The Lean Theorem Prover and the Formalization of Mathematics

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

June 23, 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.

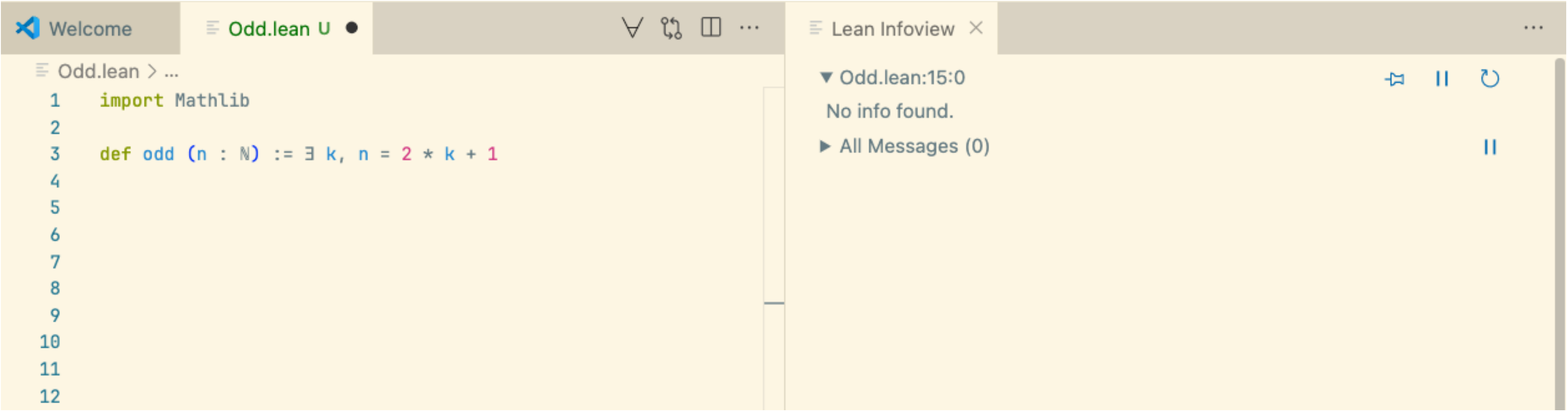
The Lean project, started in 2013, aimed at merging interactive and automated theorem proving.

Lean provides **machine-checkable proofs**.

Lean addresses the “trust bottleneck”.

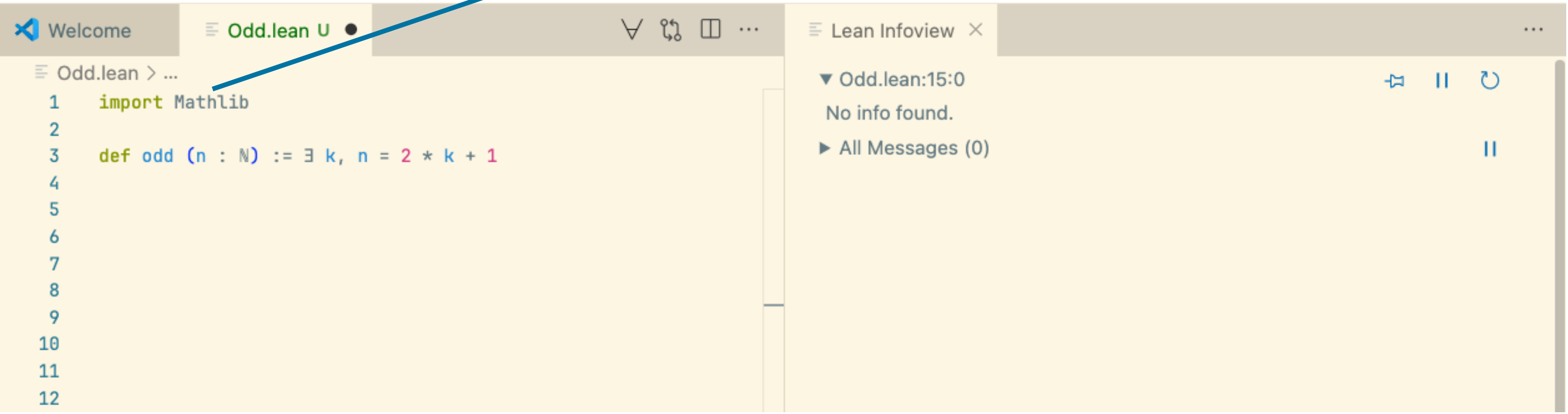
Lean opens up new possibilities for collaboration.

A small example

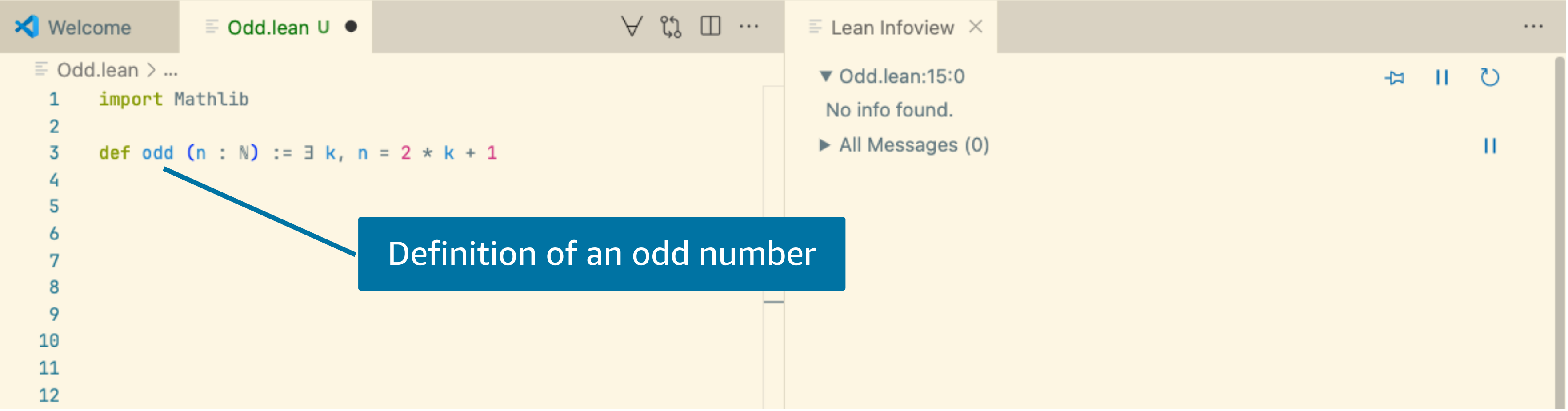


A small example

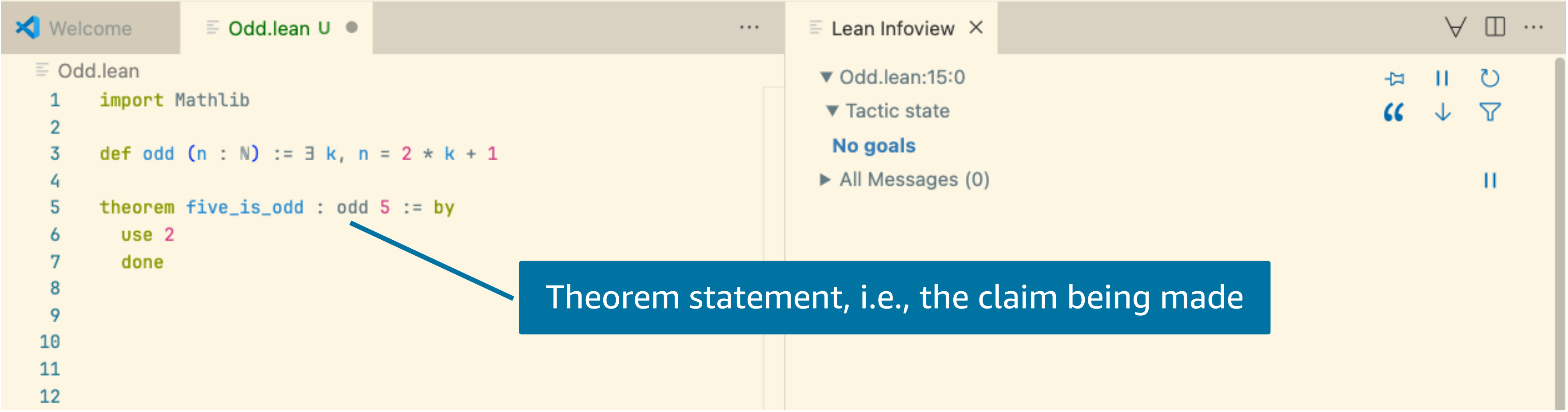
Mathlib is the Lean Mathematical library



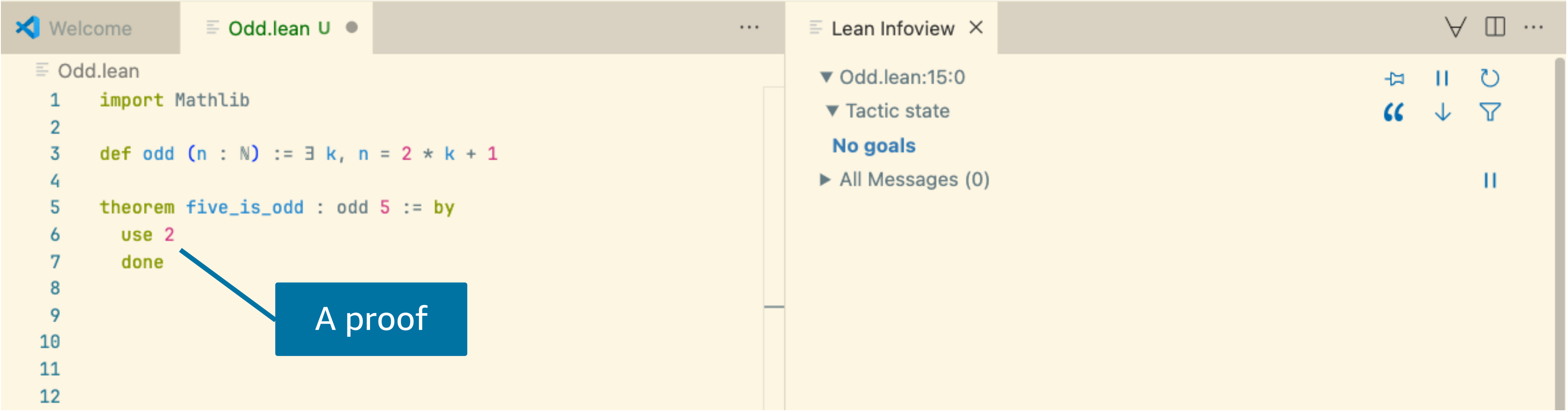
A small example



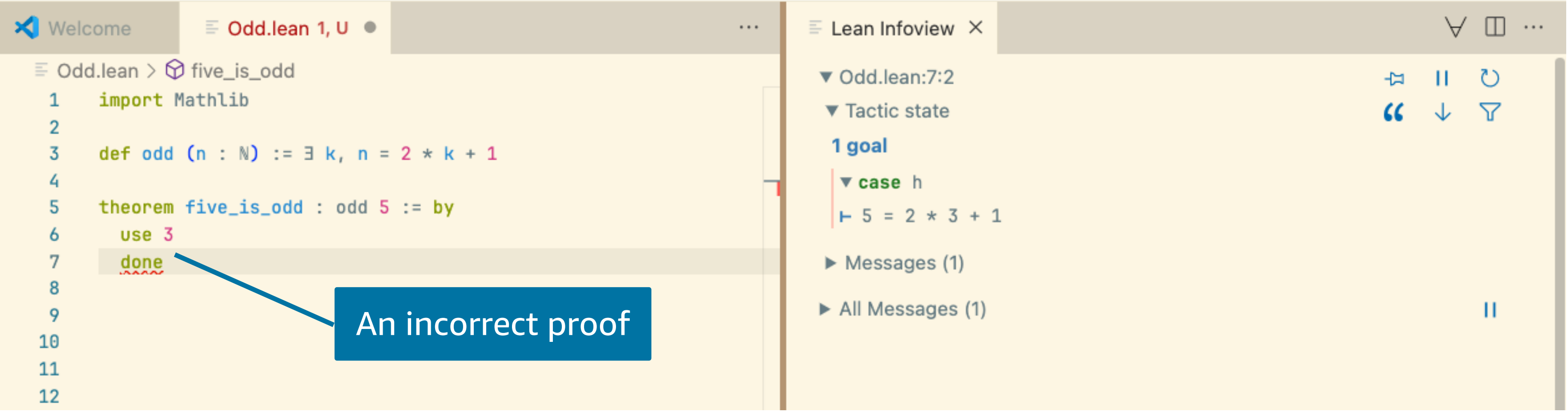
Our first theorem



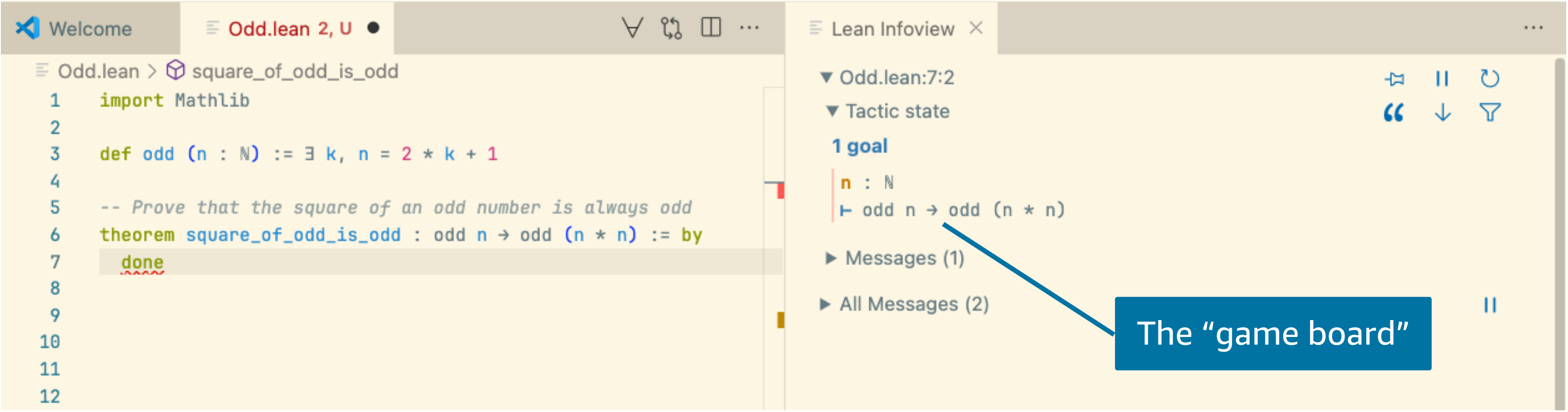
Our first theorem



Our first theorem

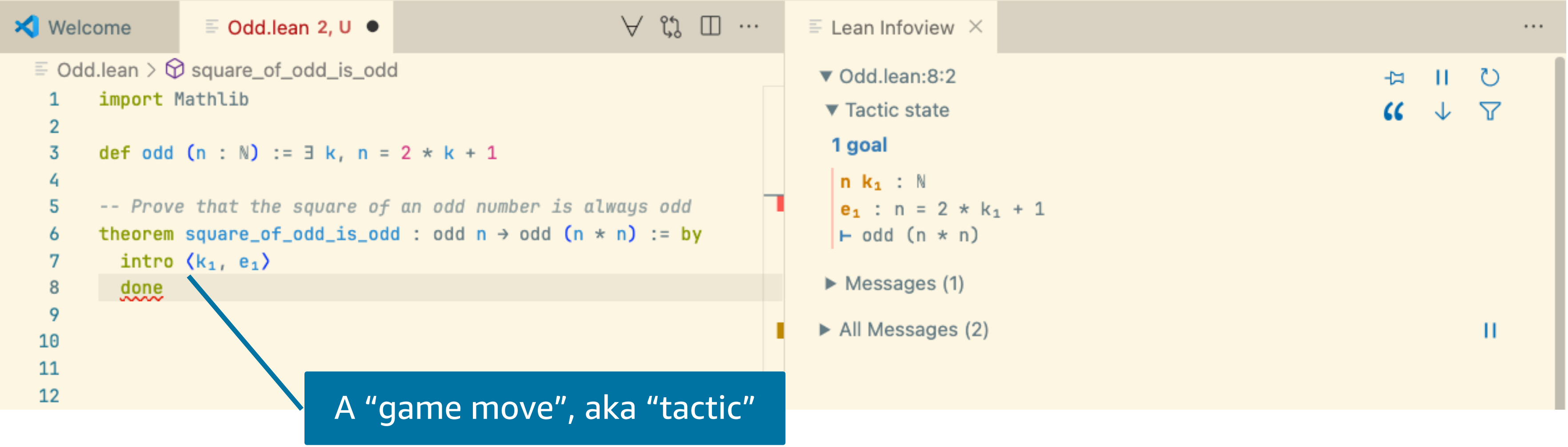


Theorem proving in Lean is an interactive game

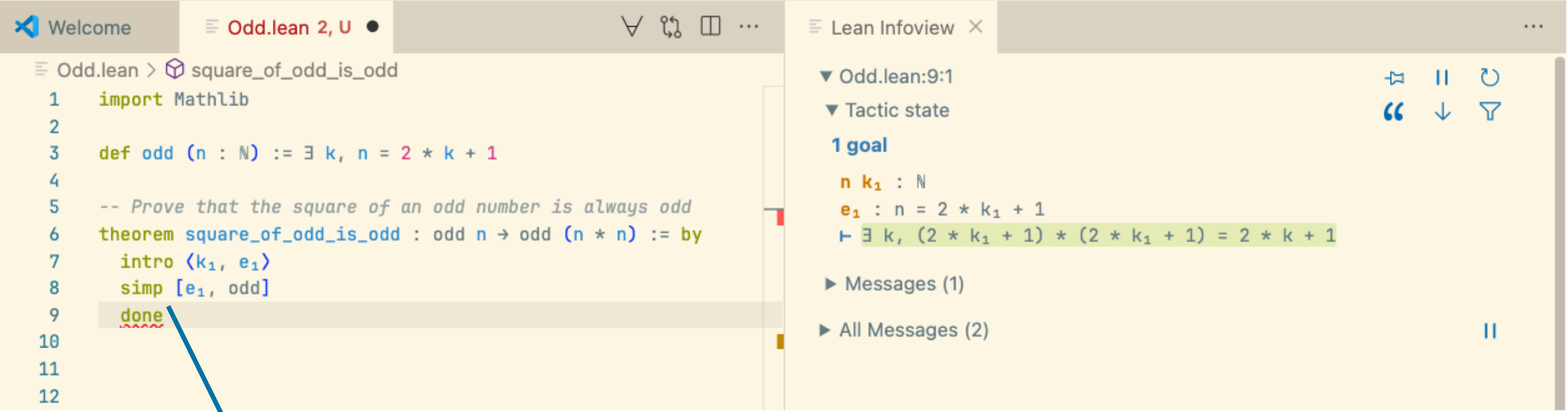


“You have written my favorite computer game”, Kevin Buzzard

Theorem proving in Lean is an interactive game

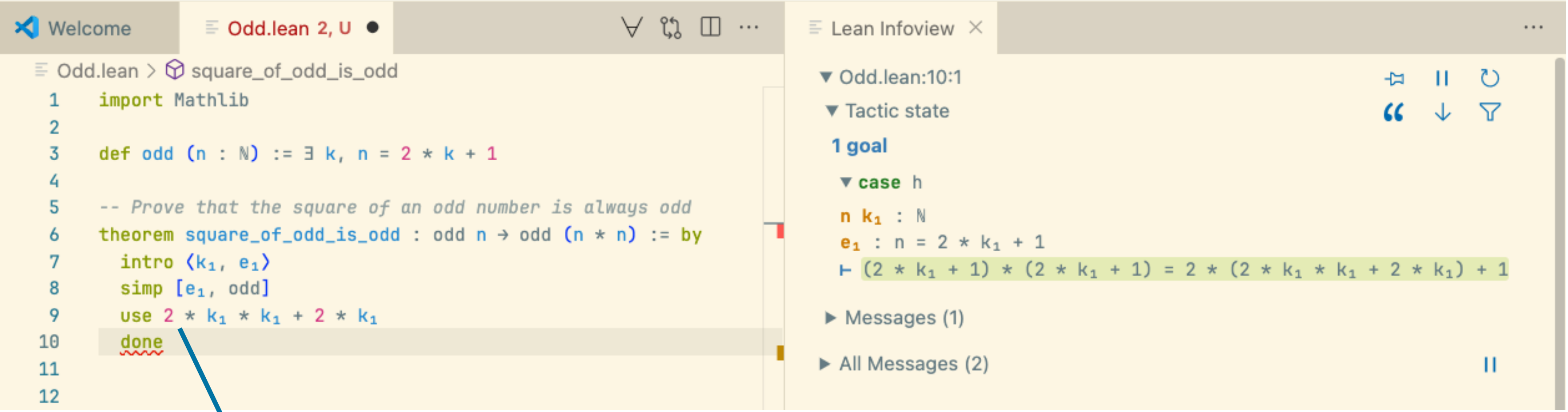


Theorem proving in Lean is an interactive game



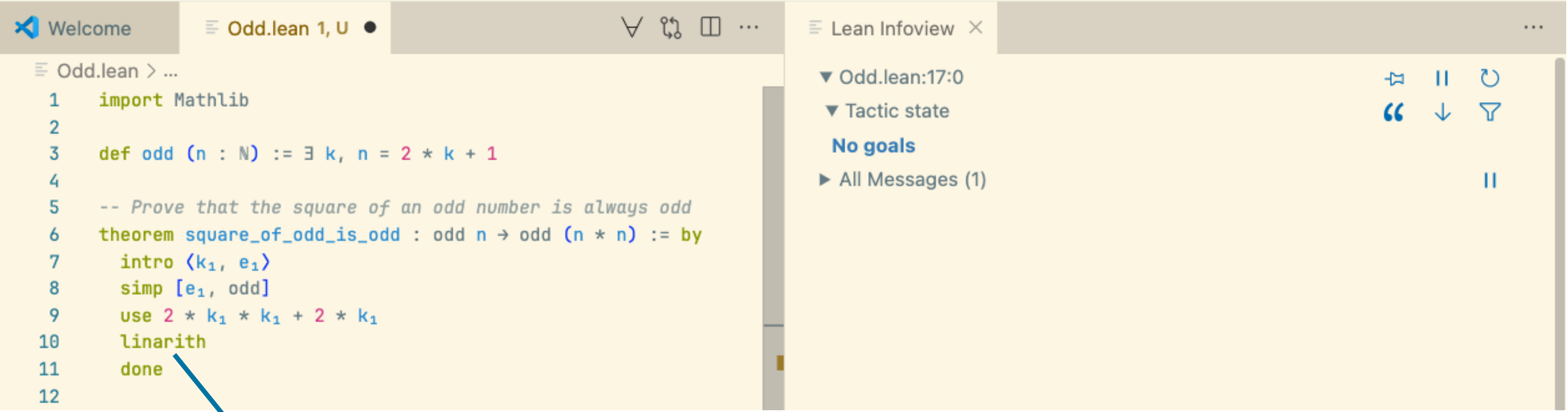
The "game move" `simp`, the simplifier, is one of the most popular moves in our game

Theorem proving in Lean is an interactive game



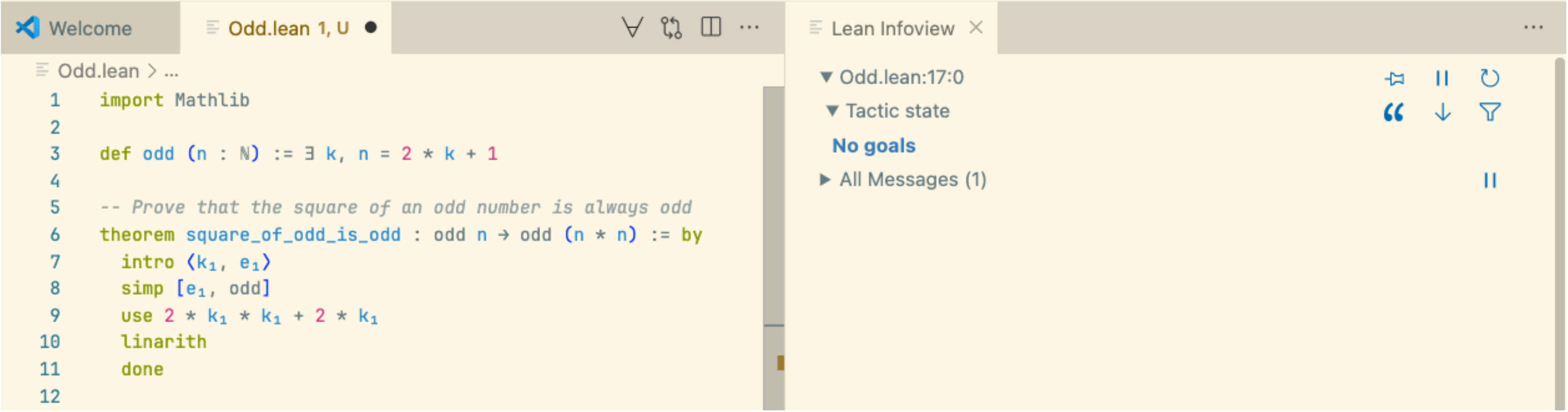
The “game move” `use` is the standard way of proving statements about existentials

Theorem proving in Lean is an interactive game



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

Theorem proving in Lean is an interactive and addictive game



"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

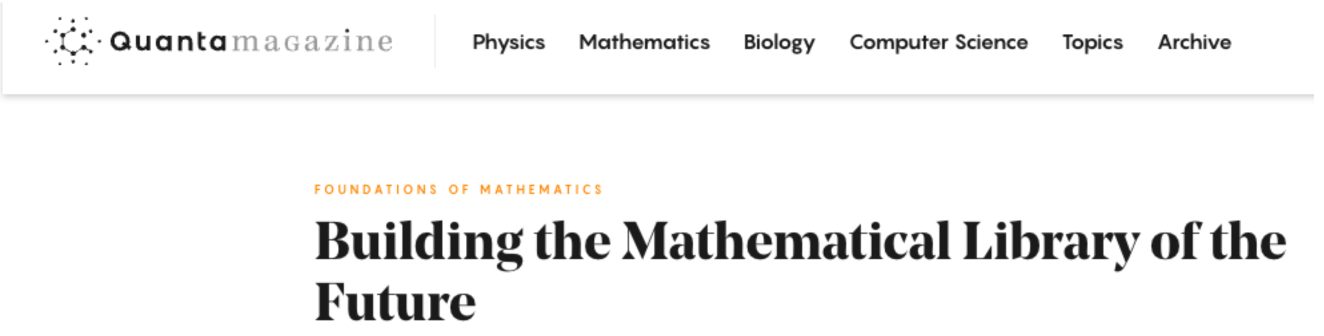
Mathlib

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


Quantamagazine
Physics Mathematics Biology Computer Science Topics Archive

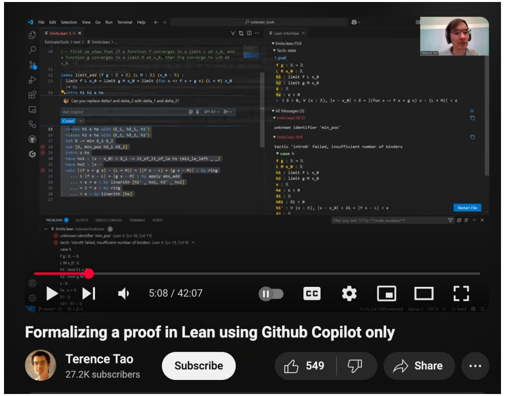
FOUNDATIONS OF MATHEMATICS

Building the Mathematical Library of the Future

Mathematics

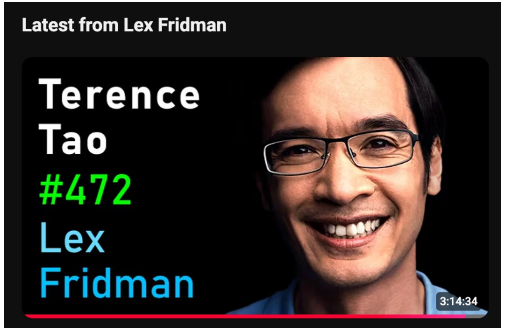
Lean is Taking Mathematics by Storm

"***Lean enables large-scale collaboration*** *by allowing mathematicians to break down complex proofs into smaller, verifiable components. This formalization process ensures the correctness of proofs and facilitates contributions from a broader community.* ***With Lean, we are beginning to see how AI can accelerate the formalization of mathematics, opening up new possibilities for research.***" — *Terence Tao*



Fermat's Last Theorem – Kevin Buzzard

Carleson's Theorem – Floris van Doorn



How did we get here?

Preamble: the Perfectoid Spaces Project

Kevin Buzzard, Patrick Massot, Johan Commelin

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

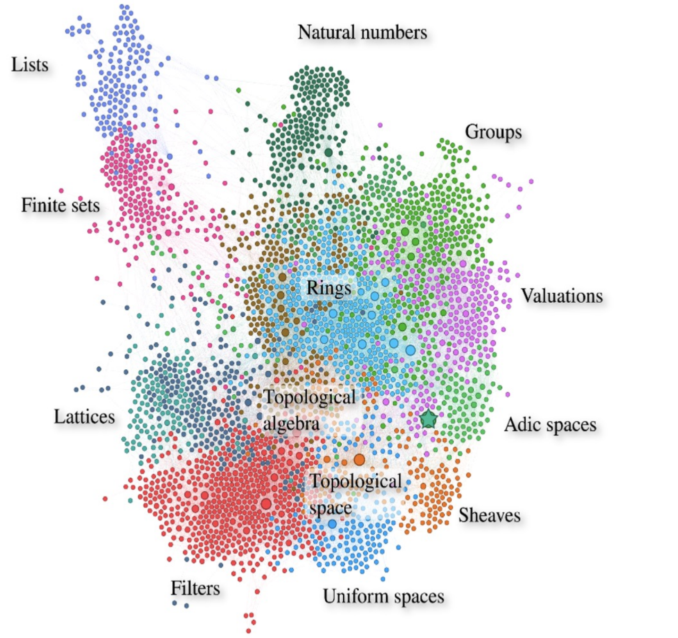
They translated Peter Scholze's definition into a form a computer can understand.

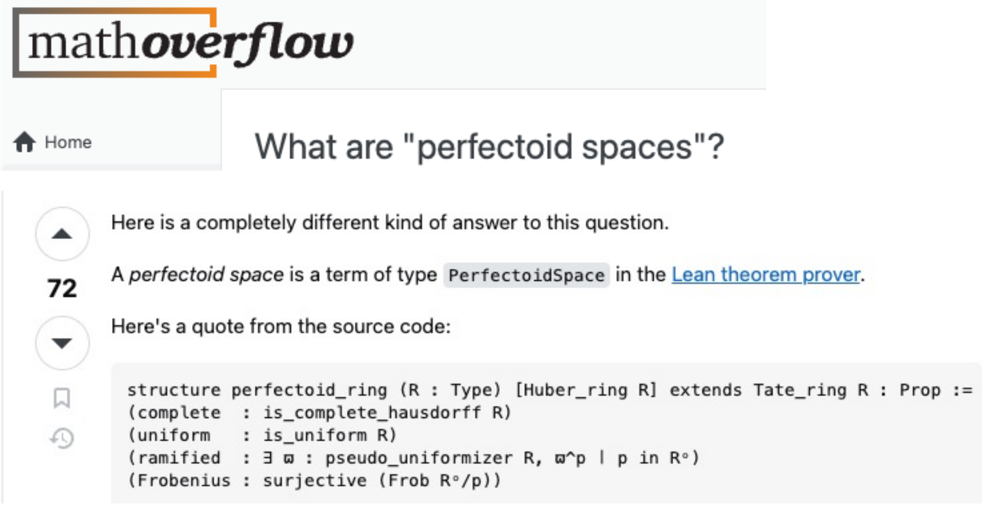
It not only achieved its goals but also demonstrated to the math community that **formal objects can be visualized and inspected with computer assistance**.

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"?

72

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)
(uniform   : is_uniform R)
(ramified  : ∃ ϖ : pseudo_uniformizer R, ϖ^p ∣ p in R°)
(Frobenius : surjective (Frob R°/p))
```

Mathlib > RingTheory > Finiteness.lean

```lean
theorem FG.stabilizes_of_iSup_eq {M' : Submodule R M} (hM' : M'.FG) (N : ℕ →o Submodule R M)
    (H : iSup N = M') : ∃ n, M' = N n := by
  obtain ⟨S, hS⟩ := hM'
  have : ∀ s : S, ∃ n, (s : M) ∈ N n := fun s =>
    (Submodule.mem_iSup_of_chain N s).mp
      (by
        rw [H, ← hS]
        exact Submodule.subset_span s.2)
  choose f hf using this
  use S.attach.sup f
  apply le_antisymm
  · conv_lhs => rw [← hS]
    rw [Submodule.span_le]
    intro s hs
    exact N.2 (Finset.le_sup <| S.mem_attach ⟨s, hs⟩) (hf _)
  · rw [← H]
    exact le_iSup _ _
```

▼Finiteness.lean:365:2

▼Tactic state

1 goal

```
▼case intro
R : Type u_1
M : Type u_2
inst†² : Semiring R
inst†¹ : AddCommMonoid M
inst† : 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' = N n
```

Mathlib > RingTheory > Finiteness.lean

```lean
theorem FG.stabilizes_of_iSup_eq {M' : Submodule R M} (hM' : M'.FG) (N : ℕ →o Submodule R M)
    (H : iSup N = M') : ∃ n, M' = N n := by
  obtain ⟨S, hS⟩ := hM'
  have : ∀ s : S, ∃ n, (s : M) ∈ N n := fun s =>
    (Submodule.mem_iSup_of_chain N s).mp
      (by
        rw [H, ← hS]
        exact Submodule.subset_span s.2)
  choose f hf using this
  use S.attach.sup f
  apply le_antisymm
  · conv_lhs => rw [← hS]
    rw [Submodule.span_le]
    intro s hs
    exact N.2 (Finset.le_sup <| S.mem_attach ⟨s, hs⟩) (hf _)
  · rw [← H]
    exact le_iSup _ _
```

▼Finiteness.lean:365:2

▼Tactic state

1 goal

▼**case** intro

```
R : Type u_1
M : Type u_2
inst✝² : Semiring R
inst✝¹ : AddCommMonoid M
inst✝ : 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

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

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

```
assert_not_exists DivisionRing

open Function

universe u'' u' u v w

variable {G : Type u''} {S : Type u'} {R : Type u} {M : Type v} {ι :

/-- A submodule of a module is one which is closed under vector oper
  This is a sufficient condition for the subset of vectors in the su
  to themselves form a module. -/
structure Submodule (R : Type u) (M : Type v) [Semiring R] [AddCommM
  AddSubmonoid M, SubMulAction R M : Type v
```

structure Submodule (R : Type u) (

▼Finiteness.lean:356:44

▼Expected type

```
R : Type u_1
M : Type u_2
inst✝⁴ : Semiring R
inst✝³ : AddCommMonoid M
inst✝² : Module R M
P : Type u_3
inst✝¹ : AddCommMonoid P
inst✝ : Module R P
f : M →ₗ[R] P
⊢ Type u_2
```

▶ All Messages (0)

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

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

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

```
add_decl_doc Submonoid.toSubsemigroup

/-- `SubmonoidClass S M` says `S` is a type of subsets `s ≤ M` that
and are closed under `(*)` -/
class SubmonoidClass (S : Type*) (M : outParam Type*) [MulOneClass M
  MulMemClass S M, OneMemClass S M : Prop

section

/-- An additive submonoid of an additive monoid `M` is a subset cont
  closed under addition. -/
structure AddSubmonoid (M : Type*) [AddZeroClass M] extends AddSubse
  /-- An additive submonoid contains `0`. -/
  zero_mem' : (0 : M) ∈ carrier
```

structure AddSubmonoid (M : Type

▼ Defs.lean:37:8

▼ Expected type

```
G : Type u''
S : Type u'
R† : Type u
M† : Type v
ι : Type w
R : Type u
M : Type v
inst†² : Semiring R
inst†¹ : AddCommMonoid M
inst† : 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

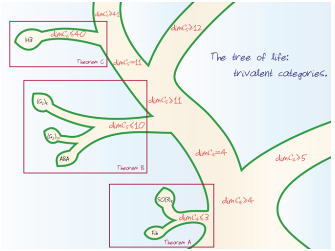
Fermat’s Last Theorem (FLT), Buzzard, 2024-ongoing, community estimates it will take +1M LoC

Automating Quantum Algebra

Here is a concrete example from quantum algebra. It comes from a classification result involving quantum SO(3) categories. Specifically, the condition that certain relations among trivalent graphs imply a constraint on the parameters d, t, and c:

```
example {α} [CommRing α] [IsCharP α 0] (d t c : α) (d_inv PSO3_inv : α)
  (Δ40 : d^2 * (d + t - d * t - 2) *
    (d + t + d * t) = 0)
  (Δ41 : -d^4 * (d + t - d * t - 2) *
    (2 * d + 2 * d * t - 4 * d * t^2 + 2 * d * t^4 + 2 * d^2 * t^4 - c * (d + t + d * t)) = 0)
  (_ : d * d_inv = 1)
  (_ : (d + t - d * t - 2) * PSO3_inv = 1) :
  t^2 = t + 1 := by grind
```

From: “Categories generated by a trivalent vertex”, Morrison, Peters, and Snyder



Automating Quantum Algebra

```
example {α} [CommRing α] [IsCharP α 0] (d t c : α) (d_inv PSO3_inv : α)
  (Δ40 : d^2 * (d + t - d * t - 2) *
    (d + t + d * t) = 0)
  (Δ41 : -d^4 * (d + t - d * t - 2) *
    (2 * d + 2 * d * t - 4 * d * t^2 + 2 * d * t^4 + 2 * d^2 * t^4 - c * (d + t + d * t)) = 0)
  (_ : d * d_inv = 1)
  (_ : (d + t - d * t - 2) * PSO3_inv = 1) :
  t^2 = t + 1 := by grind
```

This is not a toy: it encodes a real algebraic constraint derived from relations among diagrams in a pivotal tensor category.

Lean can handle this kind of reasoning automatically, in milliseconds.

Automating Quantum Algebra

We can explore new mathematical and physical structures, from topological quantum fields theories to fusion categories.

Lean is helping researchers reason reliably about complex symbolic systems that were previously handled only by hand or with unverified computer algebra.

`grind` is just another move in our interactive game.

Refactoring Math

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

"We had formalized the proof with this constant 12, and then when this new paper came out, we said, 'Okay, let's update the 12 to 11.' And what you can do with Lean is that you just in your headline theorem change a 12 to 11. You run the compiler and… of the thousands of lines of code you have, 90% of them still work, and there are a couple that are lined in red… **it immediately isolates which steps you need to change, and you can skip over everything which works just fine.**" – Terence Tao on Lex Fridman

Reasoning at the right level of abstraction

"I'm interested in developing some API for linearly ordered vector spaces, in order to easily handle manipulations of asymptotic orders" – Terence Tao on the Lean Zulip

```
example {R} [OrderedVectorSpace R] (x y z : R)
        : x ≤ 2•y → y < z → x < 2•z := by
 grind -- 🎉
```

OrderedVectorSpace implements IntModule, LinearOrder, IntModule.IsOrdered.

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**.

We don't need to trust our automation/moves.

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

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

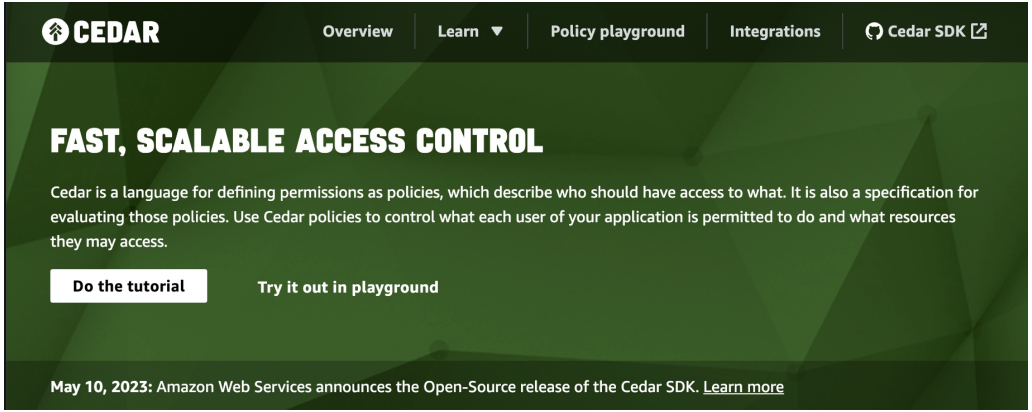
You can write code and reason about it simultaneously.

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

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

Cedar

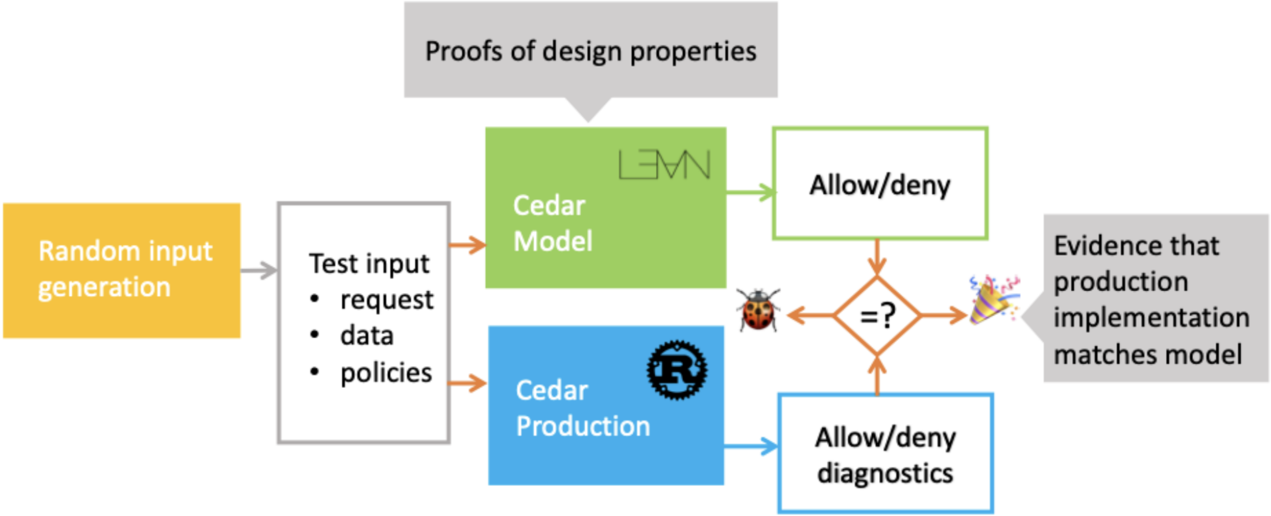
https://www.cedarpolicy.com/



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 }
```

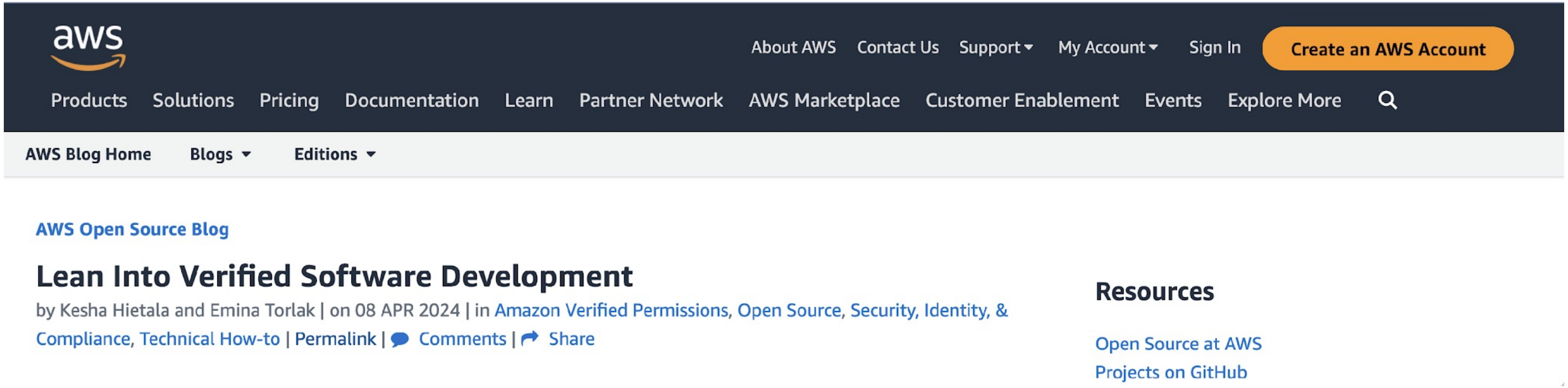
Cedar



*"**Lean is the core verification technology behind Cedar**, the open-source authorization language that powers cloud services like Amazon Verified Permissions and AWS Verified Access. Our team rigorously formalizes and verifies core components of Cedar using Lean's proof assistant, and we leverage **Lean's lightning-fast runtime** to continuously test our production Rust code against the Lean formalization. Lean's efficiency, extensive libraries, and vibrant community **enable us to develop and maintain Cedar at scale**, while ensuring the key correctness and security properties that our users depend on." — Emina Torlak, Senior Principal Applied Scientist, AWS*

Cedar

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



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.

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**.

“For SampCert, I started using Lean because of Mathlib, but I realized that Lean isn't just an excellent proof assistant, it's also a very pleasant and efficient programming language with a great ecosystem. As a result, we continued using Lean for TenCert.” Jean-Baptiste Tristan

Verifying Cryptography with Aeneas at Microsoft

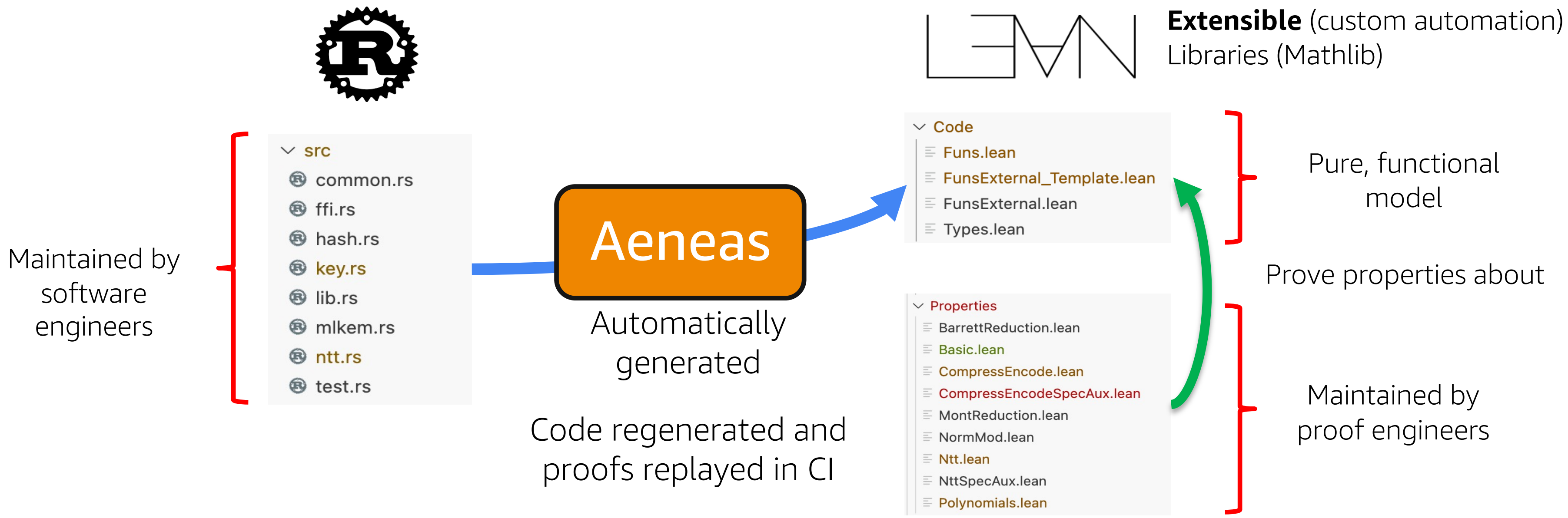
They verify (and fix/improve) the Rust code as written by software engineers.

Code is evolving (new optimizations for specific hardware): They must adapt to rewrites.

Rewriting SymCrypt in Rust to modernize Microsoft's cryptographic library.

*"**The verification crucially relies on the Lean** interactive theorem prover, whose **extensibility** has been **key** in developing custom automation to make verification amenable in an industrial setting. Lean FRO" – Son Ho*

Verifying Cryptography with Aeneas at Microsoft



KLR: a language and elaborators for machine learning kernels

Define a common representation for kernel functions with a precise formal semantics along with translations from common kernel languages to the KLR core language.

"The lean meta programming is amazing. Have managed to delete hundreds of lines of boilerplate in the last couple days." Sean McLaughlin

KLR is also open source.

```
private def evalTensorScalar (ts : TensorScalar) (t: ByteArray) : Err ByteArray := do
  match ts with
  | TensorScalar.mk op0 c0 rev0 op1 c1 rev1 =>
  let f0 <- evalAluOp op0
  let f1 <- evalAluOp op1
  let c0 := c0.toLEByteArray
  let c1 := c1.toLEByteArray
  apply2 f0 rev0 c0 f1 rev1 c1 t
```

KLR: a language and elaborators for machine learning kernels

KLR uses bit-vectors, fixed integers, etc.

```
private def decBV64 : DecodeM (BitVec 64) :=
  let u8_64 : DecodeM UInt64 := next >>= fun x => return x.toUInt64
  return ((<- u8_64) <<< 0  |||
          (<- u8_64) <<< 8  |||
          (<- u8_64) <<< 16 |||
          (<- u8_64) <<< 24 |||
          (<- u8_64) <<< 32 |||
          (<- u8_64) <<< 40 |||
          (<- u8_64) <<< 48 |||
          (<- u8_64) <<< 56).toBitVec
```

bv_decide: another powerful move

A verified bit-blaster by **Henrik Boving**, Josh Clune, Siddharth Bhat, and Alex Keizer

Uses LRAT proof producing SAT solvers: **Cadical**

```
/-
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
```

“Blasting” popcount with bv_decide

```
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
```

“Blasting” popcount with bv_decide

Imp.lean > { } Imp.Stmt > popcount_correct

```
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 &&&
```

Does Lean Have Hammers?

The Lean community is also actively developing automation.

LeanHammer: an automated reasoning tool for Lean which brings together multiple proof search and reconstruction techniques and combine them into one tool.

Lean-SMT: An SMT tactic for discharging proof goals in Lean

"Improving automation for proofs in Lean is an exciting research direction. ***Lean-SMT aims to improve automation by enabling the automatic replay in Lean of proof certificates produced*** *by SMT solvers."*
Clark Barrett

grind (again)

```
example (x : BitVec 16) : (x + 256)*(x - 256) = x^2 := by
  grind
```

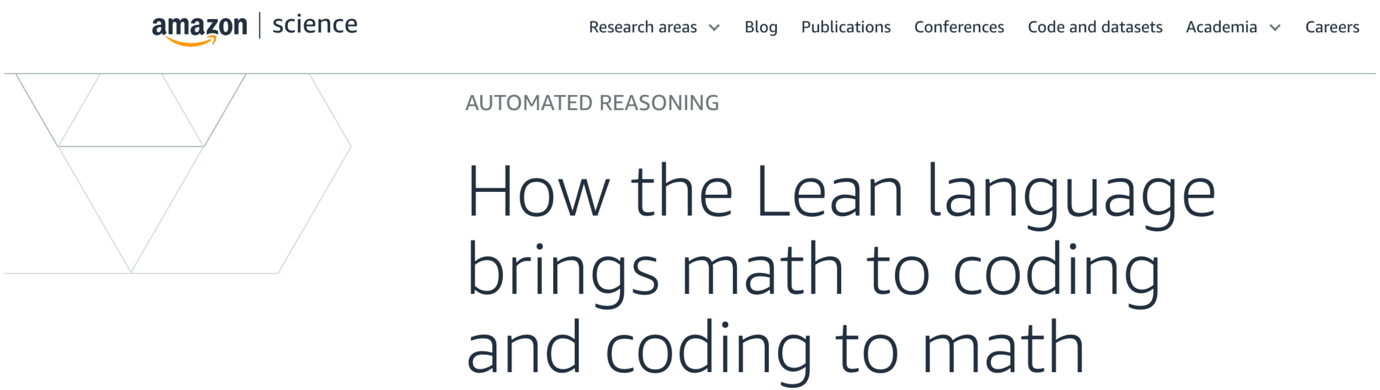
```
def siftDown (a : Array Int) (root : Nat) (e : Nat) (h : e ≤ a.size := by grind) : Array Int :=
  if _ : leftChild root < e then
    let child := leftChild root
    let child := if _ : child+1 < e then
      if a[child] < a[child + 1] then child + 1 else child
    else child
    if a[root] < a[child] then
      let a := a.swap root child
      siftDown a child e
    else a
  else a
termination_by e - root

theorem siftDown_size {a root e h} : (siftDown a root e h).size = a.size := by
   fun_induction siftDown <;> grind [siftDown]
```

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/


amazon | science

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.

Synthetic Data Generation

LLMs require **vast amounts of data** for training.

Lean mathematical libraries provide valuable, **correct-by-construction training data**.

AILean, a project led by **Soonho Kong** at AWS, uses Lean to generate **new synthetic theorems** that are correct by construction. Soonho will go much deeper later today.

Pantograph by Leni Aniva (Stanford) is also getting very popular in the Lean community.

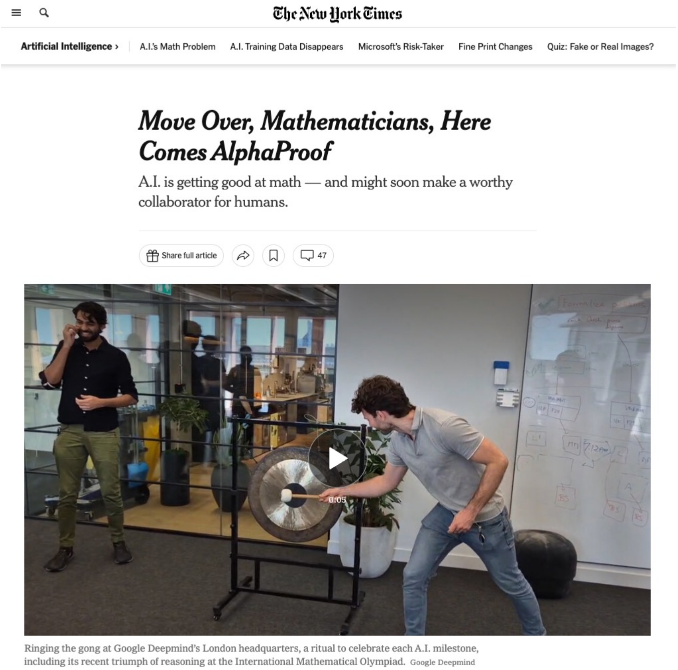
AI Proof Assistants

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.

Claude 4 is fantastic on Lean code. Their System Card contains a Lean example.


The New York Times

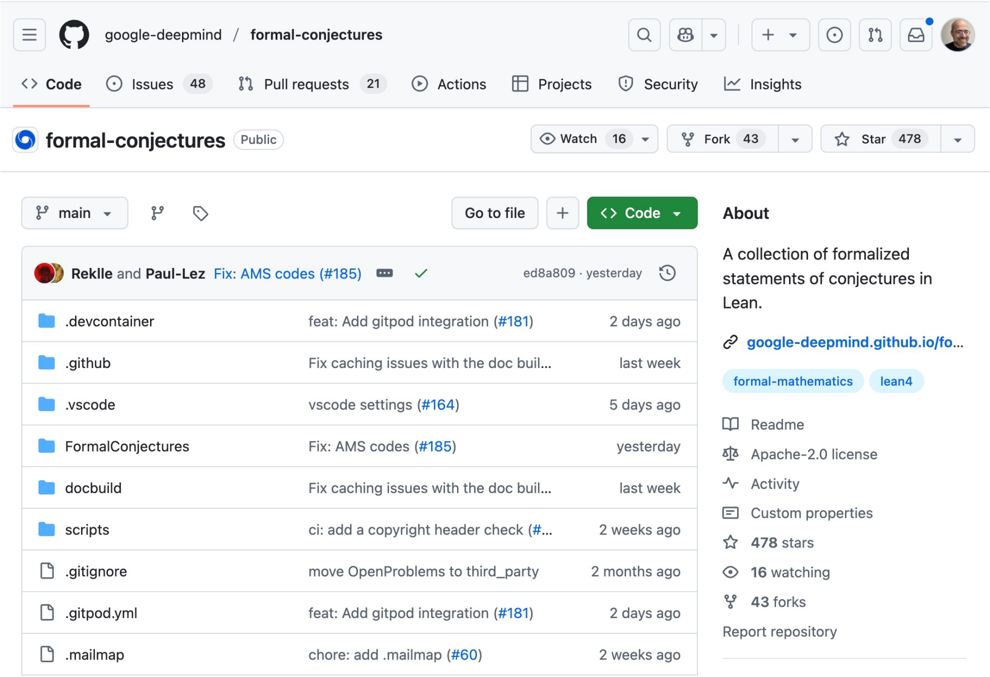
Artificial Intelligence › A.I.'s Math Problem A.I. Training Data Disappears Microsoft's Risk-Taker Fine Print Changes Quiz: Fake or Real Images?

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



"At Google DeepMind, we used Lean to build AlphaProof, a new reinforcement-learning based system for formal math reasoning ***. Lean's extensibility and verification capabilities were key in enabling the development of AlphaProof*** *." — Pushmeet Kohli, Vice President, Research Google DeepMind*

Auto-formalization

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

Bhavik Mehta · 1st
Chapman Fellow in Mathematics at Imperial College Lo...
4d · Edited ·

Thrilled to share a major milestone from Big Proof in Cambridge! 🚀 It was an immense honour to present alongside some of the most prestigious mathematicians of our time.

A highlight? Introducing Trinity, a revolutionary auto-formalisation agent. This innovative tool is part of Christian Szegedy's verified superintelligence program with Morph Labs.

Morph Labs has used Trinity to auto-formalise a proof that the famous abc conjecture is true almost always, producing over 3500 lines of Lean.

Want to learn more about my work and see Jared and me discuss Trinity's incredible capabilities? Check out the session recording: https://lnkd.in/eifg42Z5 The section 45:00 - 59:00 is unmissable, make sure to watch it all!

#FormalMathematics #AI #ProofAutomation #BigProof #Math #Lean

You and 71 others · 3 comments · 5 reposts

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.

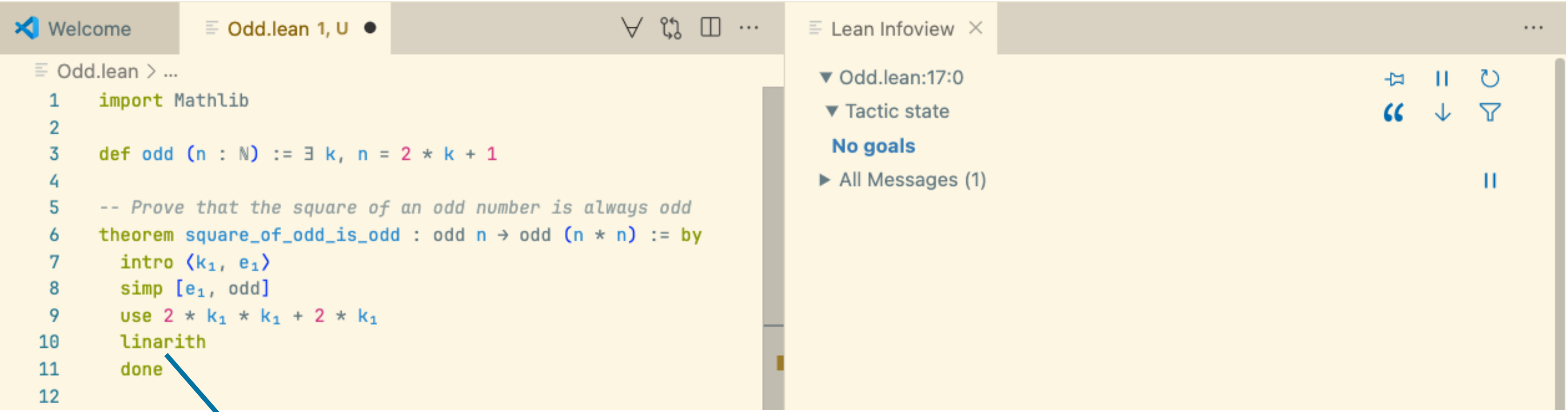
Machine-Checkable Proofs

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

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

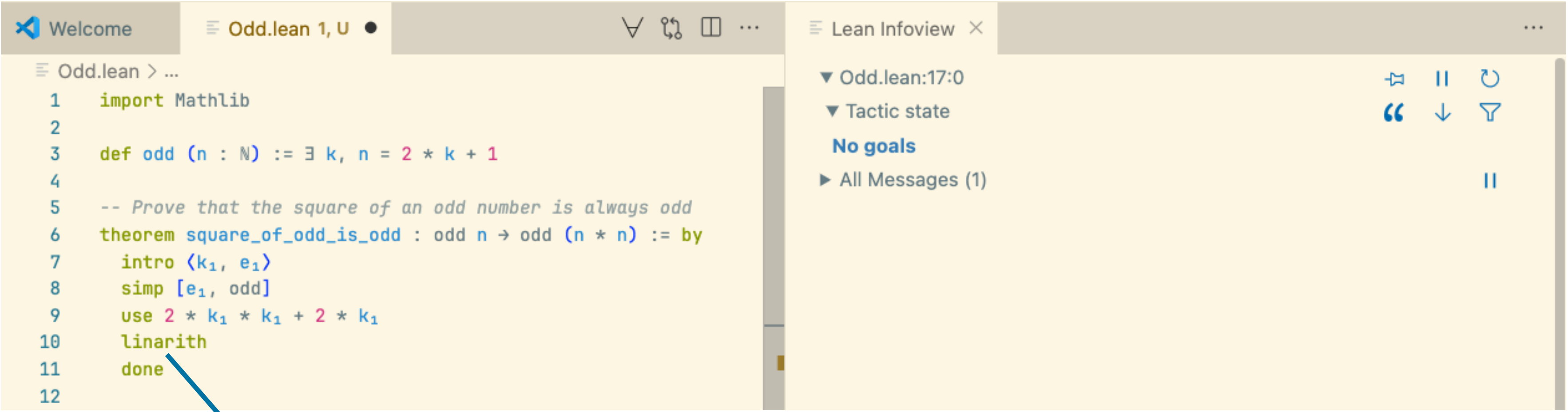
Code without fear.

Lean is a game where we can implement your own moves



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

Lean is a game where we can implement your own moves



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

The `bv_decide` and `grind` “moves” are also implemented in Lean!

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**.

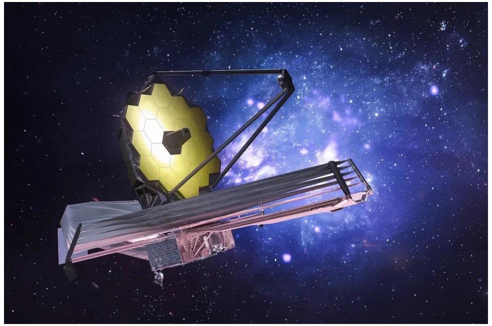
It must reach **self-sustainability in August 2028** and become the **Lean Foundation**.

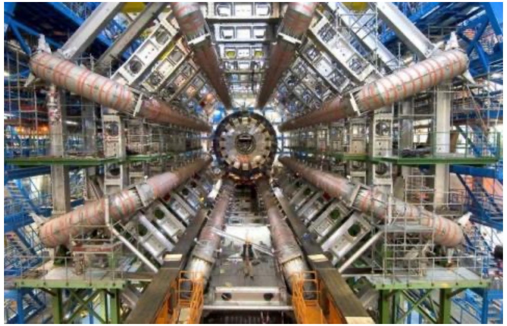
We are very grateful for all philanthropic support we have received.

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

20 releases and **4,383 pull requests** merged in the main repository only since its launch in July 2023.

Public roadmaps: https://lean-fro.org/about/roadmap-y2/

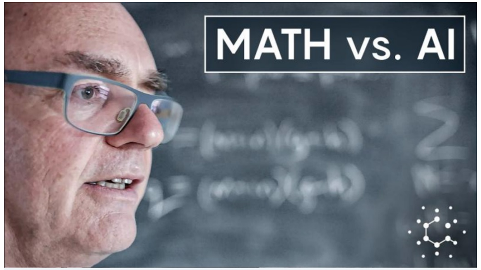
Lean project was featured in multiple venues NY Times, Quanta, Scientific American, etc.

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

Lean FRO: Roadmap

Lean v4.22's release will celebrate the Lean FRO's second anniversary

Many new features coming in the Lean FRO year 3.

New Compiler - Enhanced performance and optimization

New Module System - Faster recompilation and better dependency management

Improved do-notation - better support for reasoning about it

Enhanced Proof Automation - Continue improving bv_decide, grind, simp

Scalability Improvements - Handle larger codebases efficiently

Literate Programming System - Seamless documentation integration

New Website - Modern interface and better resources

CSLib

A Mathlib for computer science.

Steering committee of CSLib:

Swarat Chaudhuri (Google DeepMind and UT Austin)

Clark Barrett (Stanford University and Amazon)

Jason Gross (Theorama)

Leo de Moura (Amazon and Lean FRO)

CSLib aims to be a foundation for **teaching**, **research**, and new **verification** efforts, including AI-assisted.

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.

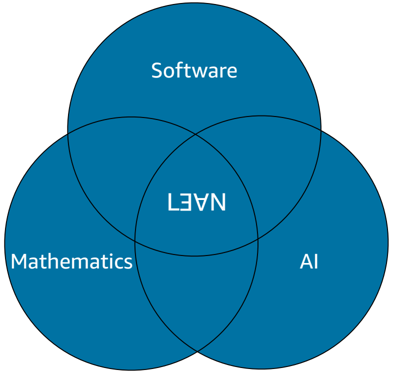
Want to contribute to a large formalization project? Join the FLT formalization project.

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/