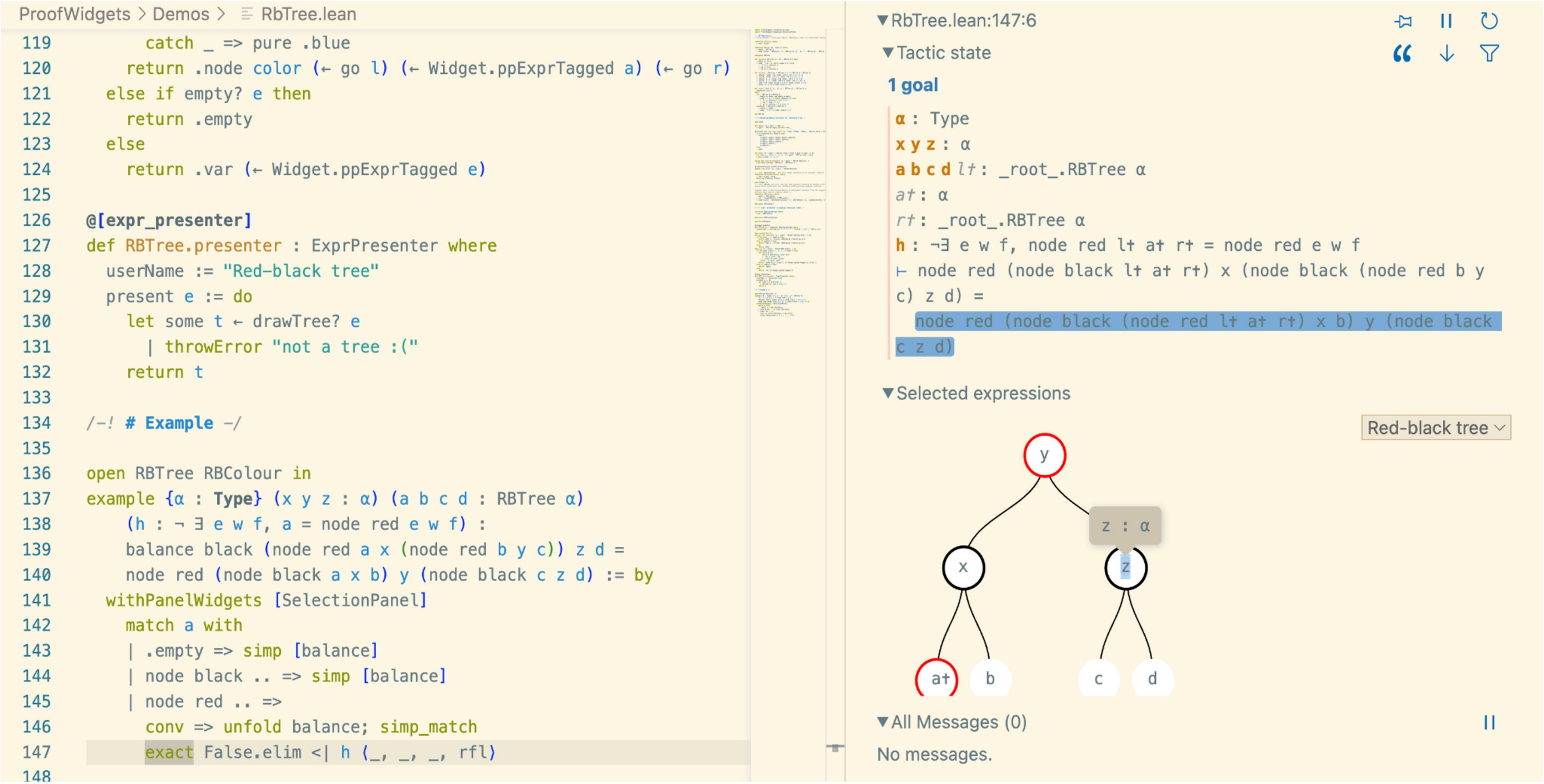


Macros in Lean are really cool, I can now annotate function arguments and automatically generate functions derivatives and proofs of smoothness. The Hamiltonian definition for the above system is defined as:

```
def LennardJones (ε minEnergy : ℝ) (radius : ℝ) (x : ℝ^(3:ℕ)) : ℝ :=
  let x' := ‖1/radius * x‖^{-6, ε}
  4 * minEnergy * x' * (x' - 1)
argument x [Fact (ε≠0)]
  isSmooth, diff, hasAdjDiff, adjDiff
```

ProofWidgets

Wojciech Nawrocki and Ed Ayers



Lean enables AI for math

Large language models are incredibly powerful, but they “**hallucinate**.”

Show that there are infinitely many integers between two distinct integers.

To show there are infinitely many integers between two distinct integers, let's assume the contrary and see if we reach a contradiction.

Assumption: There are a finite number of integers between two distinct integers a and b, where $a < b$.

If our assumption is true, then there exists a largest integer m such that $a < m < b$. Now, consider the integer $m + 1$. Clearly, $m < m + 1 \leq b$. This means $m + 1$ is another integer between a and b, which contradicts our assumption that m was the largest integer between a and b.

Hence, our assumption that there are a finite number of integers between two distinct integers a and b is false.

Therefore, there are infinitely many integers between any two distinct integers a and b.

Lean enables AI for math

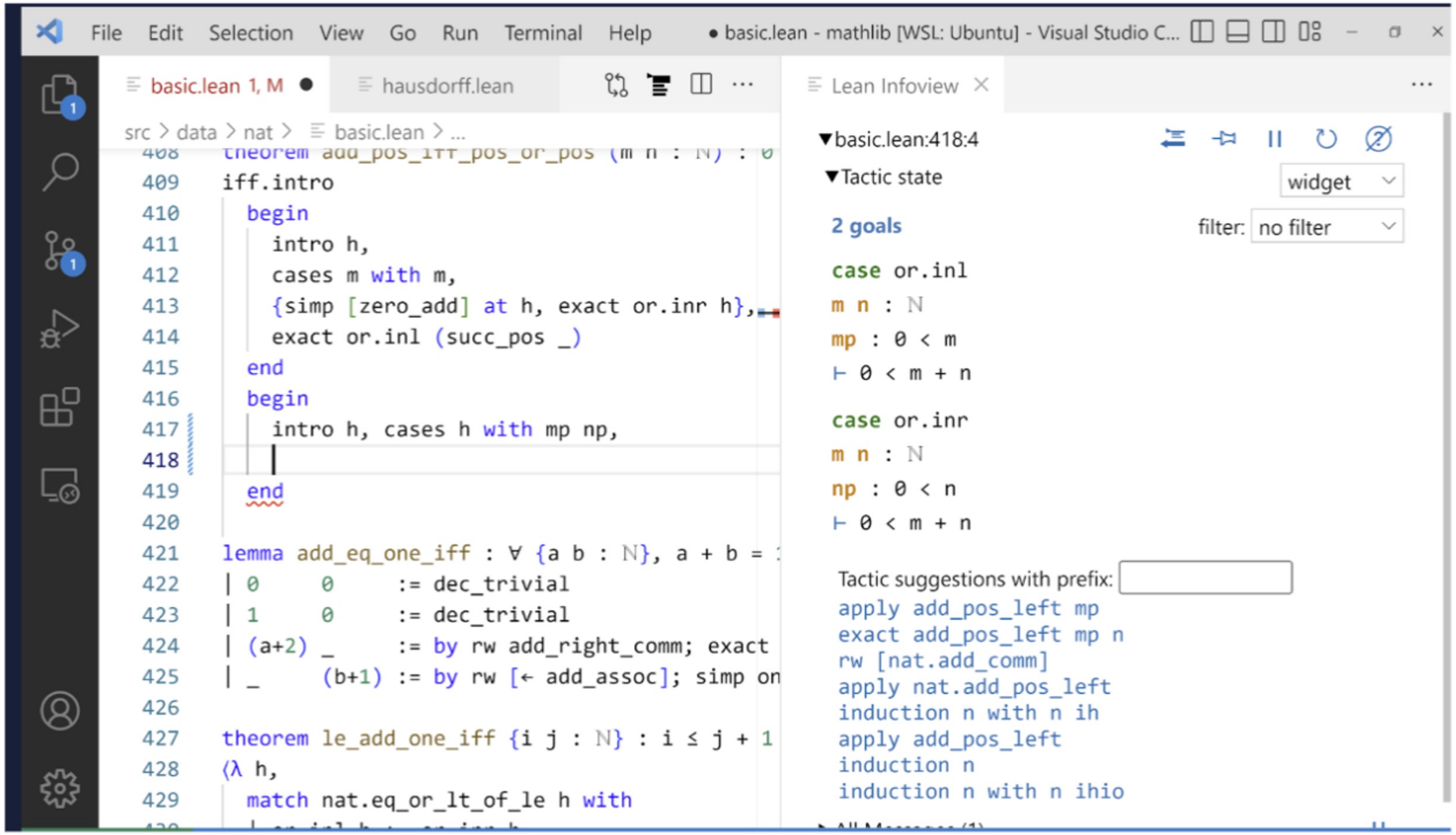
OpenAI – GPTf – Solving (Some) Formal Math Olympiad Problems with Lean

```
theorem imo_1964_p2
  (a b c : ℝ)
  (h₀ : 0 < a ∧ 0 < b ∧ 0 < c)
  (h₁ : c < a + b)
  (h₂ : b < a + c)
  (h₃ : a < b + c) :
  a^2 * (b + c - a) + b^2 * (c + a - b) + c^2 * (a + b - c)
    ≤ 3 * a * b * c :=
begin
  -- Arguments to `nlinarith` are fully invented by our model.
  nlinarith [sq_nonneg (b - a),
             sq_nonneg (c - b),
             sq_nonneg (c - a)]
end
```

nlarith is a tactic (aka proof automation procedure)

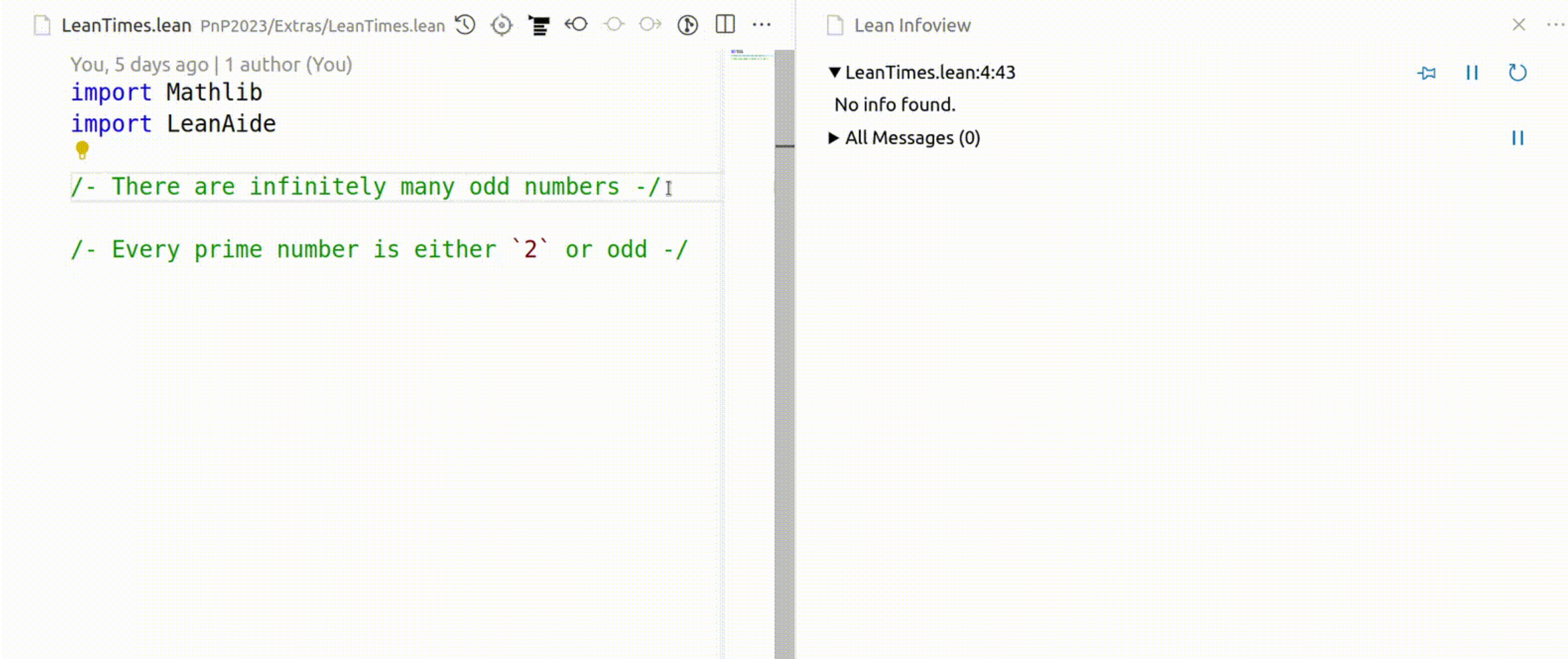
Lean enables AI for math

Meta – HyperTree Proof Search for Neural Theorem Proving



LeanAide – Autoformalization

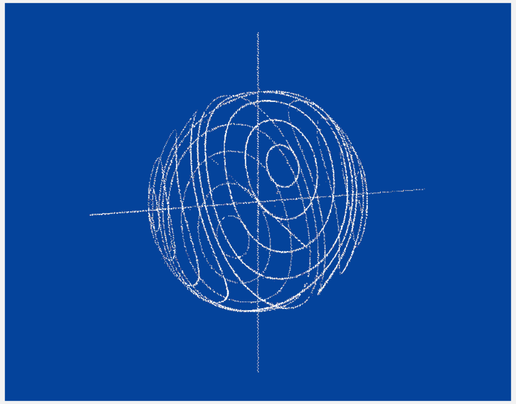
https://github.com/siddhartha-gadgil/LeanAide



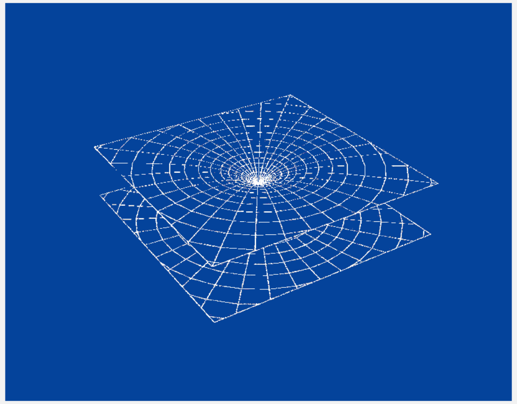
Harmonic.fun – Mathematical Superintelligence

https://harmonic.fun/

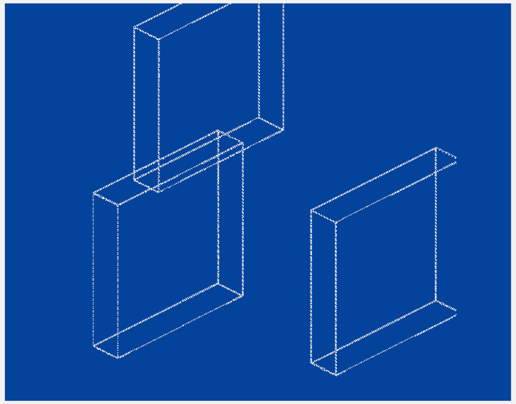
1 Building AI that is truthful, with verifiably correct and interpretable outputs



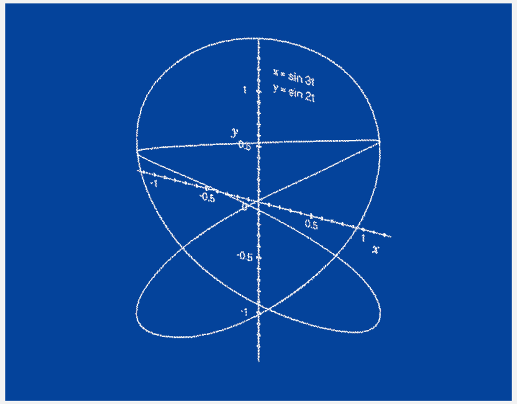
2 Changing the way mathematics is learned and taught in schools



3 Accelerating the advent of verified software synthesis in safety-critical domains



4 Solving open problems in mathematics, science, and beyond



A quick Lean tour

SSFT 2024 summer school – David Christiansen – https://github.com/david-christiansen/ssft24

```
inductive Expr where
  | const (i : BitVec 32)
  | var (name : String)
  | un (op : Expr.UnOp) (e : Expr)
  | bin (op : Expr.BinOp) (e1 e2 : Expr)
deriving Repr, DecidableEq
```

…

```
/-- Boolean conjunction -/
syntax:35 exp:35 " && " exp:36 : exp
/-- Boolean disjunction -/
syntax:35 exp:35 " || " exp:36 : exp
/-- Parens -/
syntax "(" exp ")" : exp
/-- Escape to Lean -/
syntax:max "~" term:max : exp

syntax:min "expr " "{ " exp " }" : term

open Lean in
macro_rules
  | `(expr{$x:ident}) => `(Expr.var $(quote x.getId.toString))
  | `(expr{$n:num}) => `(Expr.const $(quote n.getNat))
  | `(expr{$e1 && $e2}) => `(Expr.bin .and (expr{$e1}) (expr{$e2}))
  | `(expr{$e1 || $e2}) => `(Expr.bin .or (expr{$e1}) (expr{$e2}))
```

…

A quick Lean tour

```
inductive Stmt where
  | skip
  | seq (stmt1 stmt2 : Stmt)
  | assign (name : String) (val : Expr)
  | if (cond : Expr) (ifTrue ifFalse : Stmt)
  | while (cond : Expr) (body : Stmt)
deriving Repr, DecidableEq
```

...

```
syntax:min "imp" ppHardSpace "{" ppLine stmt ppDedent(ppLine "}") : term
```

...

```
def fact : Stmt := imp {
  out := 1;
  while (n > 0) {
    out := out * n;
    n := n - 1;
  }
}
```

A quick Lean tour

```lean
/--
Optimizes an expression by folding constants.
-/
def optimize : Expr → Expr
  | .const i => .const i
  | .var x => .var x
  | .un op e =>
    match optimize e with
    | .const i =>
      if let some v := op.apply i then .const v
      else .un op (.const i)
    | e' => .un op e'
  | .bin op e1 e2 =>
    match optimize e1, optimize e2 with
    | .const i, .const i' =>
      if let some v := op.apply i i' then .const v
      else .bin op (.const i) (.const i')
    | e1', e2' => .bin op e1' e2'
```

```lean
def optimize : Stmt → Stmt
  | imp {skip;} => imp {skip;}
  | imp {~s1 ~s2} =>
    match s1.optimize, s2.optimize with
    | imp {skip;}, s2' => s2'
    | s1', imp {skip;} => s1'
    | s1', s2' => imp {~s1' ~s2'}
  | imp {if (~c) {~s1} else {~s2}} =>
    let c' := c.optimize
    match c' with
    | .const 0 => s2.optimize
    | .const _ => s1.optimize
    | _ =>
      let s1' := s1.optimize
      let s2' := s2.optimize
      if s1' = s2' then
        s1'
      else imp {if (~c') {~s1.optimize} else {~s2.optimize}}
```

...

A quick Lean tour

```
inductive BigStep : Env → Stmt → Env → Prop where
  | skip :
    BigStep ρ (imp {skip;}) ρ
  | seq :
    BigStep ρ s1 ρ' → BigStep ρ' s2 ρ'' →
    BigStep ρ (imp{ ~s1 ~s2}) ρ''
  | assign :
    e.eval ρ = some v →
    BigStep ρ (imp {~x := ~e;}) (ρ.set x v)
  | ifTrue :
    Truthy (c.eval ρ) → BigStep ρ s1 ρ' →
    BigStep ρ (imp {if (~c) {~s1} else {~s2}}) ρ'
  | ifFalse :
    Falsy (c.eval ρ) → BigStep ρ s2 ρ' →
    BigStep ρ (imp {if (~c) {~s1} else {~s2}}) ρ'
```

...

```
theorem optimize_ok : BigStep ρ s ρ' → BigStep ρ s.optimize ρ' := by
  intro h
  induction h with simp only [optimize]
  | «skip» => constructor
  | seq s1 s2 ih1 ih2 =>
    split
    next eq2 =>
      rw [eq2] at ih1
      cases ih1; apply ih2
    next eq1 eq2 =>
      rw [eq1] at ih2
      cases ih2; apply ih1
    next =>
      apply BigStep.seq ih1 ih2
```

...

A quick Lean tour

```
def run (ρ : Env) (s : Stmt) : Nat → Option Env
  | 0 => none
  | n + 1 =>
    match s with
    | imp {skip;} =>
      some ρ
    | imp {~s1 ~s2} => do
      let ρ' ← run ρ s1 n
      run ρ' s2 n
    | imp {~x := ~e;} => do
      let v ← e.eval ρ
      pure (ρ.set x v)
    | imp {if (~c) {~s1} else {~s2}} => do
      let v ← c.eval ρ
      if v = 0 then
        run ρ s2 n
      else
        run ρ s1 n
```

...

```
theorem run'_correct : run ρ s n = some ρ' → BigStep ρ s ρ' := by
  intro term
```

...

A quick Lean tour

```
def popcount : Stmt := imp {
  x := x - ((x >>> 1) &&& 0x55555555);
  x := (x &&& 0x33333333) + ((x >>> 2) &&& 0x33333333);
  x := (x + (x >>> 4)) &&& 0x0F0F0F0F;
  x := x + (x >>> 8);
  x := x + (x >>> 16);
  x := x &&& 0x0000003F;
}
```

```
def pop_spec (x : BitVec 32) : BitVec 32 :=
  go x 0 32
where
  go (x : BitVec 32) (pop : BitVec 32) (i : Nat) : BitVec 32 :=
    match i with
    | 0 => pop
    | i + 1 =>
      let pop := pop + (x &&& 1#32)
      go (x >>> 1#32) pop i
```

```
theorem popcount_correct :
    ∃ ρ, (run (Env.init x) popcount 8) = some ρ ∧ ρ "x" = pop_spec x := by
  simp [run, popcount, Expr.eval, Expr.BinOp.apply, Env.set, Value, pop_spec, pop_spec.go]
  bv_decide
```

A quick Lean tour

Imp.lean > { } Imp.Stmt > popcount_correct

```lean
theorem popcount_correct :
    ∃ p, (run (Env.init x) popcount 8) = some p
  simp [run, popcount, Expr.eval, Expr.BinOp.app
  bv_decide

```

▼ Tactic state

1 goal

```
x : Value
⊢ ((x - (x >>> 1 &&& 1431655765#32) &&& 858993459#32) + ((x - (x >>> 1 &&&
1431655765#32)) >>> 2 &&& 858993459#32) +
            ((x - (x >>> 1 &&& 1431655765#32) &&& 858993459#32) +
                ((x - (x >>> 1 &&& 1431655765#32)) >>> 2 &&& 858993459#32)) >>>
              4 &&&
          252645135#32) +
        ((x - (x >>> 1 &&& 1431655765#32) &&& 858993459#32) +
                ((x - (x >>> 1 &&& 1431655765#32)) >>> 2 &&& 858993459#32) +
              ((x - (x >>> 1 &&& 1431655765#32) &&& 858993459#32) +
                  ((x - (x >>> 1 &&& 1431655765#32)) >>> 2 &&& 858993459#32)) >>>
                4 &&&
            252645135#32) >>>
          8 +
      (((x - (x >>> 1 &&& 1431655765#32) &&& 858993459#32) +
                ((x - (x >>> 1 &&& 1431655765#32)) >>> 2 &&& 858993459#32) +
              ((x - (x >>> 1 &&& 1431655765#32) &&& 858993459#32) +
                  ((x - (x >>> 1 &&& 1431655765#32)) >>> 2 &&& 858993459#32)) >>>
                4 &&&
            252645135#32) +
          ((x - (x >>> 1 &&& 1431655765#32) &&& 858993459#32) +
                  ((x - (x >>> 1 &&& 1431655765#32)) >>> 2 &&& 858993459#32) +
                ((x - (x >>> 1 &&& 1431655765#32) &&& 858993459#32) +
                    ((x - (x >>> 1 &&& 1431655765#32)) >>> 2 &&& 858993459#32)) >>>
                  4 &&&
              252645135#32) >>>
            8) >>>
        16 &&&
    63#32 =
  (x &&& 1#32) + (x >>> 1 &&& 1#32) + (x >>> 2 &&& 1#32) + (x >>> 3 &&& 1#32) + (x >>>
4 &&& 1#32) +
```

LeanSAT – A verified bit-blaster

- **Henrik Boving**, Josh Clune, Siddharth Bhat, and Alex Keizer
- Uses LRAT proof producing SAT solvers: **Cadical**
- SAT tactics: `sat_decide`, `sat_decide?`, `sat_check <lrat-proof>`
- Bit-vector tactics: `bv_decide`, `bv_decide?`, `bv_check <lrat-proof>`
- Simplify => Reflect => Bit-blast => AIG => CNF => SAT-solver => LRAT Proof => Verified checker
- **Implemented in Lean**

```
/-
Close a goal by:
1. Turning it into a BitVec problem.
2. Using bitblasting to turn that into a SAT problem.
3. Running an external SAT solver on it and obtaining an LRAT proof from it.
4. Verifying the LRAT proof using proof by reflection.
-/
syntax (name := bvDecideSyntax) "bv_decide" : tactic
```

LeanSAT – A verified bit-blaster

```
def _root_.Lean.MVarId.bvDecide (g : MVarId) (cfg : TacticContext) : MetaM Result := do
  let ⟨g?, simpTrace⟩ ← g.bvNormalize
  let some g := g? | return ⟨simpTrace, none⟩
  let lratCert ← g.bvUnsat cfg
  return ⟨simpTrace, some lratCert⟩
```

```
def _root_.Lean.MVarId.bvUnsat (g : MVarId) (cfg : TacticContext) : MetaM LratCert := M.run do
  let unsatProver : UnsatProver := fun bvExpr atomsAssignment => do
    withTraceNode `bv (fun _ => return "Preparing LRAT reflection term") do
      lratBitblaster cfg bvExpr atomsAssignment
  g.closeWithBVReflection unsatProver
```

```
def lratBitblaster (cfg : TacticContext) (bv : BVLogicalExpr)
    (atomsAssignment : Std.HashMap Nat Expr) : MetaM UnsatProver.Result := do
  let entry ←
    withTraceNode `bv (fun _ => return "Bitblasting BVLogicalExpr to AIG") do
      -- lazyPure to prevent compiler lifting
      IO.lazyPure (fun _ => bv.bitblast)
  let aigSize := entry.aig.decls.size
  trace[bv] s!"AIG has {aigSize} nodes."
```

…

LeanSAT – A verified bit-blaster

```
def verifyBVExpr (bv : BVLogicalExpr) (cert : LratCert) : Bool :=
  verifyCert (LratFormula.ofCnf (AIG.toCNF bv.bitblast.relabelNat)) cert

theorem unsat_of_verifyBVExpr_eq_true (bv : BVLogicalExpr) (c : LratCert)
    (h : verifyBVExpr bv c = true) : bv.unsat := by
  apply BVLogicalExpr.unsat_of_bitblast
  rw [← AIG.Entrypoint.relabelNat_unsat_iff]
  rw [← AIG.toCNF_equisat]
  apply verifyCert_correct
  rw [verifyBVExpr] at h
  assumption
```

```
theorem verifyCert_correct : ∀ cnf cert, verifyCert (LratFormula.ofCnf cnf) cert = true → cnf.unsat := by
  intro c b h1
  dsimp[verifyCert] at h1
  ...
```

LeanSAT – A verified bit-blaster

```
theorem simple (x : BitVec 64) : x + x = 2 * x := by
  bv_decide?
```

Quick Fix

Try this: bv_check "Arith.lean-simple-43-2.lrat"

Proof search and replay-ability

```
theorem simple (x : BitVec 64) : x + x = 2 * x := by
  bv_decide?
```

Quick Fix

Try this: bv_check "Arith.lean-simple-43-2.lrat"

```
theorem simple (x : BitVec 64) : x + x = 2 * x := by
  bv_check "Arith.lean-simple-43-2.lrat"
```

Many other tactics implement this idiom: simp?, aesop?, etc.

Lean at AWS

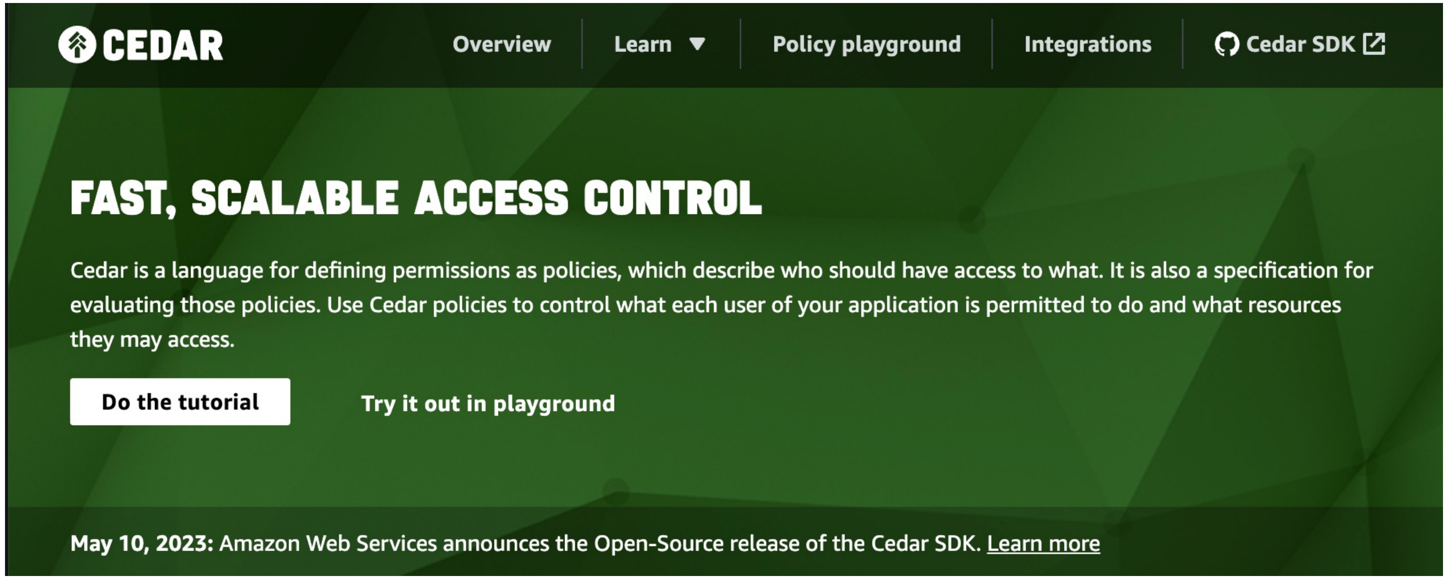
- Automated Reasoning Group
  "The Business of Proof"

- Open-source projects
  - Cedar – https://github.com/cedar-policy/cedar-spec
  - SampCert – https://github.com/leanprover/SampCert
  - LNSym – https://github.com/leanprover/LNSym
  - AILean – coming soon

- Blog post at Amazon.Science coming soon.
- Many more projects coming soon.

Cedar

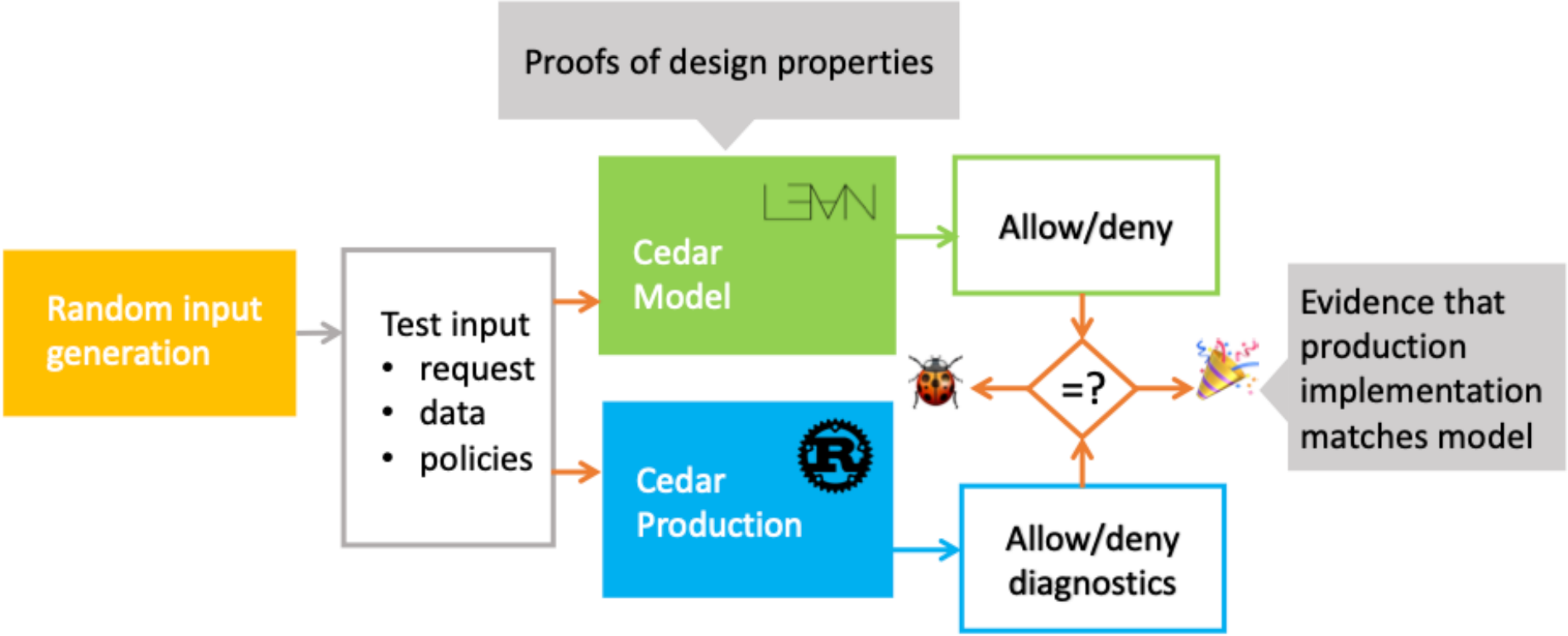
https://www.cedarpolicy.com/



https://github.com/cedar-policy/cedar-spec

```
def isAuthorized (req : Request) (entities : Entities) (policies : Policies) : Response :=
  let forbids := satisfiedPolicies .forbid policies req entities
  let permits := satisfiedPolicies .permit policies req entities
  let erroringPolicies := errorPolicies policies req entities
  if forbids.isEmpty && !permits.isEmpty
  then { decision := .allow, determiningPolicies := permits, erroringPolicies }
  else { decision := .deny,  determiningPolicies := forbids, erroringPolicies }
```

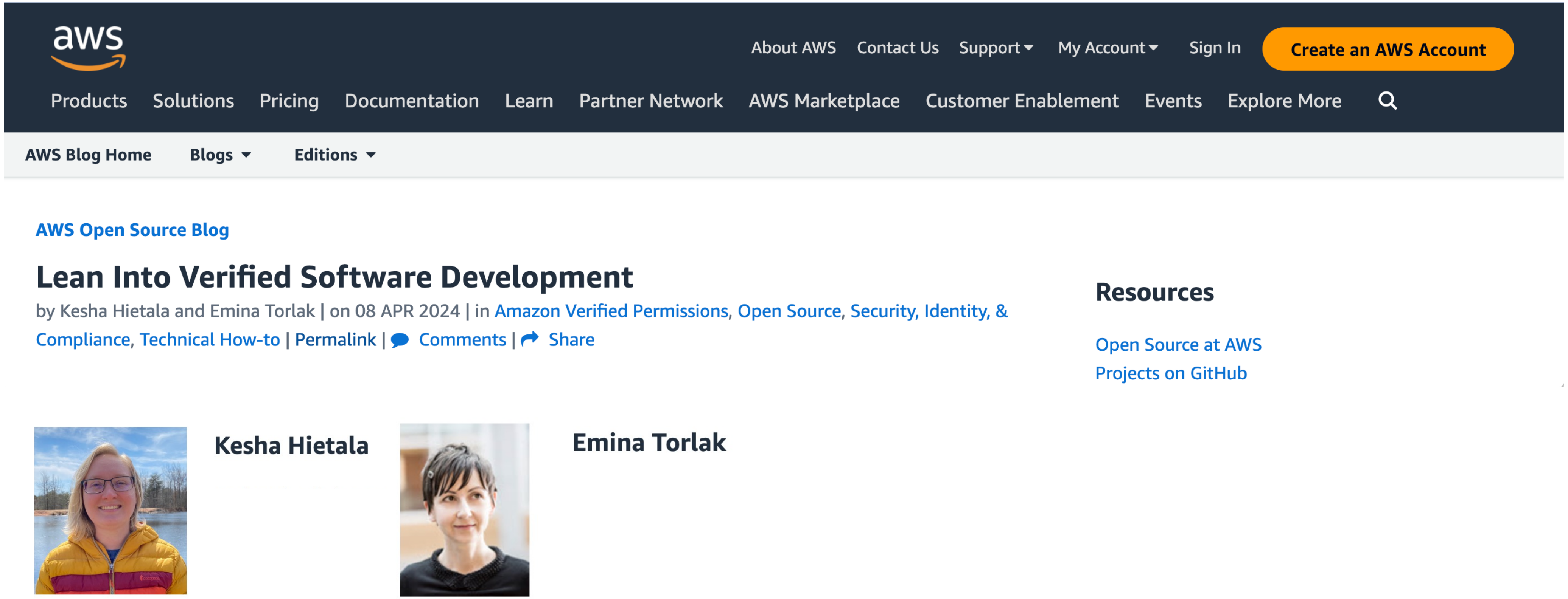
Cedar



Takeaway: “We’ve found Lean to be a great tool for verified software development. You get a full-featured programming language, fast proof checker and runtime, and a familiar way to build both models and proofs”

Cedar

To learn more about Cedar:
https://aws.amazon.com/blogs/opensource/lean-into-verified-software-development/



SampCert

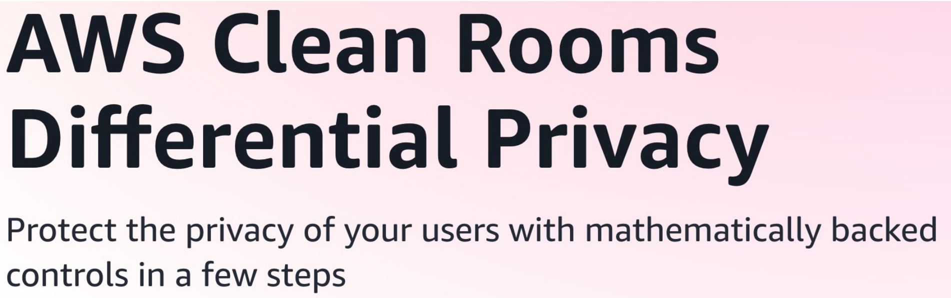
SampCert is an open-source library of formally verified **differential-privacy primitives** used by the AWS Clean Rooms Differential Privacy service for its fast and sound sampling algorithms

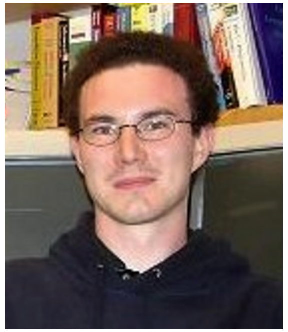
SampCert provides the only verified implementation of the discrete Gaussian sampler and the primitives of zero concentrated differential privacy

2x faster than the unverified previous implementation

The verification of code addressing practical problems in data privacy depends on the formalization of mathematical concepts from **Fourier analysis** to **number theory** and **topology**.

Led by Jean-Baptiste Tristan





LNSym

LNSym is a symbolic simulator for Armv8 machine-code programs

Led by Shilpi Goel

Open-source and under active development: https://github.com/leanprover/LNSym

Focus: **automated reasoning of cryptographic machine-code programs**

It uses Lean as

- A specification language to model the Arm instruction semantics and cryptographic protocols
- A theorem prover for reasoning

Takeaways

- Lean programs are executable and efficient: conformance testing
- Automation: SAT, verified bit-blaster, and domain specific tactics implemented in Lean
- Interactive prover when automation fails

AILean – AI for Math and Math for AI

AILean is exploring the relationship between LLMs and formal mathematics in collaboration with the [Technology Innovation Institute (TII)].

Led by Soonho Kong

AI for Math

- LLMs for enhancing proof automation
- User assistance

Math for AI

- Extracting training data from Lean proofs
- Correct by construction synthetic datasets

To learn more about Lean

Check out our website: https://lean-lang.org/

Follow Lean announcements on Twitter and Mastodon

Try out Lean online: https://live.lean-lang.org/

Check out the community website: https://leanprover-community.github.io/

Courses: https://leanprover-community.github.io/teaching/courses.html

Engage with the Lean community on the Lean Zulip Channel.

Conclusion

- Lean is an **efficient programming language** and **proof assistant**
- Machine checkable proofs **eliminate the trust bottleneck**
- Lean enables **decentralized collaboration**
- Lean is very **extensible**:
    - Users extend Lean using Lean itself without fear of introducing unsoundness
- The Mathlib community is changing how math is done
- Lean **proofs are maintainable, stable, and transparent**
- The **FRO model** has been instrumental in supporting Lean

It is not just about proving but also understanding complex objects and proofs, getting new insights, and navigating through the “thick jungles” that are beyond our cognitive abilities.