

# 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's role: **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.

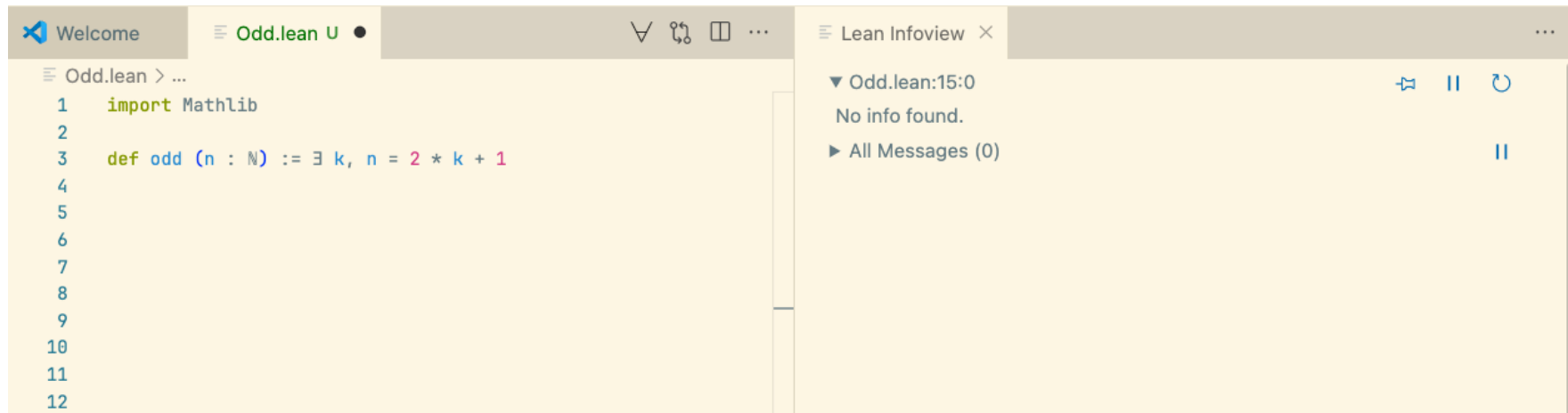
The Lean project, started in 2013, aimed at merging interactive and automated theorem proving.

Lean provides **machine-checkable proofs**.

Lean addresses the “trust bottleneck”.

**Lean opens up new possibilities for collaboration.**

## A small example



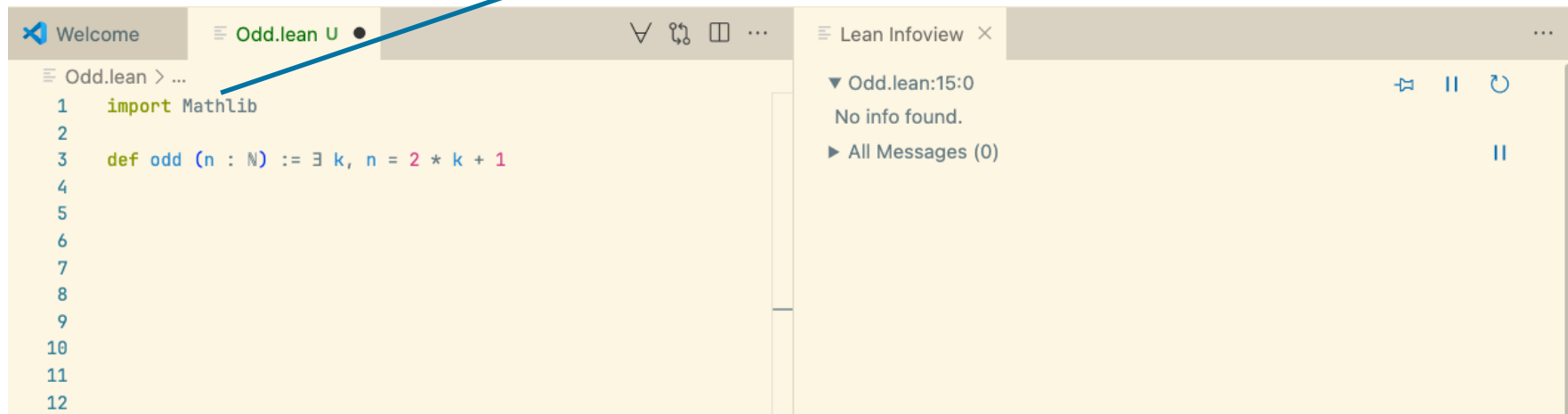
The screenshot shows the Lean IDE interface. The top bar contains a 'Welcome' button and a tab for 'Odd.lean U'. Below the top bar, the code editor displays the following Lean code:

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

To the right of the code editor is the 'Lean Infoview' panel. It shows a message from 'Odd.lean:15:0' with the text 'No info found.' and a button to view 'All Messages (0)'.

## A small example

Mathlib is the Lean Mathematical library

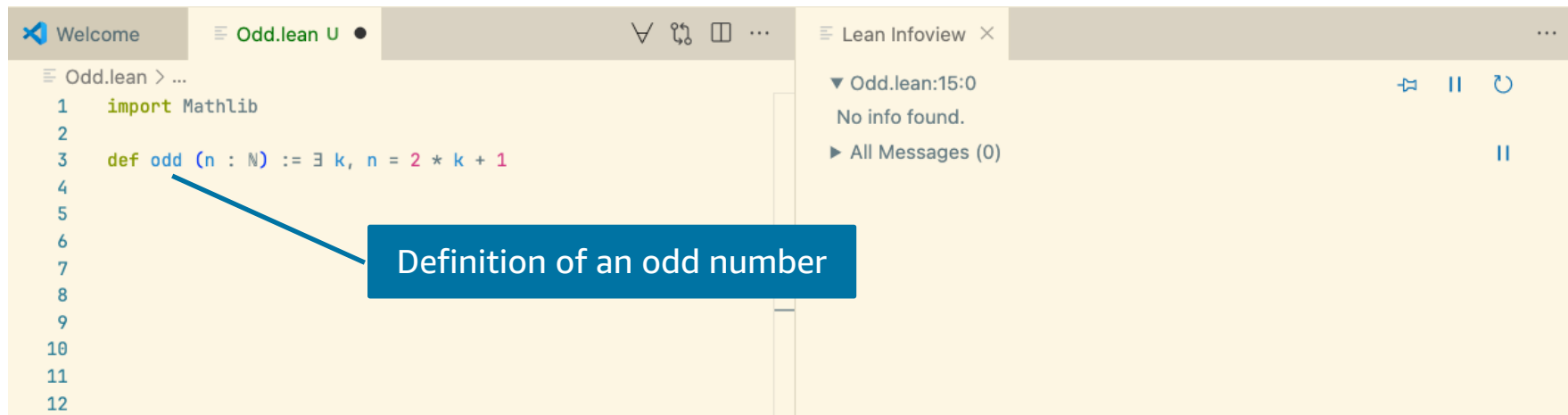


The screenshot shows the Lean IDE interface. The top bar includes a 'Welcome' button, a file explorer showing 'Odd.lean U', and a toolbar with icons for search, undo, redo, and other editor functions. The main editor area displays the following Lean code:

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

The right-hand side of the interface shows the 'Lean Infoview' panel. It displays the message 'No info found.' for the definition 'odd' at line 15:0. Below this, it indicates 'All Messages (0)'.

## A small example



The screenshot shows the Lean IDE interface. The main editor window displays the file `Odd.lean` with the following code:

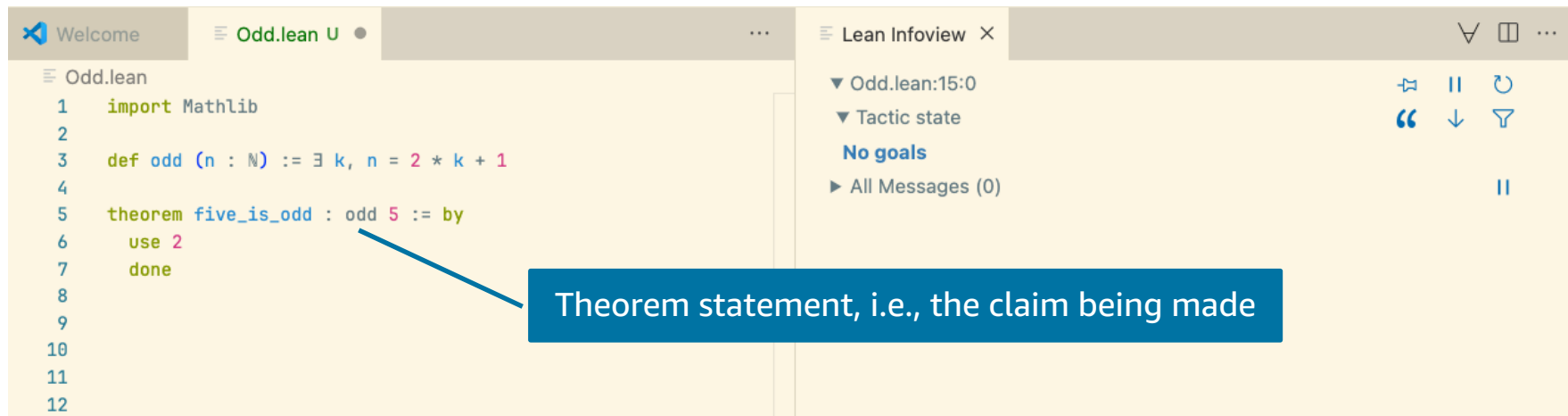
```
1 import Mathlib
2
3 def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5
6
7
8
9
10
11
12
```

A blue callout box with the text "Definition of an odd number" points to the definition of the `odd` function on line 3.

The right-hand pane shows the "Lean Infoview" for the definition. It displays the following information:

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

# Our first theorem



The screenshot shows the Lean IDE interface. The main editor displays the file `Odd.lean` with the following code:

```
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
8
9
10
11
12
```

A blue callout box points to the theorem statement `theorem five_is_odd : odd 5 := by` with the text: "Theorem statement, i.e., the claim being made".

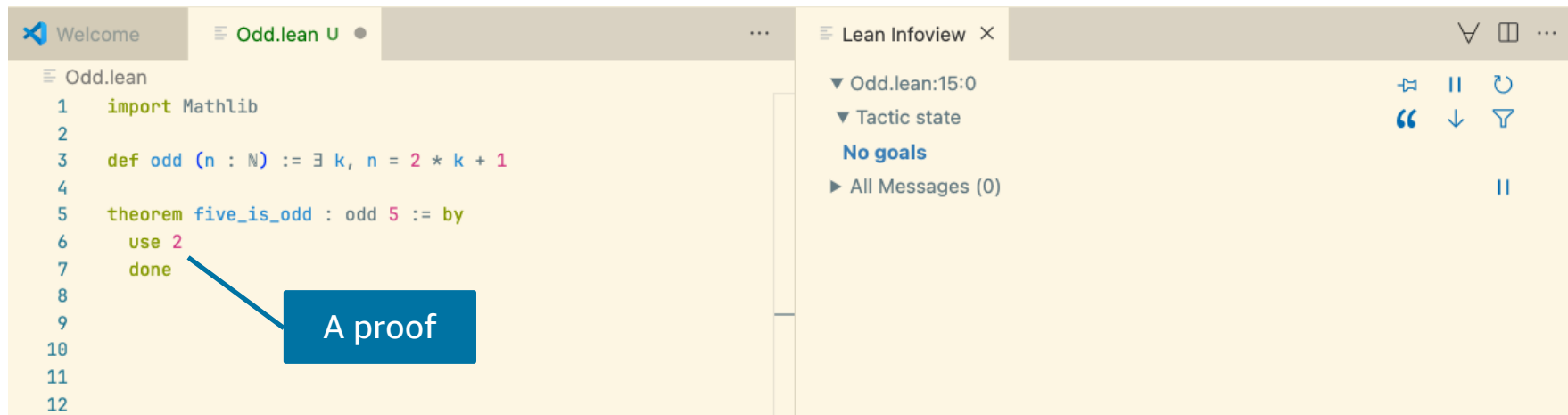
The right sidebar shows the `Lean Infoview` panel with the following content:

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

On the far right of the sidebar, there are icons for pinning, pausing, refreshing, quoting, scrolling, and filtering, along with a double vertical bar icon at the bottom.



# Our first theorem



The screenshot shows the Lean IDE interface. The main editor displays a file named `Odd.lean` with the following code:

```
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 callout box with the text "A proof" points to the `done` keyword on line 7, indicating that the proof is complete.

The right-hand pane shows the "Lean Infoview" for the current line. It displays the following information:

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

Navigation icons are visible on the right side of the Infoview pane.

# Our first theorem

The screenshot shows the Lean IDE interface. The left pane displays the source code for `Odd.lean`. The code defines an `odd` function and a theorem `five_is_odd` which is proved using the `use 3` tactic. The `done` keyword on line 7 is underlined in red, indicating an error. A blue callout box with the text "An incorrect proof" points to this `done` keyword. The right pane, titled "Lean Infoview", shows the state of the proof. It indicates the current goal is `5 = 2 * 3 + 1` under the `case h` hypothesis. The "Messages" section shows a single message, suggesting the proof attempt was not successful.

```
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
8
9
10
11
12
```

▼ Odd.lean:7:2  
▼ Tactic state  
1 goal  
▼ case h  
└ 5 = 2 \* 3 + 1  
► Messages (1)  
► All Messages (1)

An incorrect proof

# Theorem proving in Lean is an interactive game

The screenshot displays the Lean IDE interface. On the left, the source code for a theorem proof is shown:

```
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
```

On the right, the 'Lean Infoview' pane shows the current goal and tactic state:

```
▼ Odd.lean:7:2
▼ Tactic state
1 goal
  n : ℕ
  ⊢ odd n → odd (n * n)
► Messages (1)
► All Messages (2)
```

A blue box labeled "The 'game board'" points to the goal statement in the tactic state.

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

# Theorem proving in Lean is an interactive game

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   done
9
10
11
12
```

A "game move", aka "tactic"

Lean Infoview

Odd.lean:8:2

Tactic state

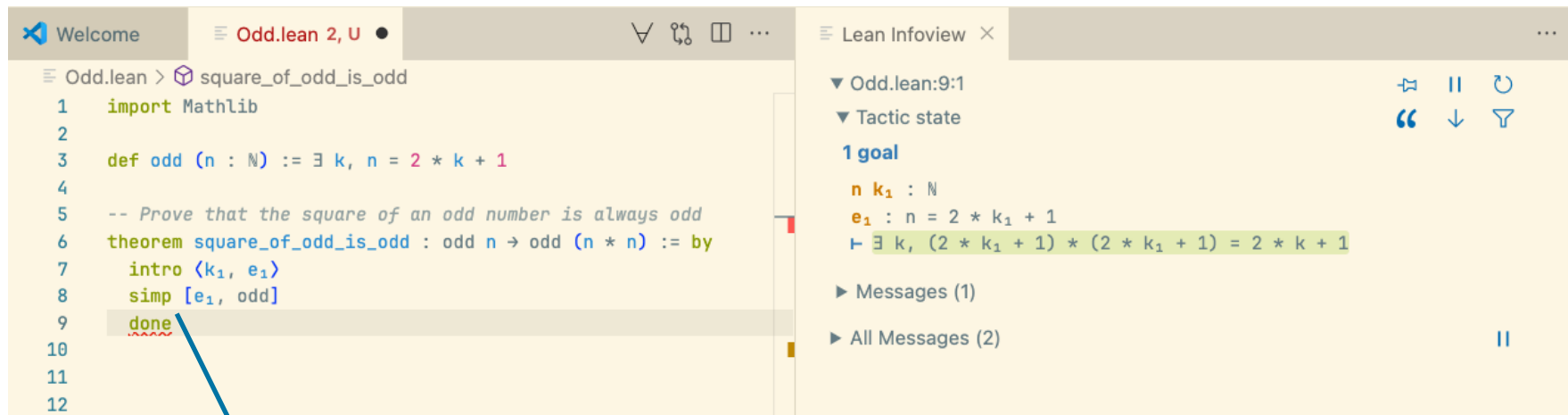
1 goal

$n \ k_1 : \mathbb{N}$   
 $e_1 : n = 2 * k_1 + 1$   
⊢ odd (n \* n)

Messages (1)

All Messages (2)

# Theorem proving in Lean is an interactive game



```
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    done
10
11
12
```

Lean Infoview

▼ Odd.lean:9:1

▼ Tactic state

1 goal

n k₁ : ℕ

e₁ : n = 2 \* k₁ + 1

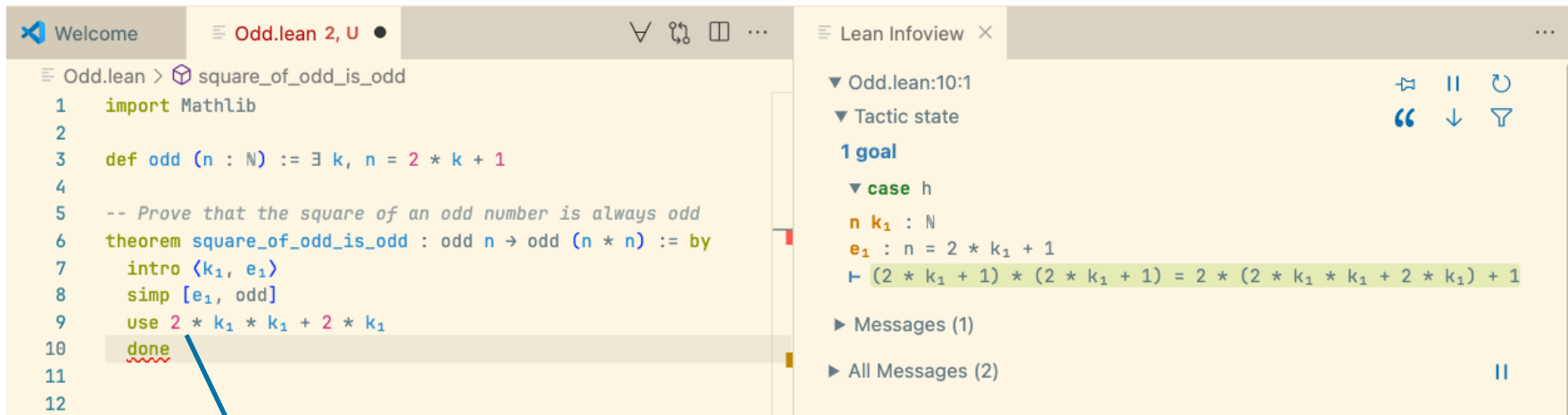
├ ∃ k, (2 \* k₁ + 1) \* (2 \* k₁ + 1) = 2 \* k + 1

► Messages (1)

► 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 IDE interface. On the left, the editor displays a Lean script for proving that the square of an odd number is odd. The script includes an import, a definition of 'odd', a theorem statement, and a proof using 'by' with several tactics. The 'done' tactic on line 10 is highlighted with a blue arrow pointing to a callout box. On the right, the 'Lean Infoview' panel shows the current goal and tactic state, including the goal, the 'case h' tactic, and the current hypotheses and goal expression.

```
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
11
12
```

Lean Infoview

▼ Odd.lean:10:1

▼ Tactic state

1 goal

▼ case h

n k₁ : ℕ

e₁ : n = 2 \* k₁ + 1

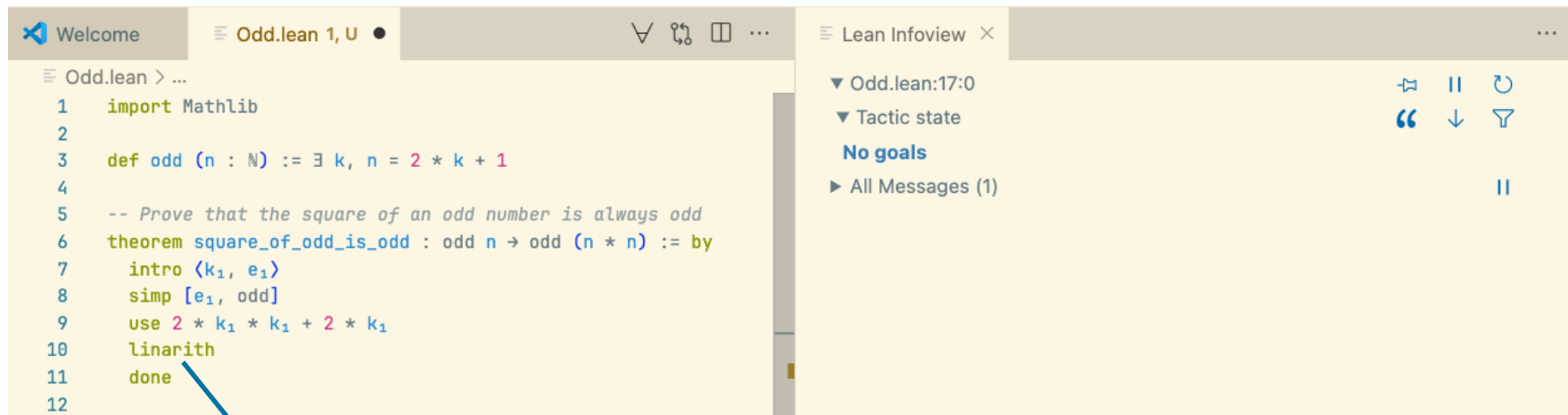
⊢ (2 \* k₁ + 1) \* (2 \* k₁ + 1) = 2 \* (2 \* k₁ \* k₁ + 2 \* k₁) + 1

► 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 IDE interface. The main editor displays a Lean script for proving that the square of an odd number is odd. The script is as follows:

```
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
12
```

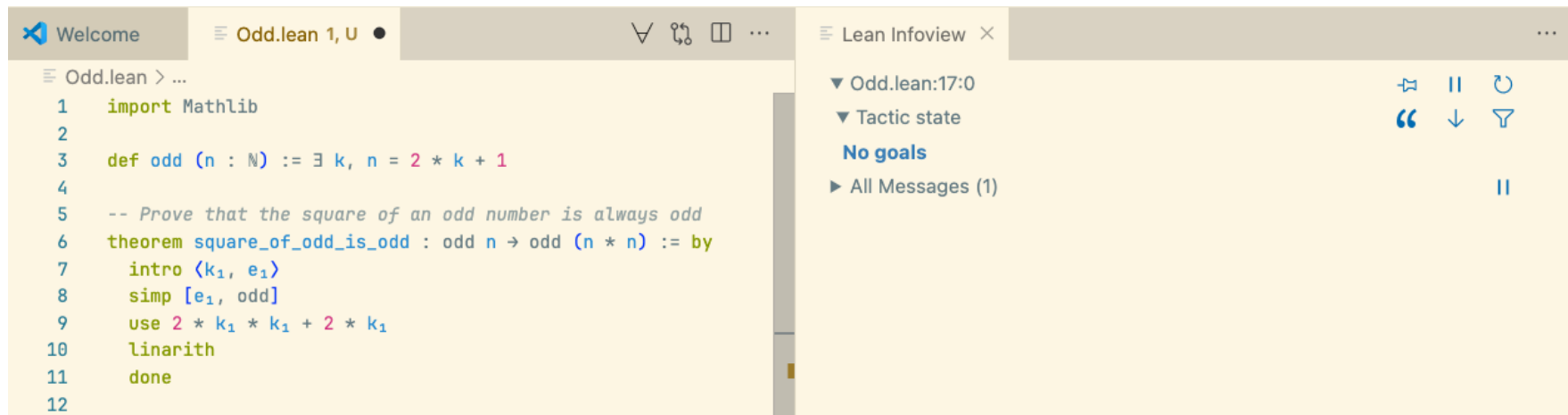
The right-hand pane shows the 'Lean Infoview' with the following content:

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

Navigation icons are visible on the right side of the Infoview pane.

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 IDE interface. The left pane displays the source code for a file named `Odd.lean`. The code defines an odd number and proves that its square is also odd. The right pane shows the Lean Infoview, which indicates that there are no goals left to prove.

```
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
12
```

Lean Infoview:

- 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



Quanta magazine

[Physics](#)

[Mathematics](#)

[Biology](#)

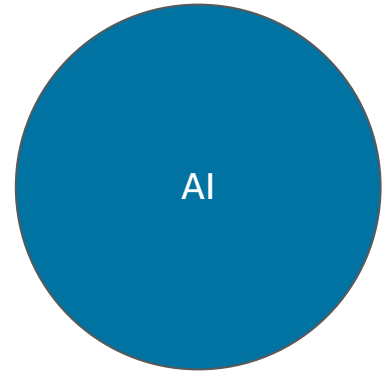
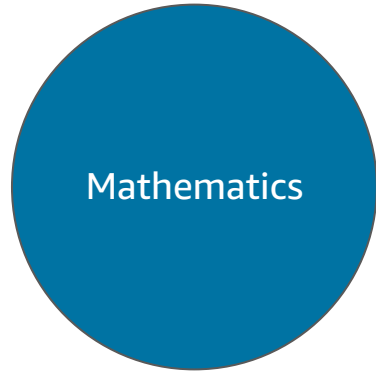
[Computer Science](#)

[Topics](#)

[Archive](#)

FOUNDATIONS OF MATHEMATICS

## Building the Mathematical Library of the Future



# Mathematics

# Preamble: the Perfectoid Spaces Project

*Kevin Buzzard, Patrick Massot, Johan Commelin*

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

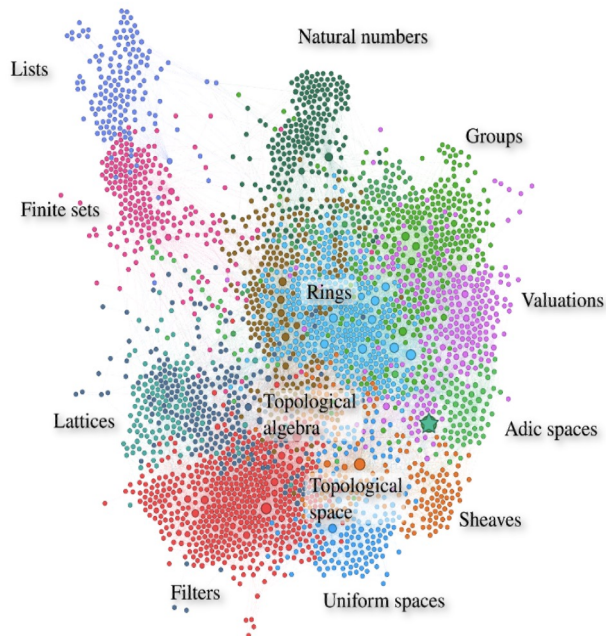
They translated Peter Scholze's definition into a form a computer can understand.

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 : ∃ ω : pseudo_uniformizer R, ω^p ∣ p in R)
  (Frobenius : surjective (Frob R/p))
```

## 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

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

## 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**.

# 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/>



# 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

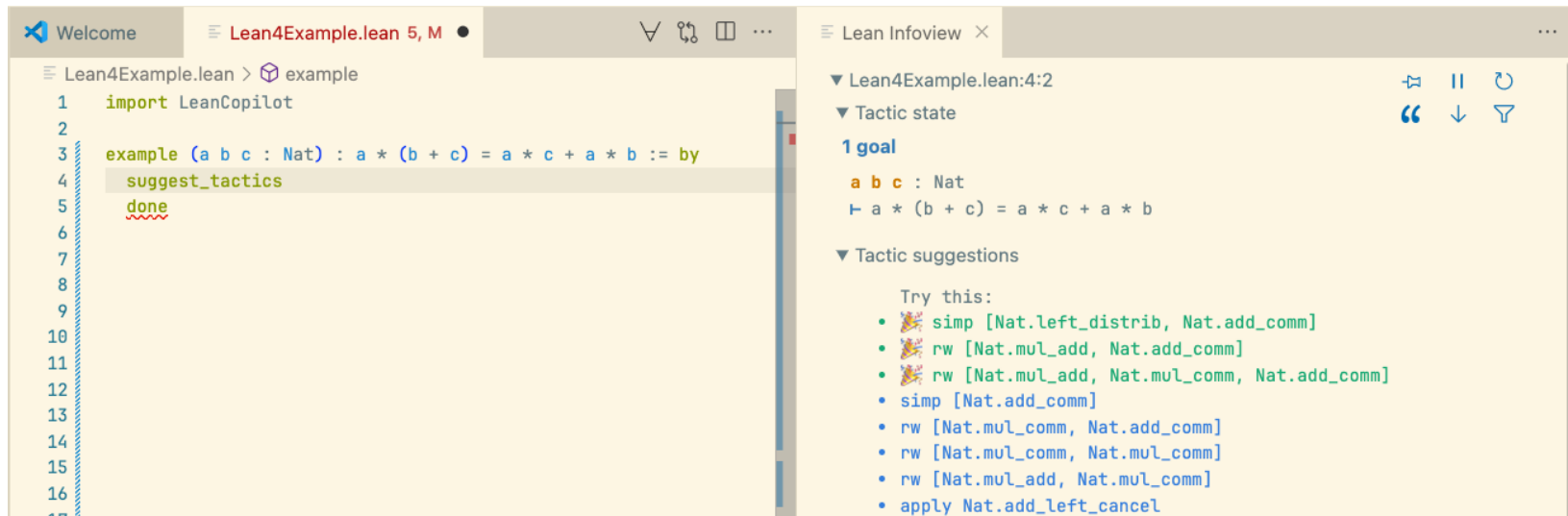
Several groups are developing AI that suggests the **next move**(s) in Lean's interactive proof game.

[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 displays the Lean IDE interface. The left pane shows a Lean 4 code file named `Lean4Example.lean` at line 5. The code defines an example `(a b c : Nat) : a * (b + c) = a * c + a * b` and uses the `suggest_tactics` command. The right pane, titled `Lean Infoview`, shows the current goal state and a list of tactic suggestions.

```
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**

▼ Lean4Example.lean:4:2

▼ Tactic state

**1 goal**

`a b c : Nat`

`⊢ a * (b + c) = a * c + a * b`

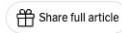
▼ Tactic suggestions

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.



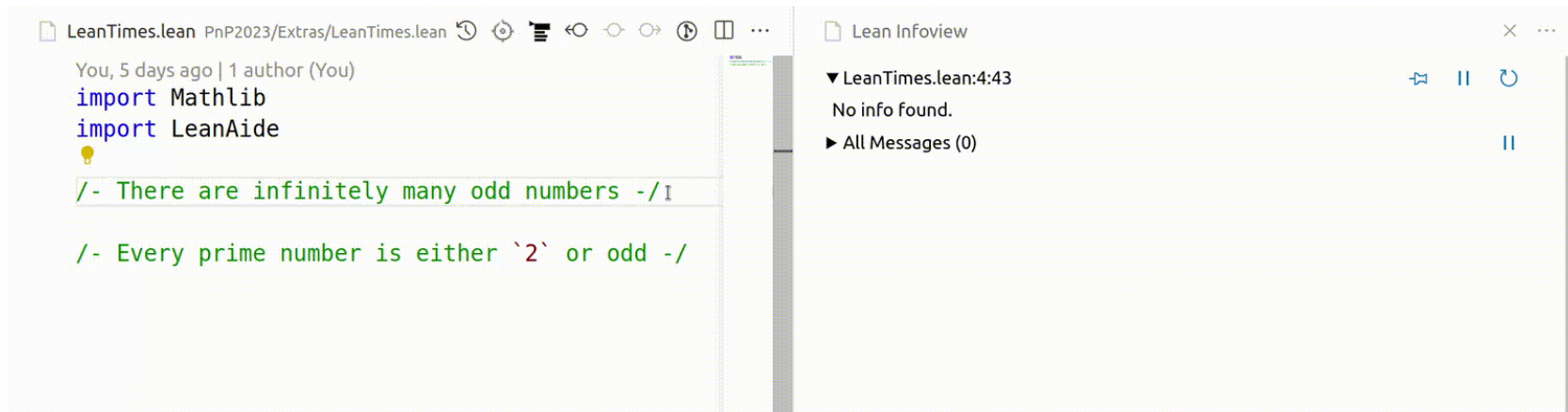
Ring the gong at Google Deepmind's London headquarters, a ritual to celebrate each A.I. milestone, including its recent triumph of reasoning at the International Mathematical Olympiad. Google Deepmind

# 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.



```
LeanTimes.lean PnP2023/Extras/LeanTimes.lean
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 -/

Lean Infview
▼ 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**.

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.

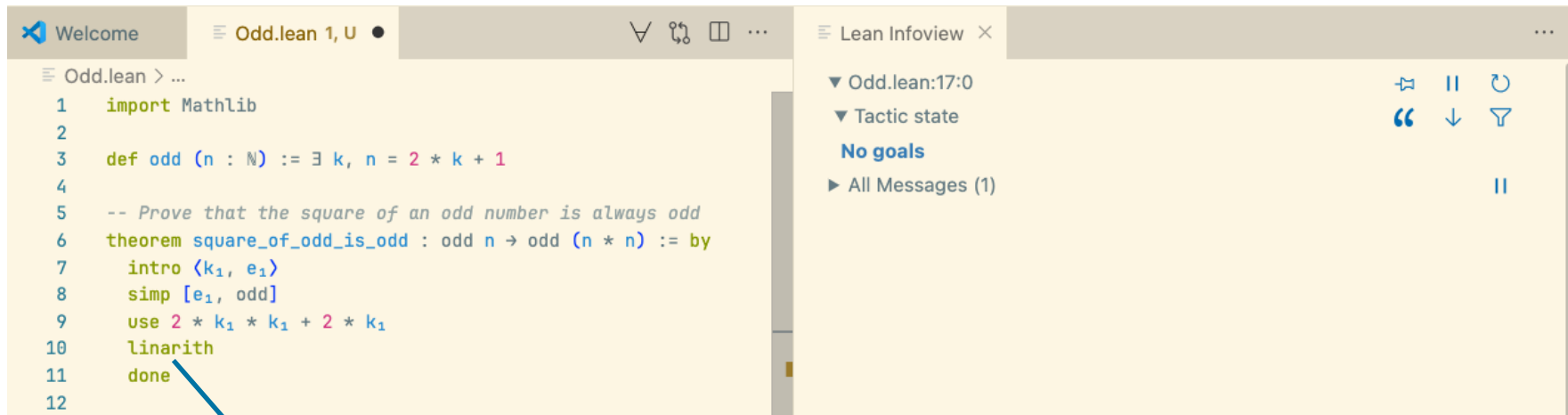
## Machine-Checkable Proofs

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 IDE interface. On the left, a code editor displays a proof script in `Odd.lean`. The script defines an `odd` function and proves a theorem about the square of an odd number. The `linarith` tactic is used to finish the proof. On the right, the 'Lean Infoview' panel shows the current tactic state, which is empty, indicating the proof is complete.

```

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
12

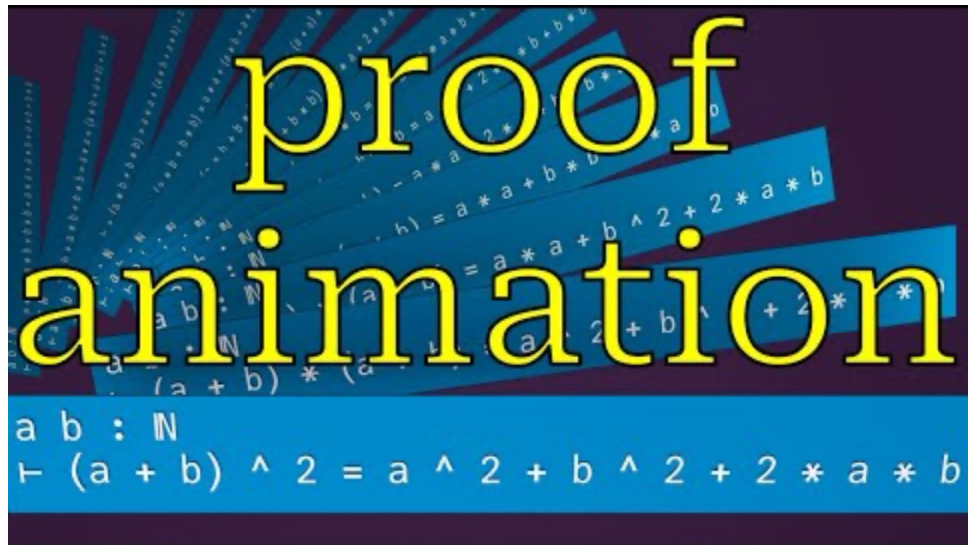
```

The right panel shows the tactic state:

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

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

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**.

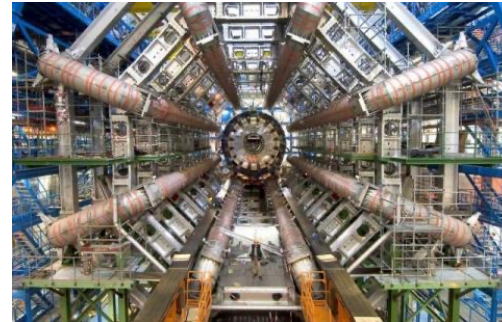
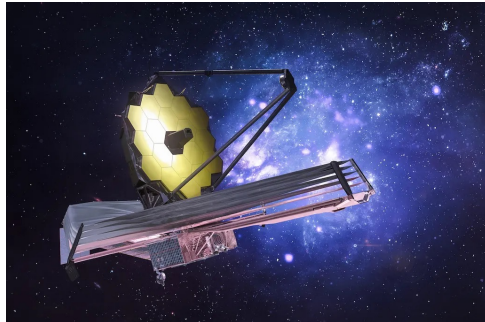
It must reach **self-sustainability in August 2028** and become the **Lean Foundation**.

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.

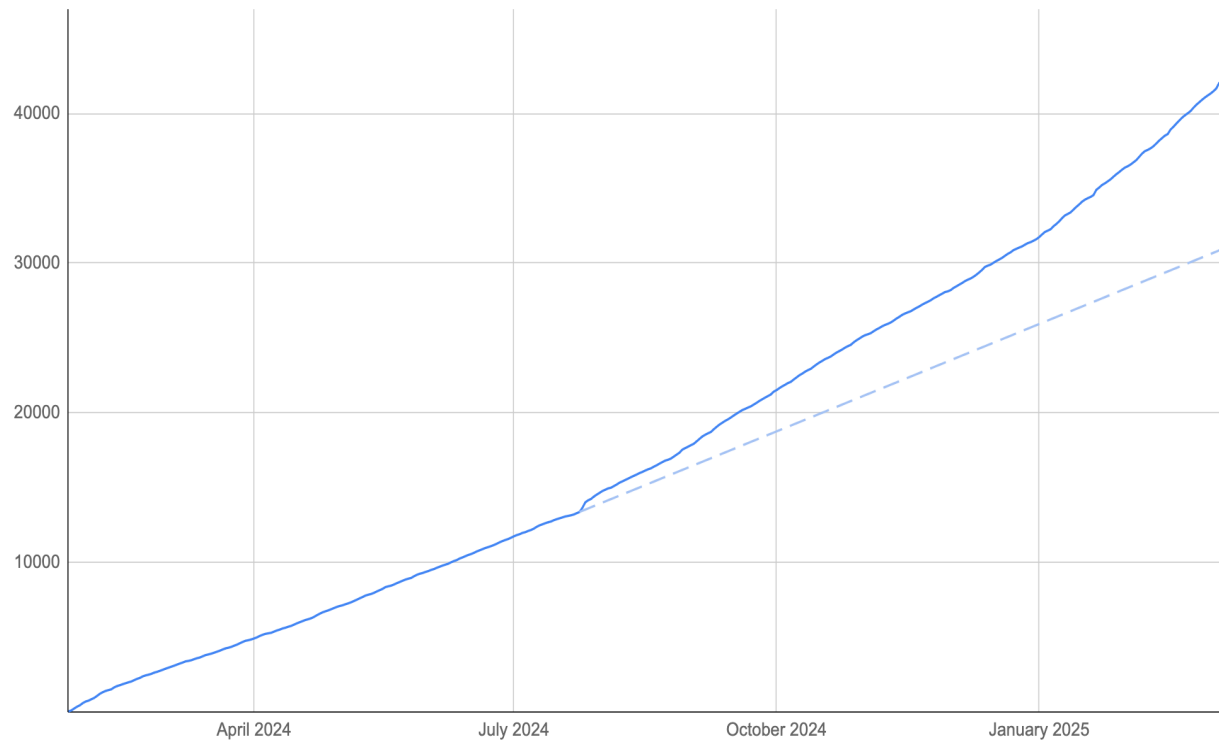
<https://www.convergentresearch.org/about-fros>



# 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](#).

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



## Conclusion

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

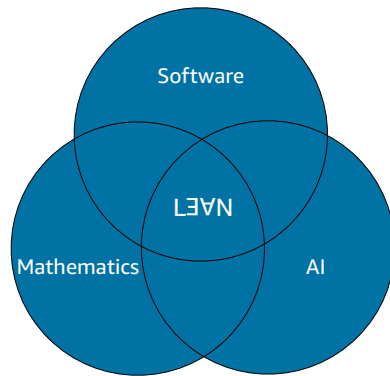
The Mathlib community is changing how math is done.

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/>

