

Verified Collaboration: How Lean is Transforming Mathematics, Programming, and AI

Leo de Moura
Senior Principal Applied Scientist, AWS
Chief Architect, Lean FRO

March 12, 2025

Breaking the Cycle of Uncertainty: Math, Software, and AI You Can Trust

Math, software, and AI often rely on **manual review** or **partial testing**.

An error in a theorem or critical software system can have massive consequences.

Progress dies where fear of mistakes lives.

Breaking the Cycle of Uncertainty: Math, Software, and AI You Can Trust

Math, software, and AI often rely on **manual review or partial testing**.

An error in a theorem or critical software system can have massive consequences.

Progress dies where fear of mistakes lives.

Lean: **machine-checkable proofs eliminate guesswork and create trust.**

If every step is formally verified, we unlock unprecedented confidence and collaboration.



Lean is an open-source programming language and proof assistant that is transforming how we approach mathematics, software verification, and AI.

The Lean project, started in 2013, aimed at merging interactive and automated theorem proving.

Lean provides **machine-checkable proofs**.

Lean addresses the “trust bottleneck”.

Lean opens up new possibilities for collaboration.

A small example

The screenshot shows the Lean 4 IDE interface. On the left, there is a code editor window titled "Odd.lean U" containing the following Lean code:

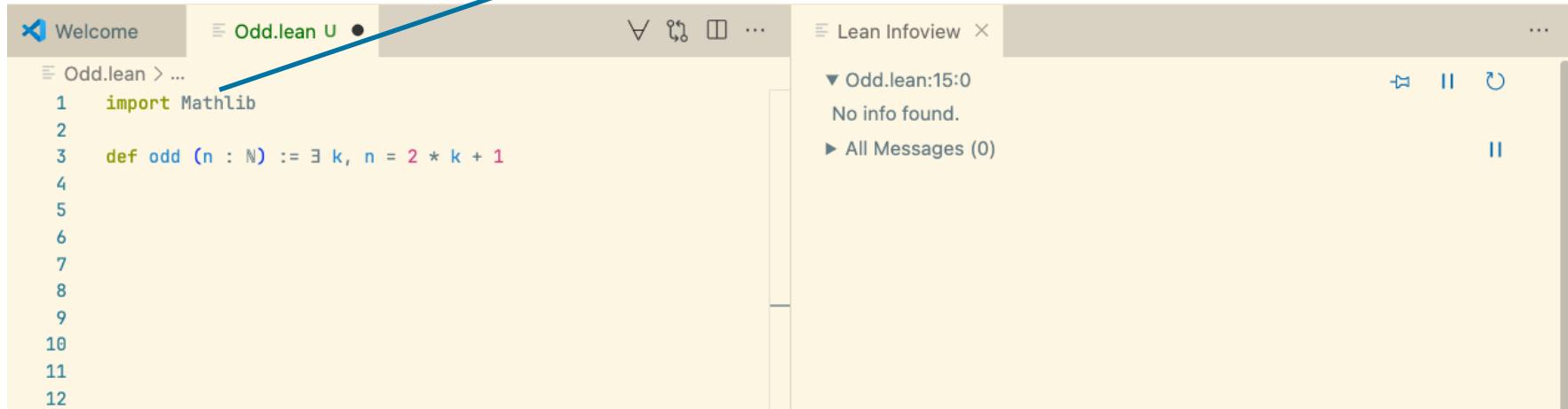
```
1 import Mathlib
2
3 def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5
6
7
8
9
10
11
12
```

On the right, there is an "Infoview" panel titled "Lean Infoview". It displays the following information:

- Odd.lean:15:0
- No info found.
- All Messages (0)

A small example

Mathlib is the Lean Mathematical library



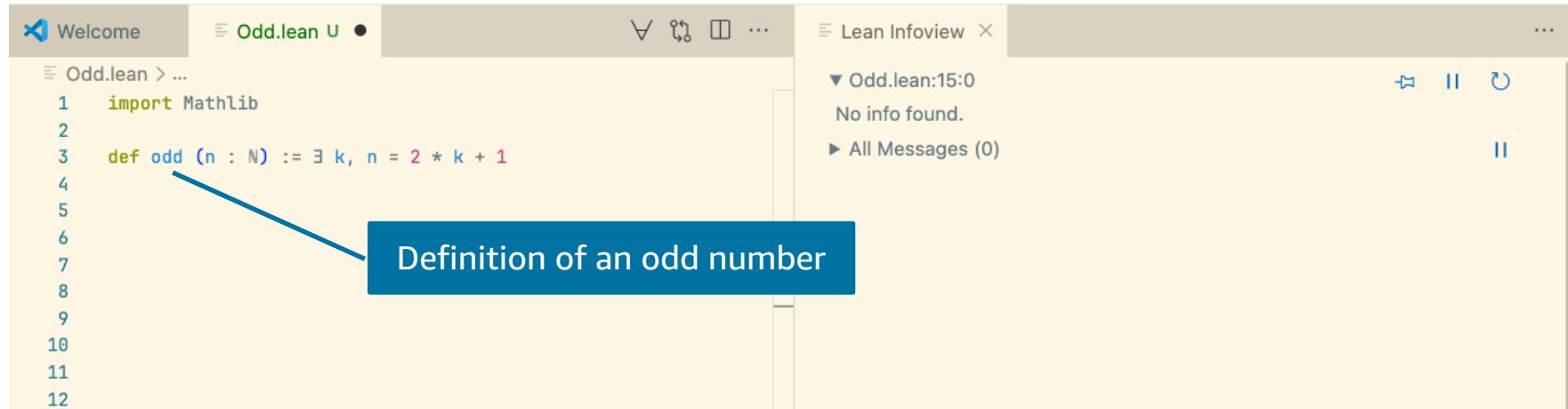
The screenshot shows the Lean code editor interface. On the left, there is a code editor window titled "Odd.lean > ...". The code inside is:

```
1 import Mathlib
2
3 def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5
6
7
8
9
10
11
12
```

On the right, there is a "Lean Infoview" window. It shows the following information:

- Odd.lean:15:0
- No info found.
- All Messages (0)

A small example



The screenshot shows the Lean IDE interface with two tabs open: "Odd.lean" and "Lean Infoview".

The "Odd.lean" tab contains the following code:

```
1 import Mathlib
2
3 def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5
6
7
8
9
10
11
12
```

A blue arrow points from the text "Definition of an odd number" to the line "def odd (n : ℕ) := ∃ k, n = 2 * k + 1".

The "Lean Infoview" tab shows:

- Odd.lean:15:0
- No info found.
- All Messages (0)

Definition of an odd number

Our first theorem

The screenshot shows the Lean 4 IDE interface. On the left, the code editor displays `Odd.lean` with the following content:

```
1 import Mathlib
2
3 def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5 theorem five_is_odd : odd 5 := by
6   use 2
7   done
```

A blue arrow points from the word `five_is_odd` in the code editor to a callout box labeled "Theorem statement, i.e., the claim being made".

The right side of the interface shows the `Lean Infoview` window, which contains the following information:

- Odd.lean:15:0
- Tactic state
- No goals**
- All Messages (0)

At the top of the interface, there are tabs for `Welcome`, `Odd.lean U`, and `Lean Infoview`. The `Lean Infoview` tab is currently active.

Theorem statement, i.e., the claim being made

Our first theorem

The screenshot shows the Lean 4 IDE interface with two main panes. The left pane displays the code file `Odd.lean`:

```
1 import Mathlib
2
3 def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5 theorem five_is_odd : odd 5 := by
6   use 2
7   done
```

A blue arrow points from the word "done" in the code to a teal button labeled "A proof". The right pane is titled "Lean Infoview" and shows the following state:

- Odd.lean:15:0
- Tactic state
- No goals**
- All Messages (0)

There are also some navigation icons at the top right of the infoview pane.

Our first theorem

The screenshot shows the Lean 4 IDE interface with two main panes. The left pane is the code editor for `Odd.lean`, and the right pane is the `Lean Infoview`.

Code Editor (Odd.lean):

```
>Welcome
Odd.lean 1, U ...
Odd.lean > five_is_odd
1 import Mathlib
2
3 def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5 theorem five_is_odd : odd 5 := by
6   use 3
7   done
```

A blue arrow points from the word "done" in the code editor to a blue box containing the text "An incorrect proof".

Lean Infoview:

- Odd.lean:7:2
- Tactic state
- 1 goal**
 - case h
 - $\vdash 5 = 2 * 3 + 1$
- Messages (1)
- All Messages (1)

An incorrect proof

Theorem proving in Lean is an interactive game

The screenshot shows the Lean IDE interface. On the left, the code editor displays `Odd.lean` with the following content:

```
Odd.lean > square_of_odd_is_odd
1 import Mathlib
2
3 def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5 -- Prove that the square of an odd number is always odd
6 theorem square_of_odd_is_odd : odd n → odd (n * n) := by
7   done
8
9
10
11
12
```

On the right, the Lean Infoview window shows the tactic state:

- Goal: $n : \mathbb{N}$
 $\vdash \text{odd } n \rightarrow \text{odd } (n * n)$
- Tactic state
- 1 goal
- Messages (1)
- All Messages (2)

A blue arrow points from the "All Messages (2)" link to a callout box containing the text "The ‘game board’".

"You have written my favorite computer game", Kevin Buzzard

Theorem proving in Lean is an interactive game

The screenshot shows the Lean 4 code editor interface. On the left, the code file `Odd.lean` contains the following code:

```
1 import Mathlib
2
3 def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5 -- Prove that the square of an odd number is always odd
6 theorem square_of_odd_is_odd : odd n → odd (n * n) := by
7   intro ⟨k₁, e₁⟩
8   done
```

A blue arrow points from the word "done" in the code to a callout box at the bottom containing the text "A ‘game move’, aka ‘tactic’".

On the right, the `Lean Infoview` window displays the current tactic state:

- Lean.infoview** tab is selected.
- Tactic state**:
 - 1 goal**
 - $n \ k_1 : \mathbb{N}$
 - $e_1 : n = 2 * k_1 + 1$
 - $\vdash \text{odd } (n * n)$
- Messages**: 1 message
- All Messages**: 2 messages

Theorem proving in Lean is an interactive game

The screenshot shows the Lean 4 code editor interface. On the left, there's a code editor window titled "Odd.lean 2, U" containing the following Lean code:

```
1 import Mathlib
2
3 def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5 -- Prove that the square of an odd number is always odd
6 theorem square_of_odd_is_odd : odd n → odd (n * n) := by
7   intro ⟨k₁, e₁⟩
8   simp [e₁, odd]
9   done
```

The "done" command is highlighted with a red underline. A blue arrow points from the word "simp" in the code to the "simp" tactic in the "Tactic state" pane on the right.

The "Lean Infoview" pane on the right displays the current tactic state:

- Goal: $n \ k_1 : \mathbb{N}$
- Hypothesis: $e_1 : n = 2 * k_1 + 1$
- Target: $\exists k, (2 * k_1 + 1) * (2 * k_1 + 1) = 2 * k + 1$

Below the goal, there are sections for "Messages (1)" and "All Messages (2)".

The “game move” `simp`, the simplifier, is one of the most popular moves in our game

Theorem proving in Lean is an interactive game

The screenshot shows the Lean 4 code editor interface. On the left, the code file `Odd.lean` contains a definition of an odd number and a theorem to prove its square is odd. The proof uses existential quantification (`exists`) and the tactic `use` to handle the existential part. The right side shows the Lean Infoview, which displays the current tactic state, including the goal $(2 * k_1 + 1) * (2 * k_1 + 1) = 2 * (2 * k_1 * k_1 + 2 * k_1) + 1$. A blue arrow points from the word `use` in the code to the `case h` section in the Infoview, highlighting the connection between the two.

```
Odd.lean > square_of_odd_is_odd
1 import Mathlib
2
3 def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5 -- Prove that the square of an odd number is always odd
6 theorem square_of_odd_is_odd : odd n → odd (n * n) := by
7   intro ⟨k₁, e₁⟩
8   simp [e₁, odd]
9   use 2 * k₁ * k₁ + 2 * k₁
10  done
```

Lean Infoview

- Odd.lean:10:1
- Tactic state
- 1 goal
 - case h
 - n k₁ : ℕ
 - e₁ : n = 2 * k₁ + 1
 - $\vdash (2 * k_1 + 1) * (2 * k_1 + 1) = 2 * (2 * k_1 * k_1 + 2 * k_1) + 1$
- Messages (1)
- All Messages (2)

The “game move” `use` is the standard way of proving statements about existentials

Theorem proving in Lean is an interactive game

The screenshot shows the Lean 4 IDE interface. On the left, the code editor displays a file named `Odd.lean` with the following content:

```
1 import Mathlib
2
3 def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5 -- Prove that the square of an odd number is always odd
6 theorem square_of_odd_is_odd : odd n → odd (n * n) := by
7   intro ⟨k₁, e₁⟩
8   simp [e₁, odd]
9   use 2 * k₁ * k₁ + 2 * k₁
10  linarith
11  done
```

The code editor has line numbers from 1 to 12. A blue arrow points from the word "linarith" in line 10 towards a callout box at the bottom.

On the right, the `Lean Infoview` window shows the following state:

- Odd.lean:17:0
- Tactic state: **No goals**
- All Messages (1)

We complete this level using `linarith`, the linear arithmetic, move

Theorem proving in Lean is an interactive and addictive game

The screenshot shows the Lean 4 IDE interface. On the left, the code file `Odd.lean` is open, displaying the following code:

```
import Mathlib

def odd (n : ℕ) := ∃ k, n = 2 * k + 1

-- Prove that the square of an odd number is always odd
theorem square_of_odd_is_odd : odd n → odd (n * n) := by
  intro ⟨k₁, e₁⟩
  simp [e₁, odd]
  use 2 * k₁ * k₁ + 2 * k₁
  linarith
  done
```

On the right, the `Lean Infoview` window is visible, showing the tactic state and message history. The message history pane displays:

- Odd.lean:17:0
- Tactic state
- No goals**
- All Messages (1)

"You can do 14 hours a day in it and not get tired and feel kind of high the whole day.

You're constantly getting positive reinforcement", Amelia Livingston

Mathlib

The Lean Mathematical Library supports a wide range of projects.

It is an open-source **collaborative project** with over 500 contributors and 1.7M LoC.

"I'm investing time now so that somebody in the future can have that amazing experience",

Heather Macbeth



FOUNDATIONS OF MATHEMATICS

Building the Mathematical Library of the Future



Mathematics

Software

AI

Mathematics

Preamble: the Perfectoid Spaces Project

Kevin Buzzard, Patrick Massot, Johan Commelin

Goal: Demonstrate that we can **define complex mathematical objects** in Lean.

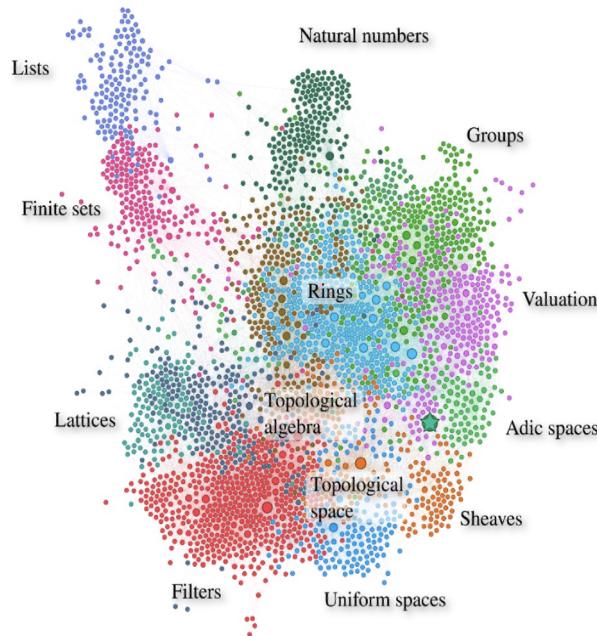
They translated Peter Scholze's definition into a form a computer can understand.

It not only achieved its goals but also demonstrated to the math community that
formal objects can be visualized and inspected with computer assistance.

Math is now **data** that can be **processed, transformed**, and **inspected** in various ways.

Preamble: the Perfectoid Spaces Project (cont.)

Kevin Buzzard, Patrick Massot, Johan Commelin



mathoverflow

Home What are "perfectoid spaces"?

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

72 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)
  (uniform : is_uniform R)
  (ramified : ∃ w : pseudo_uniformizer R, w^p ⊥ p in R°)
  (Frobenius : surjective (Frob R°/p))
```

Mathlib > RingTheory > Finiteness.lean

```

355 theorem FG.stabilizes_of_iSup_eq {M' : Submodule R M} (hM' : M'.FG) (N : N →o Submodule R M)
356   (H : iSup N = M') : ∃ n, M' = N n := by
357   obtain ⟨S, hs⟩ := hM'
358   have : ∀ s : S, ∃ n, (s : M) ∈ N n := fun s =>
359     (Submodule.mem_iSup_of_chain N s).mp
360     (by
361       rw [H, ← hs]
362       exact Submodule.subset_span s.2)
363   choose f hf using this
364   use S.attach.sup f
365   apply le_antisymm
366   · conv_lhs => rw [← hs]
367     rw [Submodule.span_le]
368   intro s hs
369   exact N.2 (Finset.le_sup <| S.mem_attach ⟨s, hs⟩) (hf _)
370   · rw [← H]
371     exact le_iSup _ _

```

▼ Finiteness.lean:365:2

▼ Tactic state

1 goal▼ **case** intro

R : Type u_1

M : Type u_2

inst^{t²} : Semiring Rinst^{t¹} : AddCommMonoid Minst^t : Module R M

M' : Submodule R M

N : N →o Submodule R M

H : iSup ↑N = M'

S : Finset M

hs : span R ↑S = M'

f : {x // x ∈ S} → N

hf : ∀ (s : {x // x ∈ S}), ↑s ∈ N (f s)

↑S ⊢ ∃ n, M' = N n

Mathlib > RingTheory > Finiteness.lean

```
555
356 theorem FG.stabilizes_of_iSup_eq {M' : Submodule R M} (hM' : M'.FG) (N : N →o Submodule R M)
357   (H : iSup N = M') : ∃ n, M' = N n := by
358   obtain ⟨S, hs⟩ := hM'
359   have : ∀ s : S, ∃ n, (s : M) ∈ N n := fun s =>
360     (Submodule.mem_iSup_of_chain N s).mp
361     (by
362       rw [H, ← hs]
363       exact Submodule.subset_span s.2)
364   choose f hf using this
365   use S.attach.sup f
366   apply le_antisymm
367   · conv_lhs => rw [← hs]
368     rw [Submodule.span_le]
369     intro s hs
370     exact N.2 (Finset.le_sup <| S.mem_attach ⟨s, hs⟩) (hf _)
371   · rw [← H]
372     exact le_iSup _ _
```

▼ Finiteness.lean:365:2

▼ Tactic state

1 goal

▼ case intro

R : Type u_1
M : Type u_2
inst^{t2} : Semiring R
inst^{t1} : AddCommMonoid M
inst^t : Module R M
M' : Submodule R M
N : N →o Submodule R M
H : iSup ↑N = M'
S : Finset M
hs : span R ↑S = M'
f : {x // x ∈ S} → N
hf : ∀ (s : {x // x ∈ S}), ↑s ∈ N (f s)
⊢ ∃ n, M' = M' : Submodule R M

Mathlib > RingTheory > Finiteness.lean

```
355 theorem FG.stabilizes_of_iSup_eq {M' : Submodule R M} (hM' : M'.FG) (N : N → Submodule R M)
```

Defs.lean ~/projects/mathlib4/Mathlib/Algebra/Module/Submodule - Definitions (1)

```
25 assert_not_exists DivisionRing
26
27 open Function
28
29 universe u'' u' u v w
30
31 variable {G : Type u''} {S : Type u'} {R : Type u} {M : Type v} {ι :
32
33 /-- A submodule of a module is one which is closed under vector oper-
34   This is a sufficient condition for the subset of vectors in the su-
35   to themselves form a module. -/
36 structure Submodule (R : Type u) (M : Type v) [Semiring R] [AddCommM-
37   AddSubmonoid M, SubMulAction R M : Type v
```

structure Submodule (R : Type u) (M : Type v) [Semiring R] [AddCommM-

▼ Finiteness.lean:356:44

▼ Expected type

R : Type u_1
M : Type u_2
instt⁴ : Semiring R
instt³ : AddCommMonoid M
instt² : Module R M
P : Type u_3
instt¹ : AddCommMonoid P
instt : Module R P
f : M →[R] P
↳ Type u_2

► All Messages (0)

Mathlib > Algebra > Module > Submodule > `Defs.lean` > Submodule

```
34  This is a sufficient condition for the subset of vectors in the submodule
35  to themselves form a module. -/
36 structure Submodule (R : Type u) (M : Type v) [Semiring R] [AddCommMonoid M] [Module R M] extends
37  AddSubmonoid M, SubMulAction R M : Type v
```

Defs.lean ~/projects/mathlib4/Mathlib/Algebra/Submonoid - Definitions (1)

```
84 add_decl_doc Submonoid.toSubsemigroup
85
86 /-- `SubmonoidClass S M` says `S` is a type of subsets `s ⊆ M` that
87 and are closed under `(*)` -/
88 class SubmonoidClass (S : Type*) (M : outParam Type*) [MulOneClass M]
89   MulMemClass S M, OneMemClass S M : Prop
90
91 section
92
93 /-- An additive submonoid of an additive monoid `M` is a subset containing
94 zero that is closed under addition. -/
95 structure AddSubmonoid (M : Type*) [AddZeroClass M] extends AddSubsemigroup M
96 /-- An additive submonoid contains `0`. -/
97 zero_mem' : (0 : M) ∈ carrier
98
```

▼ `Defs.lean:37:8`

▼ Expected type

`G` : Type `u''`
`S` : Type `u'`
`Rt` : Type `u`
`Mt` : Type `v`
`r` : Type `w`
`R` : Type `u`
`M` : Type `v`
`inst2` : Semiring `R`
`inst1` : AddCommMonoid `M`
`instt` : Module `R M`
↳ Type `v`

► All Messages (0)

The Challenge

In November of 2020, Peter Scholze posits the Liquid Tensor Experiment (LTE) 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",*

Peter Scholze

The First Victory

Johan Commelin led a team with several members of the **Lean community and announced the formalization of the crucial intermediate lemma** that Scholze was unsure about, with only minor corrections, in **May 2021**.

"[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", Peter Scholze

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

Achieving the Unthinkable

The full challenge was completed in July 2022.

The team not only verified the proof but also simplified it.

Moreover, they did this without fully understanding the entire proof.

Johan, the project lead, reported that he could only see two steps ahead. **Lean was a guide.**

"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

Only the Beginning

Independence of the Continuum Hypothesis, Han and van Doorn, 2021

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

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

The Equational Theories Project, Tao, 2024

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

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 C/C++, Rust, Lean, etc.

You can implement your own checker.

What did we learn?

Machine-checkable proofs enable a new level of **collaboration** in mathematics.

The power of the **community**.

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**.

What did we learn?

Another unexpected benefit of formal mathematics: **auto refactoring** and **generalization**.

general An example of why formalization is useful Mar 31

Riccardo Brasca EDITED 7:53 AM

I really like what is going on with #12777. @Sebastian Monnet proved that if `E`, `F` and `K` are fields such that `finite_dimensional F E`, then `fintype (E →ₐ [F] K)`. We already have `docs#field.alg_hom.fintype`, that is exactly the same statement with the additional assumption `is_separable F E`.

The interesting part of the PR is that, with the new theorem, the linter will automatically flag all the theorem that can be generalized (for free!), removing the separability assumption. I think in normal math this is very difficult to achieve, if I generalize a 50 years old paper that assumes `p ≠ 2` to all primes, there is no way I can manually check and maybe generalize all the papers that use the old one.

3 5

Software

Lean in Software Verification: The Story of SampCert

Lean is a programming language, and is used in **many software verification projects**.

You can write code and reason about it simultaneously.

You can prove that your code has the properties you expect.

"Testing can show the presence of bugs, but not their absence", E. Dijkstra

Differential Privacy

A mathematical framework that ensures the **privacy of individuals** in a dataset by adding controlled **random noise** to the data.

Discrete sampling algorithms, like the **Discrete Gaussian Sampler**, are used to add carefully calibrated noise to data.

What may go wrong if a buggy sampler is used?

Privacy Violations: leakage of sensitive information

Incorrect Results: distorted analysis results

SampCert

A project led by **Jean-Baptiste Tristan** at AWS.

An **open-source** Lean library of formally **verified differential privacy primitives**.

Tristan's implementation is not only verified, but it is also **twice as fast as the previous one**.

He managed to implement **aggressive optimizations** because Lean served as a guide, ensuring that **no bugs** were introduced.

AWS Clean Rooms Differential Privacy

Protect the privacy of your users with mathematically backed controls in a few steps

SampCert would not exist without Mathlib

SampCert is software, but its verification relies heavily on Mathlib.

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**.

What did we learn?

Machine-checkable proofs enable you to **code without fear**.

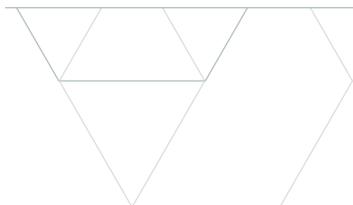
Industrial projects: Verified compilers, policy languages, cryptographic libraries, etc.

Many more at the **Lean Project Registry**: <https://reservoir.lean-lang.org/>



Research areas ▾ Blog Publications Conferences Code and datasets Academia ▾ Careers

AUTOMATED REASONING



How the Lean language brings math to coding and coding to math

AI

Lean Enables Verified AI for Mathematics and Code

LLMs are powerful tools, but they are prone to **hallucinations**.

In Math, a **small mistake can invalidate the whole proof**.

Imagine manually checking an AI-generated proof with the size and complexity of FLT.

The informal proof is **over 200 pages**.

Buzzard estimates a formal proof will require more than **1M LoC** on top of Mathlib.

Machine-checkable proofs are the antidote to hallucinations.

AI Proof Assistants

Several groups are developing AI that suggests the **next move**(s) in Lean's interactive proof game.

[LeanDojo](#) is an open-source project from Caltech, and everything (model, datasets, code) is open.

[OpenAI](#) and [Meta AI](#) have also developed AI assistants for Lean.

AI Proof Assistants

LeanCopilot is part of the LeanDojo project at Caltech. It uses the move (aka tactic) suggestion feature available in the Lean IDE.

The screenshot shows the Lean IDE interface with two main panes:

- Lean4Example.lean (Left Pane):** A code editor window titled "Lean4Example.lean 5, M". The code contains:

```
1 import LeanCopilot
2
3 example (a b c : Nat) : a * (b + c) = a * c + a * b := by
4   suggest_tactics
5   done
```

- Lean Infoview (Right Pane):** A pane titled "Lean Infoview" showing the current tactic state and suggestions.

Tactic state:

```
a b c : Nat
 $\vdash a * (b + c) = a * c + a * b$ 
```

Tactic suggestions:

- Try this:
- `simp [Nat.left_distrib, Nat.add_comm]`
- `rw [Nat.mul_add, Nat.add_comm]`
- `rw [Nat.mul_add, Nat.mul_comm, Nat.add_comm]`
- `simp [Nat.add_comm]`
- `rw [Nat.mul_comm, Nat.add_comm]`
- `rw [Nat.mul_comm, Nat.mul_comm]`
- `rw [Nat.mul_add, Nat.mul_comm]`
- `apply Nat.add_left_cancel`

Move Over, Mathematicians, Here Comes AlphaProof

A.I. is getting good at math — and might soon make a worthy collaborator for humans.

 Share full article    47



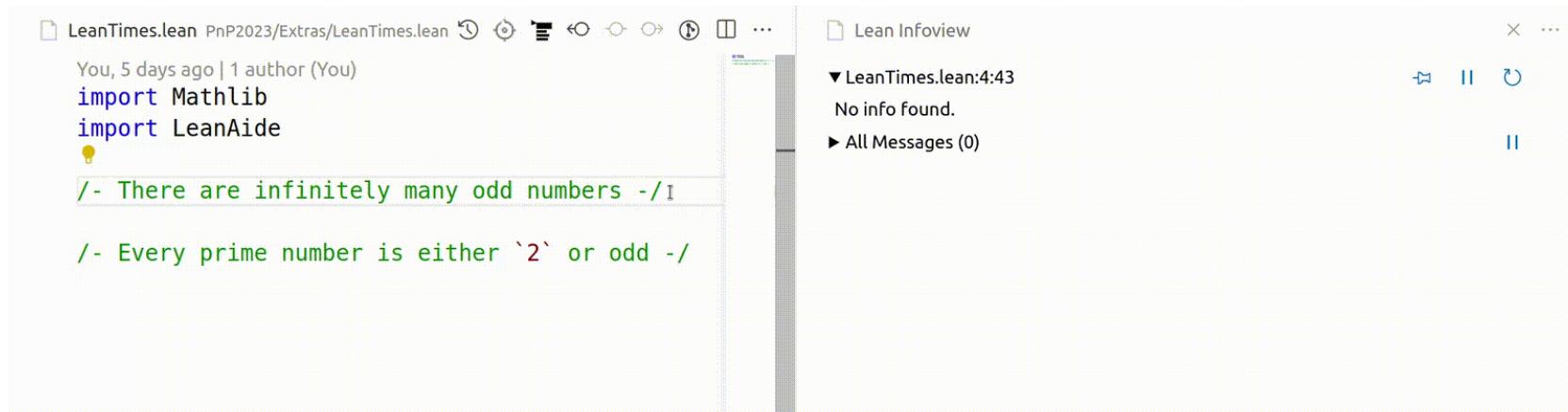
Ringing the gong at Google Deepmind's London headquarters, a ritual to celebrate each A.I. milestone, including its recent triumph of reasoning at the International Mathematical Olympiad. Google Deepmind

Auto-formalization

The process of converting natural language into a formal language like Lean.

It is much **easier to learn to read Lean than to write it.**

[LeanAide](#) is one of the auto-formalization tools available for Lean.



The screenshot shows the LeanAide interface. On the left, there is a code editor window titled "LeanTimes.lean" which contains the following Lean code:You, 5 days ago | 1 author (You)
import Mathlib
import LeanAide
/- There are infinitely many odd numbers -/
/- Every prime number is either `2` or odd -/Above the code editor, there are several small icons for file operations. To the right of the code editor is an "Infoview" panel titled "Lean Infoview". It displays the following information:

- LeanTimes.lean:4:43
- No info found.
- All Messages (0)

What did we learn?

Machine-checkable proofs enable **AI that does not hallucinate**.

LLMs enable **auto-formalization**.

LLMs are getting better and better at explaining Lean code.

In an era of big data and LLMs, machine-checkable proofs ensure trust in results.

AI systems that prove rather than guess.

Before we wrap up...

Lean Enables Decentralized Collaboration

Lean is Extensible

Users extend Lean using Lean itself.

Lean is implemented in Lean.

You can make it your own.

You can create your own moves.

Machine-Checkable Proofs

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

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

Code without fear.

Lean is a game where we can implement your own moves

The screenshot shows the Lean code editor interface. On the left, the code file `Odd.lean` contains a proof script. Lines 1-4 import `Mathlib` and define `odd` as $\exists k, n = 2 * k + 1$. Lines 5-12 prove that the square of an odd number is always odd, using `simp`, `use`, `linarith`, and `done`. The right side shows the `Lean Infoview` panel with sections for `Odd.lean:17:0`, `Tactic state`, and `No goals`. A message `All Messages (1)` is present. A blue arrow points from the `linarith` line in the code to the `LinArith` section in the infoview.

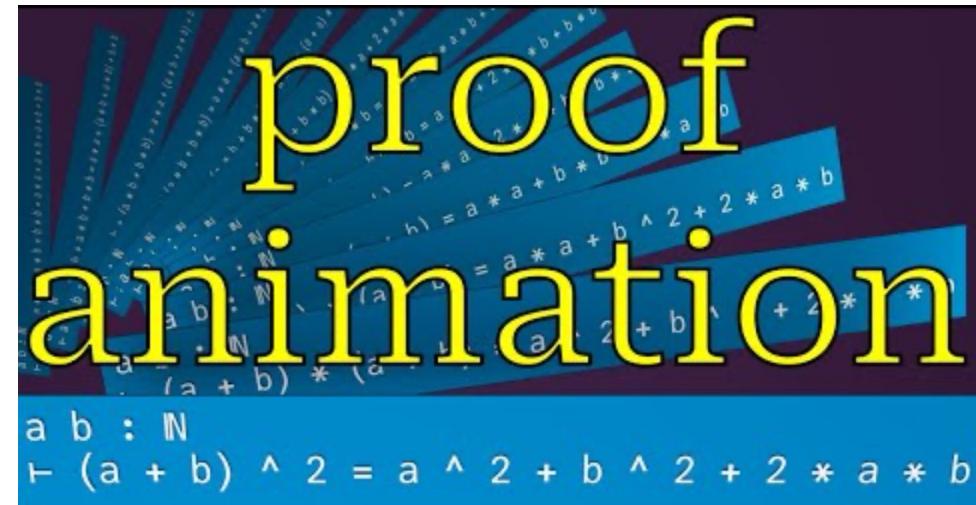
```
1 import Mathlib
2
3 def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5 -- Prove that the square of an odd number is always odd
6 theorem square_of_odd_is_odd : odd n → odd (n * n) := by
7   intro ⟨k₁, e₁⟩
8   simp [e₁, odd]
9   use 2 * k₁ * k₁ + 2 * k₁
10  linarith
11  done
```

The `linarith` “move” was implemented by the `Mathlib` community in Lean!

You can use Lean to introspect its internal data

The tool [lean-training-data](#) is implemented in Lean itself. **It is a Lean package.**

A similar approach can be used to automatically generate proof animations.



Lean FRO: Shaping the Future of Lean Development

The Lean Focused Research Organization (FRO) is a non-profit dedicated to Lean's development.

Founded in **August 2023**, the organization has 19 members.

Its mission is to enhance critical areas: **scalability, usability, documentation, and proof automation.**

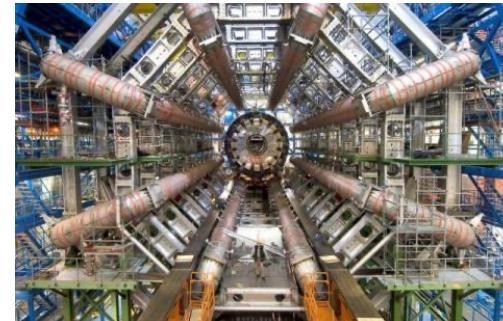
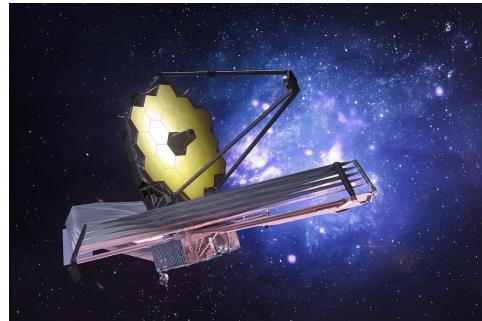
It must reach **self-sustainability in August 2028** and become the **Lean Foundation**.

Philanthropic support is gratefully acknowledged from the **Simons Foundation**, the **Alfred P. Sloan Foundation**, **Richard Merkin**, and **Founders Pledge**.

FROs accelerate scientific progress / Lean as a Catalyst

James Webb Telescope and CERN illustrate a common pattern in science: a need for projects that are bigger than an academic lab can undertake, more coordinated than a loose consortium or themed department, and not directly profitable enough to be a venture-backed startup or industrial R&D project.

<https://www.convergentresearch.org/about-fros>



Lean FRO: by numbers

16 releases and **3,433 pull requests** merged in the main repository only since its launch in July 2023.

[Latest v4.17.0](#): 168 new features, 57 bug fixes, 13 documentation improvements, and more.

Lean project was featured in multiple venues NY Times, Quanta, Scientific American, etc.

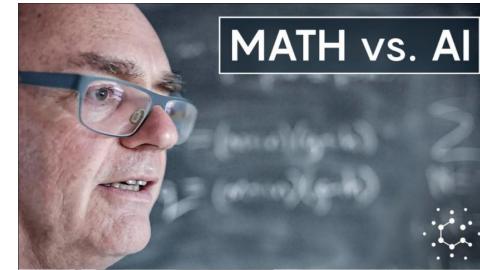


The New York Times

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?



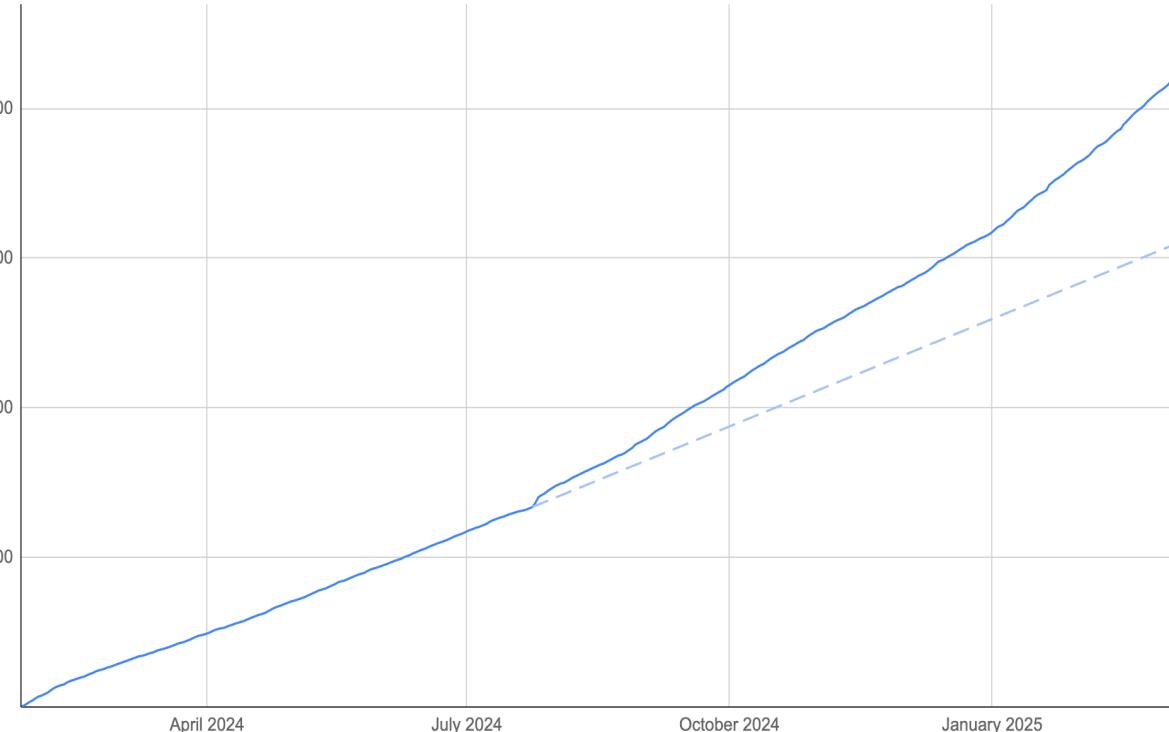
When Computers Write Proofs, What's the Point of Mathematicians?

[youtube.com](https://www.youtube.com)

Growth of Lean projects on GitHub



New installations of Lean Development Environment (2024 to present)





How can I contribute?

Help building [Mathlib](#).

Want to engage with the vibrant Lean community? Join our [Zulip channel](#).

Interested in ML kernels? Contribute to the [KLR project](#).

Want to contribute to a large formalization project? Join the [FLT formalization project](#).

Start your own open-source Lean project! Your package will be available on our registry [Reservoir](#).

Start using Lean online: [live.lean-lang.org](#)

Support the Lean FRO: Funding, partnerships, or simply advocating the project.

Conclusion

Lean is an **efficient programming language** and **proof assistant**.

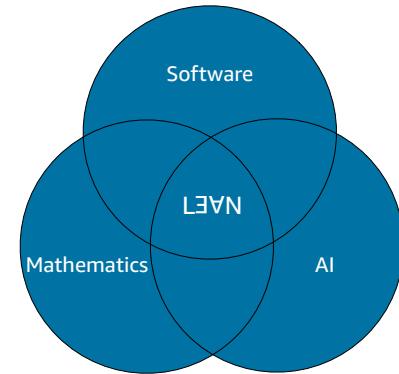
The Mathlib community is changing how math is done.

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**.

Lean tracks details, so humans focus on big ideas.

Decentralized collaboration with Lean: Large teams can collectively tackle huge proofs without losing track.

The entire discipline thrives when no one has to “take it on faith.”



Thank You

<https://leanprover.zulipchat.com/>
x: @leanprover
LinkedIn: Lean FRO
Mastodon: @leanprover@functional.cafe
#leanlang, #leanprover

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

