Lean 4: Empowering the Formal Mathematics Revolution and Beyond

Leonardo de Moura
Senior Principal Applied Scientist - AWS
Chief Architect - Lean FRO

LEAN Proof Assistant & Programming Language

Based on dependent type theory

Goals

- Extensibility, Expressivity, Scalability, Efficiency

A platform for

- Formalized mathematics
- Software development and verification
- Developing custom automation and Domain Specific Languages

Small trusted kernel, external type/proof checkers

LEAN is and IDE fo formal mathematics

Lean is a development environment for formal mathematics.

Proofs and definitions are machine checkable.

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

A compiler for mathematics: high-level language ⇒ kernel code

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

LEAN is easy to install

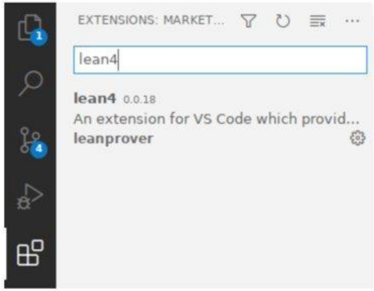
Lean Manual

Quickstart

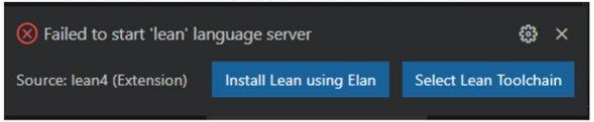
These instructions will walk you through setting up Lean using the "basic" setup and VS Code as the editor. See Setup for other ways, supported platforms, and more details on setting up Lean.

1. Install VS Code.

2. Launch VS Code and install the `lean4` extension.



3. Create a new file using "File > New File" and click the `Select a language` link and type in `lean4` and hit ENTER. You should see the following popup:



LEAN enables decentralized collaboration

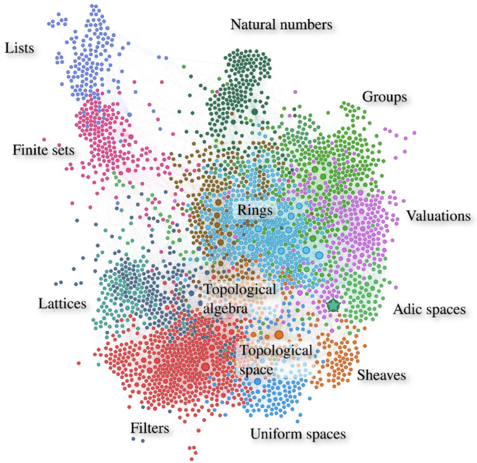
Meta-programming

Users extend Lean using Lean itself.

Proof automation.

Visualization tools.

Custom notation.



Formal Proofs

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

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

Hack without fear.

LEAN an example

```
class Distrib (α : Type u) extends Mul α, Add α where
  left_distrib  : ∀ a b c : α, a*(b+c) = a*b + a*c
  right_distrib : ∀ a b c : α, (a+b)*c = a*c + b*c

class Ring (α : Type u) extends AddCommGroup α, Monoid α, Distrib α
```

LEAN an example

```
class Distrib (α : Type u) extends Mul α, Add α where
  left_distrib  : ∀ a b c : α, a*(b+c) = a*b + a*c
  right_distrib : ∀ a b c : α, (a+b)*c = a*c + b*c

class Ring (α : Type u) extends AddCommGroup α, Monoid α, Distrib α
```

```
/--
  `Matrix m n α` is the type of matrices whose rows are indexed by `m`
  and whose columns are indexed by `n`, and the elements have type `α`. -/
def Matrix (m : Type u) (n : Type v) (α : Type w) : Type (max u v w) :=
  m → n → α
```

LEAN an example

```
class Distrib (α : Type u) extends Mul α, Add α where
  left_distrib  : ∀ a b c : α, a*(b+c) = a*b + a*c
  right_distrib : ∀ a b c : α, (a+b)*c = a*c + b*c

class Ring (α : Type u) extends AddCommGroup α, Monoid α, Distrib α
```

```
/--
  `Matrix m n α` is the type of matrices whose rows are indexed by `m`
  and whose columns are indexed by `n`, and the elements have type `α`. -/
def Matrix (m : Type u) (n : Type v) (α : Type w) : Type (max u v w) :=
  m → n → α
```

```
/- Scoped notation for accessing values stored in matrices. -/
scoped syntax:max term noWs "[" term ", " term "]" : term

macro_rules
  | `($x[$i, $j]) => `($x $i $j)

instance [Add α] : Add (Matrix m n α) where
  add x y i j := x[i, j] + y[i, j]
```

LEAN an example

```lean
class Distrib (α : Type u) extends Mul α, Add α where
  left_distrib  : ∀ a b c : α, a*(b+c) = a*b + a*c
  right_distrib : ∀ a b c : α, (a+b)*c = a*c + b*c

class Ring (α : Type u) extends AddCommGroup α, Monoid α, Distrib α
```

```lean
/--
  `Matrix m n α` is the type of matrices whose rows are indexed by `m`
  and whose columns are indexed by `n`, and the elements have type `α`. -/
def Matrix (m : Type u) (n : Type v) (α : Type w) : Type (max u v w) :=
  m → n → α
```

```lean
/- Scoped notation for accessing values stored in matrices. -/
scoped syntax:max term noWs "[" term ", " term "]" : term

macro_rules
  | `($x[$i, $j]) => `($x $i $j)

instance [Add α] : Add (Matrix m n α) where
  add x y i j := x[i, j] + y[i, j]
```

```lean
instance  [Fintype n] [DecidableEq n] [Ring α] : Ring (Matrix n n α) :=
  { Matrix.semiring, Matrix.addCommGroup with }
```

an example

```
class Distrib (α : Type u) extends Mul α, Add α where
  left_distrib  : ∀ a b c : α, a*(b+c) = a*b + a*c
  right_distrib : ∀ a b c : α, (a+b)*c = a*c + b*c

class Ring (α : Type u) extends AddCommGroup α, Monoid α, Distrib α
```

```
/--
  `Matrix m n α` is the type of matrices whose rows are indexed by `m`
  and whose columns are indexed by `n`, and the elements have type `α`. -/
def Matrix (m : Type u) (n : Type v) (α : Type w) : Type (max u v w) :=
  m → n → α
```

```
/- Scoped notation for accessing values stored in matrices. -/
scoped syntax:max term noWs "[" term ", " term "]" : term

macro_rules
  | `($x[$i, $j]) => `($x $i $j)

instance [Add α] : Add (Matrix m n α) where
  add x y i j := x[i, j] + y[i, j]
```

```
instance  [Fintype n] [DecidableEq n] [Ring α] : Ring (Matrix n n α) :=
  { Matrix.semiring, Matrix.addCommGroup with }
```

```
@[simp]
theorem transpose_add [Add α] (M : Matrix m n α) (N : Matrix m n α) : (M + N)ᵀ = Mᵀ + Nᵀ := by
  ext i j
  simp
```

Should we trust LEAN?

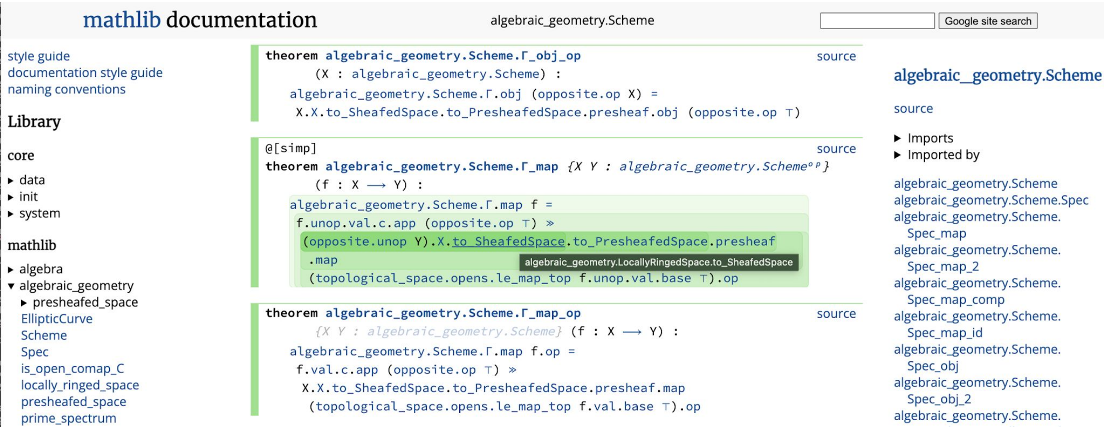
Lean has a small trusted proof checker.

Do I need to trust the checker?

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

You can implement your own checker.

LEAN Community develops Mathlib

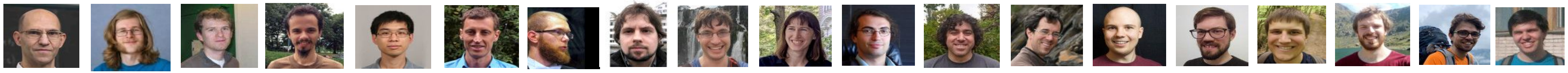


The Lean Mathematical Library

The mathlib Community*

Abstract

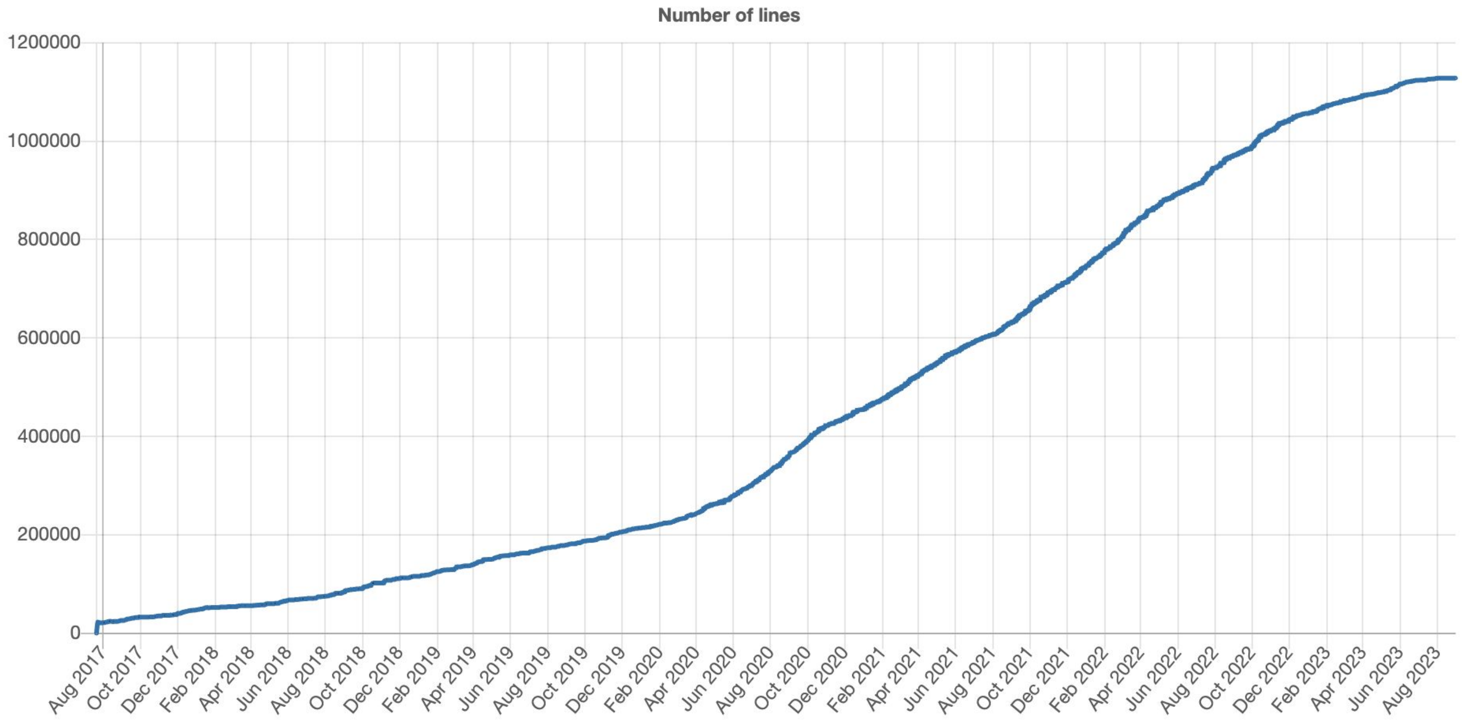
This paper describes mathlib, a community-driven effort to build a unified library of mathematics formalized in the Lean proof assistant. Among proof assistant libraries, it is distinguished by its dependently typed foundations, focus on classical mathematics, extensive hierarchy of structures, use of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



Mathlib statistics

Counts

Definitions	Theorems	Contributors
66599	122987	310



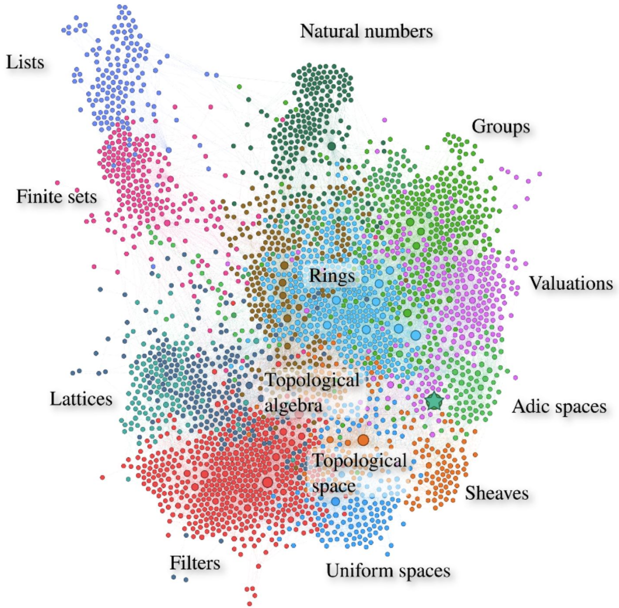
Lean perfectoid spaces

by Kevin Buzzard, Johan Commelin, and Patrick Massot

What is it about?

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

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

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

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

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

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

end
```



mathoverflow

Home

Questions

Tags

What are "perfectoid spaces"?

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

67

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

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

Here's a quote from the source code:

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

The LEAN ecosystem

Lean developers

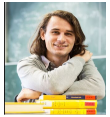
Mathematicians

AI Research

Software developers

Students

mathlib



FAB STRACTS

FLYPITCH

formally proving the independence of the continuum hypothesis

Lean perfectoid spaces

by Kevin Buzzard, Johan Commelin, and Patrick Massot

The LEAN ecosystem

Lean developers

Mathematicians

AI Research

Software developers

Students

OpenAI GPT-f for Lean
Facebook AI

"This will help make Lean a prime choice for machine learning research."

IMO Grand Challenge

The LEAN ecosystem

Lean developers

Mathematicians

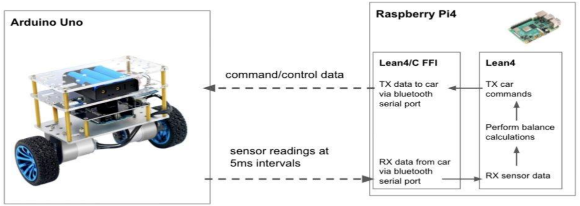
AI Research

Software developers

Students

A great language for Math is also a great language for programming.

Lean is a language for "programming your proofs and proving your programs"



The LEAN ecosystem

Lean developers

Mathematicians

AI Research

Software developers

Students

We can reach self-motivated students with no access to formal math education.

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

condensed mathematics > **Condensed R-modules** — Oct 07

Peter Scholze (EDITED)

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

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

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

lean-gptf > **OpenAI gpt-f key** — Oct 08

Stanislas Polu — 6:03 AM

@Ayush Agrawal let me check 👍

👍 1

We had a bit of a backlog — 6:33 AM

Good think you reached out. Invites are out.

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

👍 1

FLT regular > **Cyclotomic field defn** — Oct 25

Eric Rodriguez — 10:09 AM

I noticed this project so far is working with `adjoin_root cyclotomic`. I wonder if instead, `X^n-1.splitting_field` is a better option. I think the second option is better suited to Galois theory (as then the `.gal` has good defeq) and also easier to generalise to other fields. (it works for all fields with n ≠ 0, whilst I think this one may not)

general > **Bachelor thesis accomplished** 🎉 — Today

Giacomo Maletto — 9:52 AM

Hello, I'm a math student at University of Turin and I've been using proof assistants for about a year, with the objective of formalizing a computer science paper written by my advisor (about a class of functions similar in spirit to primitive recursive functions, but which are all invertible).

After a lot of work here's my thesis! https://github.com/GiacomoMaletto/RPP/blob/main/Tesi/main.pdf (Lean code in the same repo).

It's written in an informal, colloquial manner and I tried to turn it into an introduction/invitation to Lean.

Actually I've used Coq for 90% of the duration of the project, completed it, and then switched to Lean - doing basically the same thing in about 750 LOC instead of >3000. I'm not turning back.

Looking forward to start using Lean for something more involved!

🎉 17 · 1

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

Jia Xuan Ng (EDITED)

Hi everyone, I'm trying to prove ∀ x y z : A, x ≠ y → (x ≠ z ∨ y ≠ z) :=, which I believe to be provable.

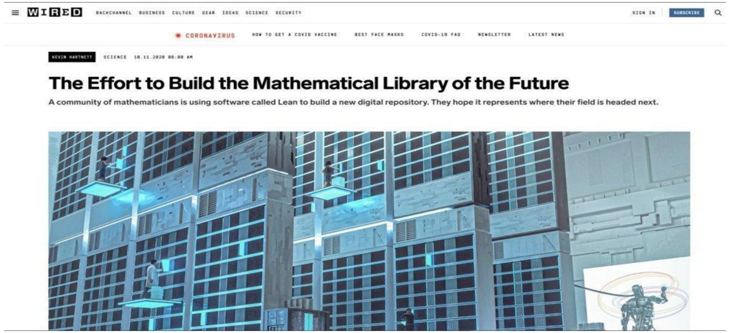
Reason why this is is because I use implication logical equivalences e.g. P -> Q === !P V Q such that I derived:

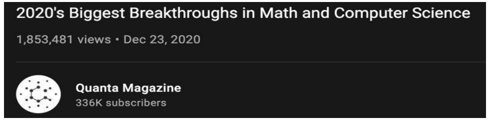
x ≠ y → ¬ (x ≠ z) → y ≠ z ==> x ≠ y → x = z → y ≠ z which is essentially stating:

"If x isn't equivalent to y, if x is equivalent to z, then y isn't equivalent to z", which is a tautology.

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

The Lean Mathematical Library goes viral – 2020





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

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

The Liquid Tensor Experiment (LTE) – 2021

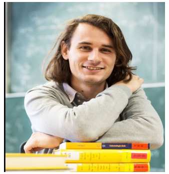
Peter Scholze (Fields Medal 2018) was unsure about one of his latest results in Analytic Geometry.

The Lean community and Scholze formalized the result he was unsure about.

We thought it would take years (Scholze included).

Trust agnostic collaboration allowed us to achieve it in months. (Math Hive in action).

"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*



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

2023 has been a great year for LEAN

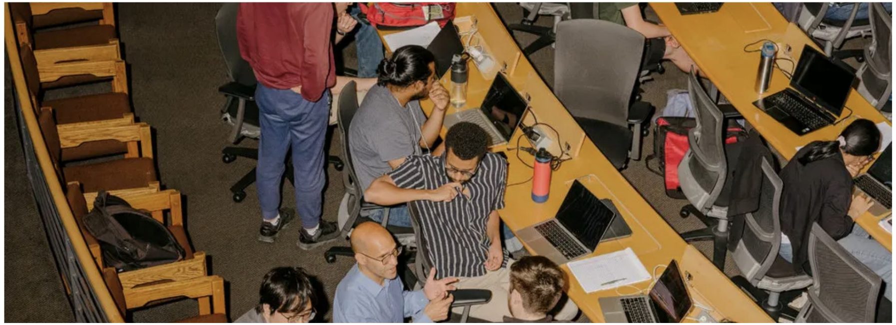
The New York Times

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

A.I. Is Coming for Mathematics, Too

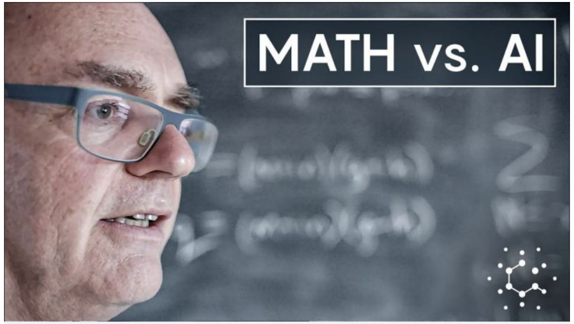
For thousands of years, mathematicians have adapted to the latest advances in logic and reasoning. Are they ready for artificial intelligence?

Give this article



Terence Tao
@tao@mathstodon.xyz

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

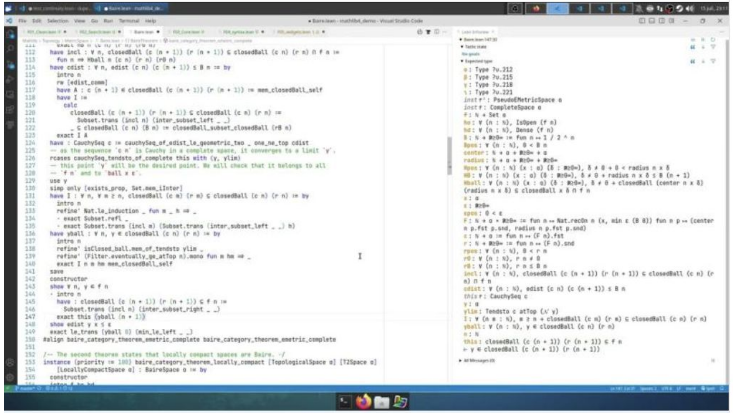


When Computers Write Proofs, What's the Point of Mathematicians?
youtube.com

2023 has been a great year for LEAN

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

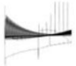
I am thrilled to announce that the Mathlib (https://lnkd.in/gx6eh4aG) port to Lean 4 has been successfully completed this weekend. It is truly remarkable that over 1 million lines of formal mathematics have been successfully migrated. Once again, the community has amazed me and surpassed all my expectations.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



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

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

Ecstatic to come across the following post today! 😃 Here is the link to the original: https://lnkd.in/dSDFSVhS, and website: https://lnkd.in/dB9427pU



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

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

Abstract Formalities

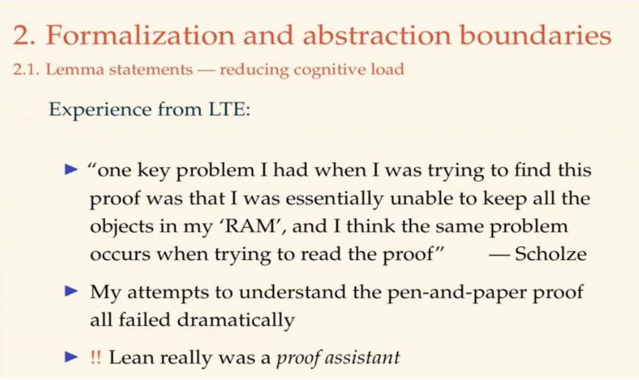
Johan Commelin's talk: http://www.fields.utoronto.ca/talks/Abstract-Formalities

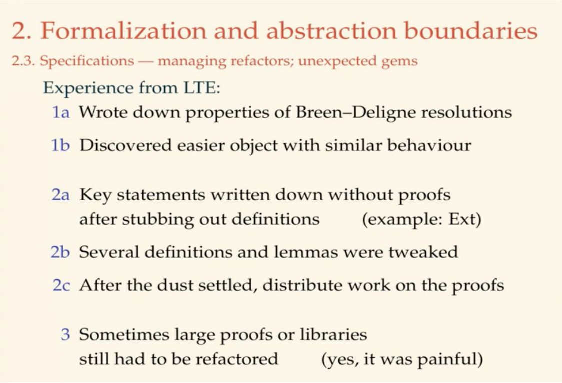
Abstraction boundaries in Mathematics.

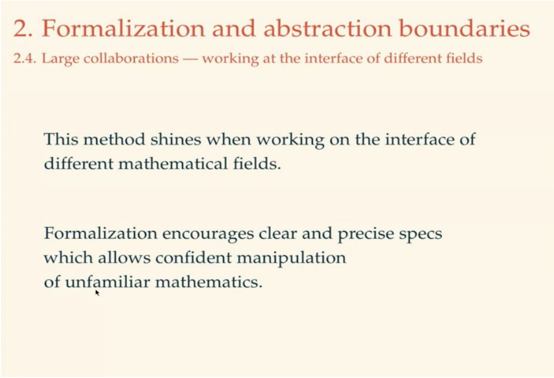
Formal mathematics as a tool for reducing the cognitive load.

Not just from raw proof complexity, but also

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







Extensibility

We build **with (not for)** the community

Mathlib is not just math, but many Lean extensions too.

The community extends Lean using Lean itself.

We wrote Lean 4 in Lean to make sure every single part of the system is extensible.

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

Lean 4 is an efficient programming language

We want proof automation written by users to be very efficient.

Lean memory manager is **now** the Bing memory manager (Daan Leijen - RiSE).

"Functional but in Place" (FBIP) distinguished paper award at PLDI'21.

Proofs are used to optimize code too.

It is a fully extensible programming language.

There are many more surprises coming...

Lean is a language for "programming your proofs and proving your programs"

Domain Specific Languages in Lean

Extensible Parser and Hygienic Macro System

```
syntax "{ " ident (" : " term)? " // " term " }" : term

macro_rules
  | `({ $x : $type // $p }) => `(Subtype (fun ($x:ident : $type) => $p))
  | `({ $x // $p })         => `(Subtype (fun ($x:ident : _) => $p))
```

We have many different syntax categories.

```
syntax   stx "+" : stx
syntax   stx "*" : stx
syntax   stx "?" : stx
syntax:2 stx " <|> " stx:1 : stx

macro_rules
  | `(stx| $p +) => `(stx| many1($p))
  | `(stx| $p *) => `(stx| many($p))
  | `(stx| $p ?) => `(stx| optional($p))
  | `(stx| $p₁ <|> $p₂) => `(stx| orelse($p₁, $p₂))
```

Hygiene

```
notation "const" e ⇒ fun x ⇒ e
```

"Of course" *e* may not capture *x*

```
macro "elab" ... ⇒ do
  ...;
  `(@[$elabAttr] def myElaborator (stx : Syntax) : $type := match_syntax stx with ...)
```

"Of course" *myElaborator* may not be captured from outside

“do” notation : another DSL

```
def Poly.eval? (e : Poly) (a : Assignment) : Option Rat := Id.run do
  let mut r := 0
  for (c, x) in e.val do
    if let some v := a.get? x then
      r := r + c*v
    else
      return none
  return r
```

“do” notation : another DSL

```
private def congrApp (mvarId : MVarId) (lhs rhs : Expr) : MetaM (List MVarId) :=
  lhs.withApp fun f args => do
    let infos := (← getFunInfoNArgs f args.size).paramInfo
    let mut r := { expr := f : Simp.Result }
    let mut newGoals := #[]
    let mut i := 0
    for arg in args do
      let addGoal ←
        if i < infos.size && !infos[i].hasFwdDeps then
          pure infos[i].binderInfo.isExplicit
        else
          pure (← whnfD (← inferType r.expr)).isArrow
      if addGoal then
        let (rhs, newGoal) ← mkConvGoalFor arg
        newGoals := newGoals.push newGoal.mvarId!
        r ← Simp.mkCongr r { expr := rhs, proof? := newGoal }
      else
        r ← Simp.mkCongrFun r arg
      i := i + 1
    let proof ← r.getProof
    unless (← isDefEqGuarded rhs r.expr) do
      throwError "invalid 'congr' conv tactic, failed to resolve{indentExpr rhs}\n=?={indentExpr r.expr}"
    assignExprMVar mvarId proof
    return newGoals.toList
```

Tactic/synthesis framework: another DSL

Go to tactic/synthesis mode

```
variables {α : Type u} {β : Type v}
variables {ra  : α → α → Prop} {rb  : β → β → Prop}

def lexAccessible (aca : (a : α) → Acc ra a) (acb : (b : β) → Acc rb b) (a : α) (b : β) : Acc (Lex ra rb) (a, b) := by
  induction (aca a) generalizing b
  | intro xa aca iha =>
    induction (acb b)
    | intro xb acb ihb =>
      apply Acc.intro (xa, xb)
      intro p lt
      cases lt
      | left  a₁ b₁ a₂ b₂ h => apply iha a₁ h
      | right a b₁ b₂ h     => apply ihb b₁ h
```

Construct a lambda

Construct an application

The tactic framework is implemented in Lean itself

```
def cases (mvarId : MVarId) (majorFVarId : FVarId) (givenNames : Array (List Name)) (useUnusedNames : Bool) : MetaM (Array CasesSubgoal) :=
  withMVarContext mvarId do
    checkNotAssigned mvarId `cases
    let context? ← mkCasesContext? majorFVarId
    match context? with
    | none     => throwTacticEx `cases mvarId "not applicable to the given hypothesis"
    | some ctx =>
      if ctx.inductiveVal.nindices == 0 then
        inductionCasesOn mvarId majorFVarId givenNames useUnusedNames ctx
      else
        let s₁ ← generalizeIndices mvarId majorFVarId
        trace[Meta.Tactic.cases]! "after generalizeIndices\n{MessageData.ofGoal s₁.mvarId}"
        let s₂ ← inductionCasesOn s₁.mvarId s₁.fvarId givenNames useUnusedNames ctx
        let s₂ ← elimAuxIndices s₁ s₂
        unifyCasesEqs s₁.numEqs s₂
```

Users can add their own primitives

Challenges

Automation

Scalability

Usability

Language

AI

Funding

Automation

A "this is obvious" proof is unacceptable in Lean.

Lean fills the gaps in user provided constructions and proofs.

The overhead factor is currently over 20.

Dependent type theory (DTT) is a rich foundation, but hard to automate.

We have more than 20 years of experience in automated theorem proving at MSR.

How to lift successful techniques from first-order logic to DTT?

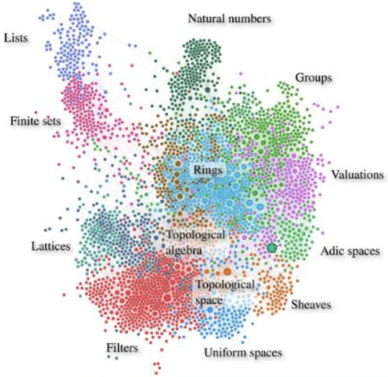
Is it possible to achieve overhead factor < 1?

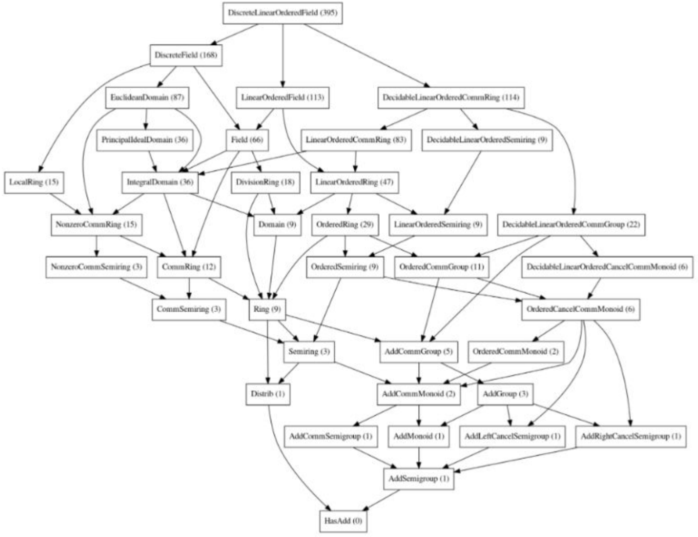
Scalability

Formal mathematical objects are massive for cutting edge math.

Many different techniques.

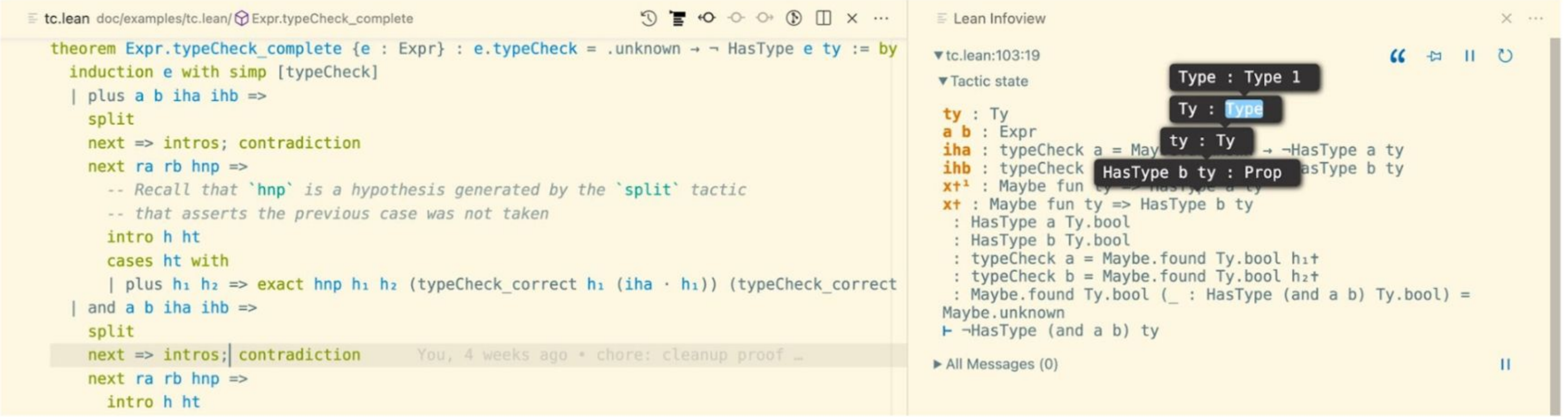
- New data-structures (e.g., Term Indexing for DTT)
- New algorithms (e.g., Tabled Type Class Resolution)
- Engineering (e.g., mmap)
- Lean Code generator (e.g., FBIP)

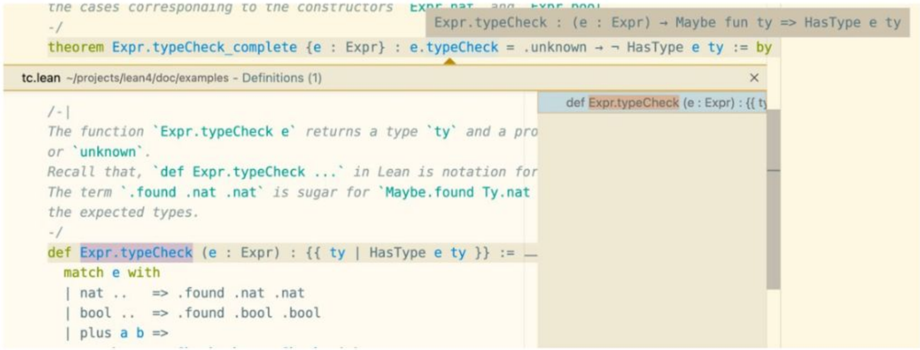


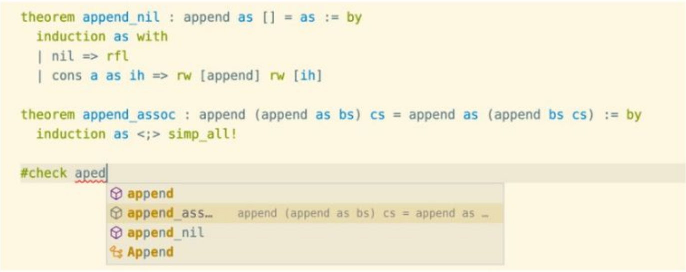


Usability

Several improvements and hundreds of commits. Joint work: MSR, KIT, CMU

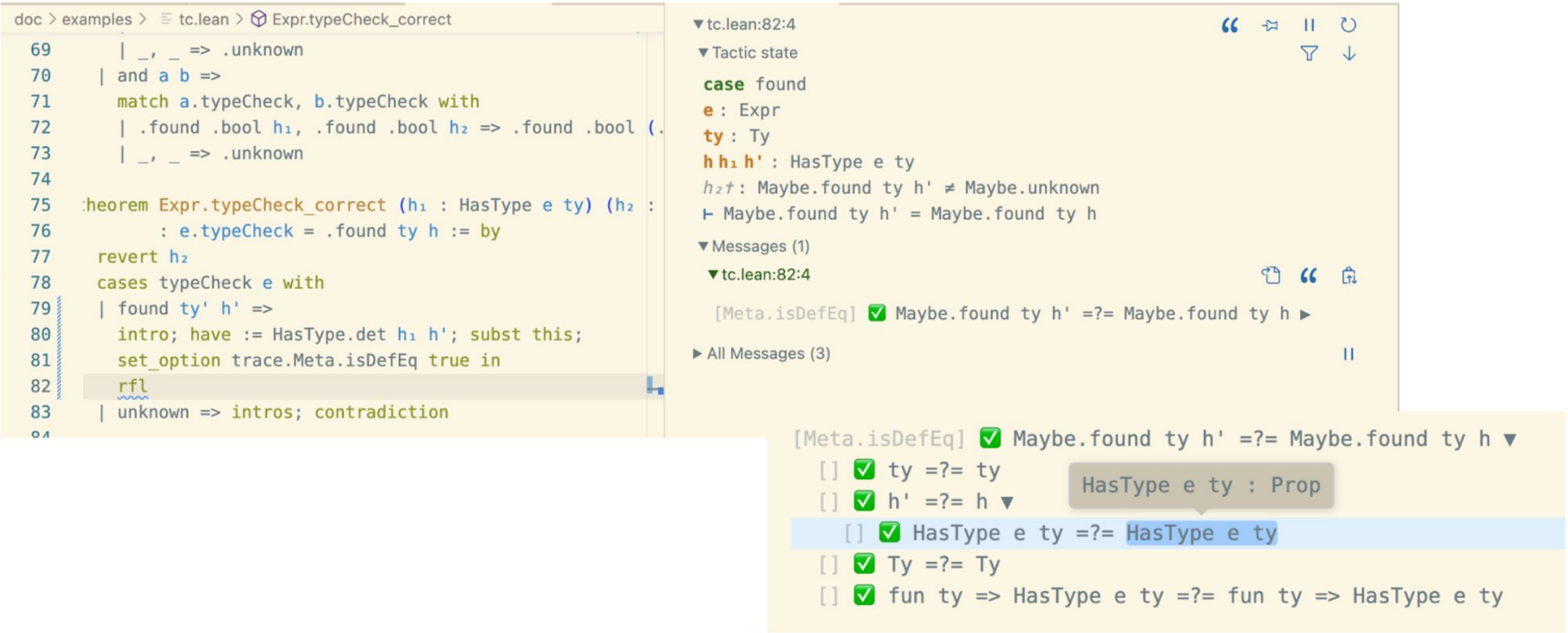




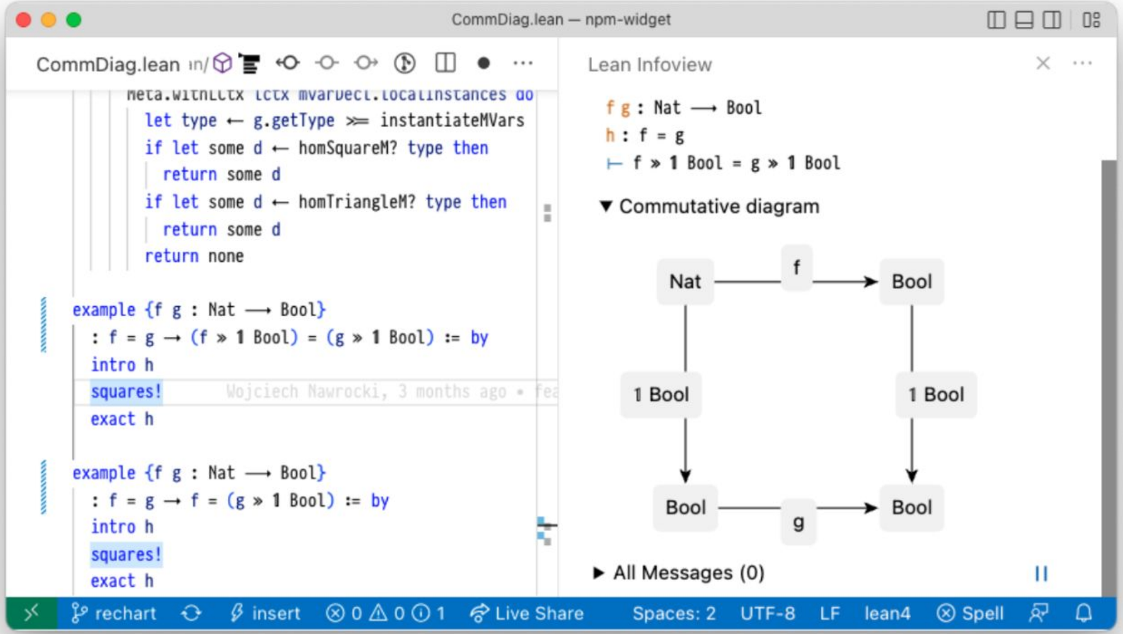


Usability

Collapsible trace messages



Usability


CommDiag.lean — npm-widget
CommDiag.lean
Lean Infoview
Meta.withLCtx lctx mvarDecl.localInstances do
let type ← g.getType >>= instantiateMVars
if let some d ← homSquareM? type then
return some d
if let some d ← homTriangleM? type then
return some d
return none
example {f g : Nat ⟶ Bool}
: f = g → (f ≫ 𝟙 Bool) = (g ≫ 𝟙 Bool) := by
intro h
squares!
Wojciech Nawrocki, 3 months ago
exact h
example {f g : Nat ⟶ Bool}
: f = g → f = (g ≫ 𝟙 Bool) := by
intro h
squares!
exact h
f g : Nat ⟶ Bool
h : f = g
⊢ f ≫ 𝟙 Bool = g ≫ 𝟙 Bool
Commutative diagram
Nat
f
Bool
𝟙 Bool
𝟙 Bool
Bool
g
Bool
All Messages (0)
rechart
insert
Live Share
Spaces: 2
UTF-8
LF
lean4
Spell

Language

The Lean language is rich and extensible.

- Coercions
- Overloaded notation
- Implicit arguments
- Type classes
- Hygienic macros
- Unification hints
- Embedded domain specific languages (DSLs)

There is no spec, we are learning it with the community.

Every new gadget must have a well-defined semantics.

Lean enables AI for math

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

PROBLEM 4

Adapted from IMO 1964 Problem 2

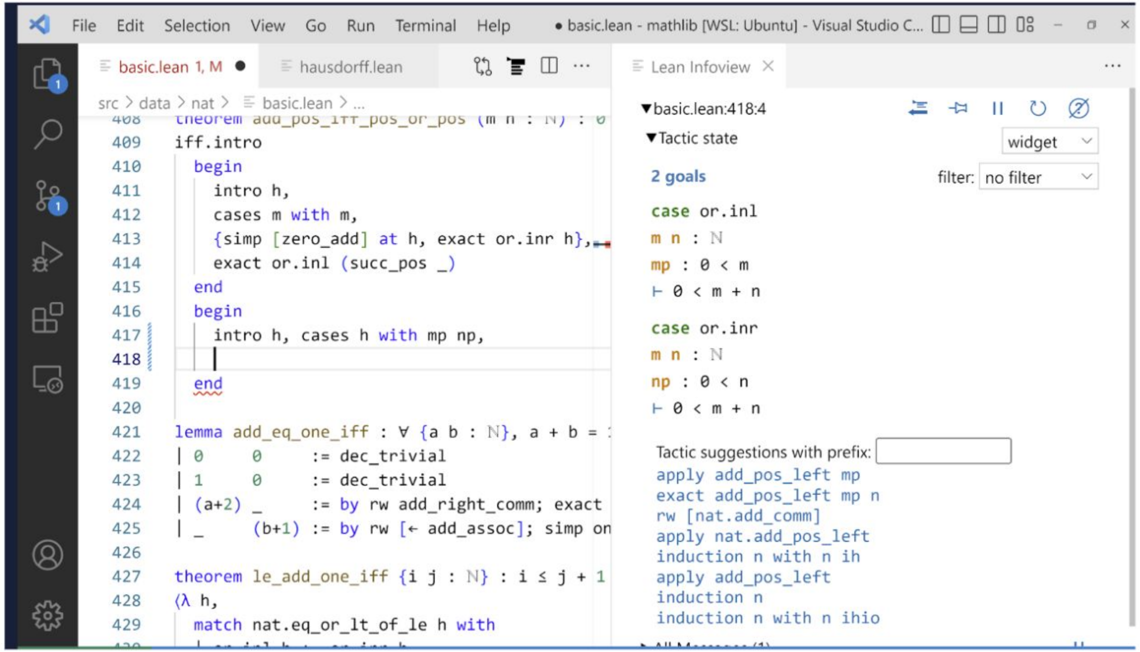
Suppose a, b, c are the sides of a triangle. Prove that $a^2(b+c-a)+b^2(c+a-b)+c^2(a+b-c) \leq 3abc$.

FORMAL | INFORMAL

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

Lean enables AI for math

Meta – HyperTree Proof Search for Neural Theorem Proving



Lean enables AI for math

Lean Chat by **Zhangir Azerbayev** and **Edward Ayers**
available at the VS Code marketplace

Lean enables AI for math

Lean Chat by **Zhangir Azerbayev** and **Edward Ayers**
available at the VS Code marketplace

If x and g are elements of the group G, prove that $|x| = |g^{-1}xg|$.

Lean enables AI for math

Lean Chat by **Zhangir Azerbayev** and **Edward Ayers**
available at the VS Code marketplace

If x and g are elements of the group G, prove that $|x| = |g^{-1}xg|$.

```
theorem  order_conjugate (G : Type*) [group G] (x g : G) :
  order x = order (g⁻¹ * x * g) :=
```

Lean enables AI for math

Lean Chat by **Zhangir Azerbayev** and **Edward Ayers** available at the VS Code marketplace

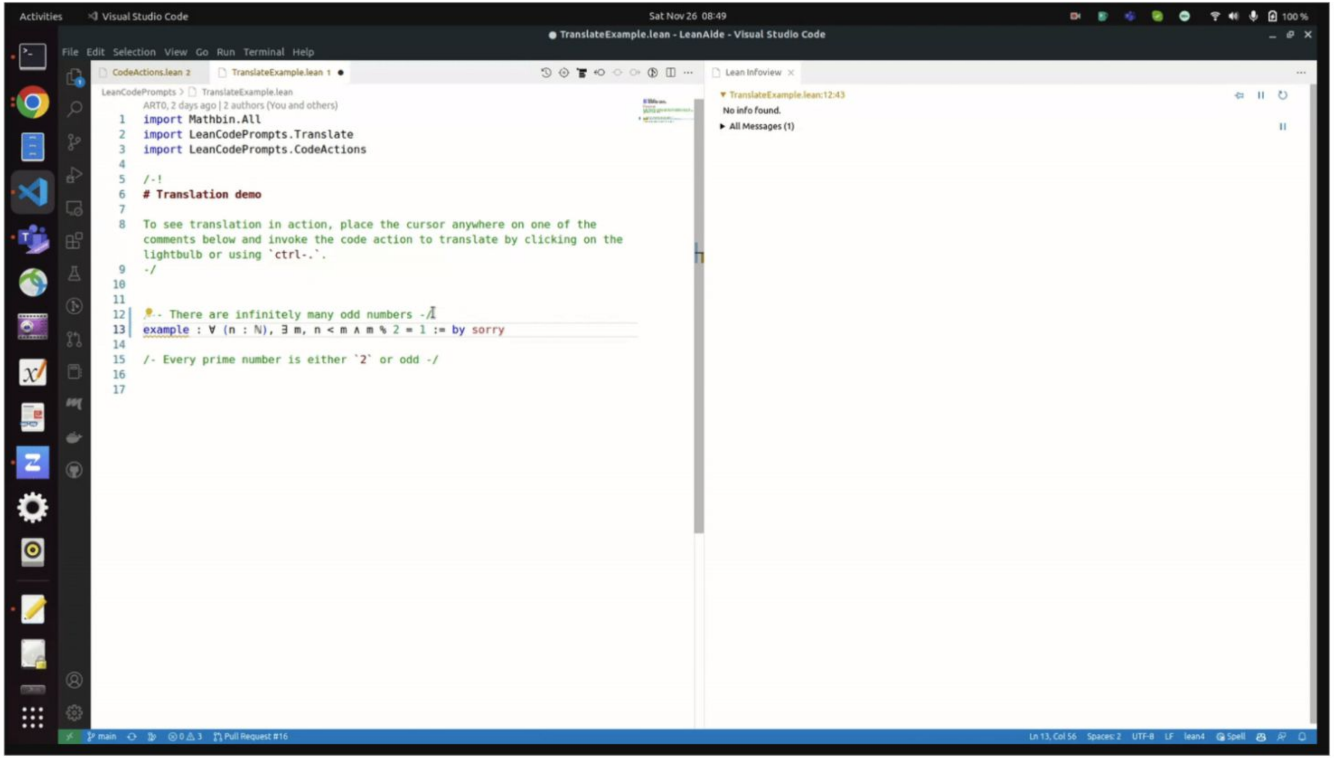
If x and g are elements of the group G, prove that $|x| = |g^{-1}xg|$.

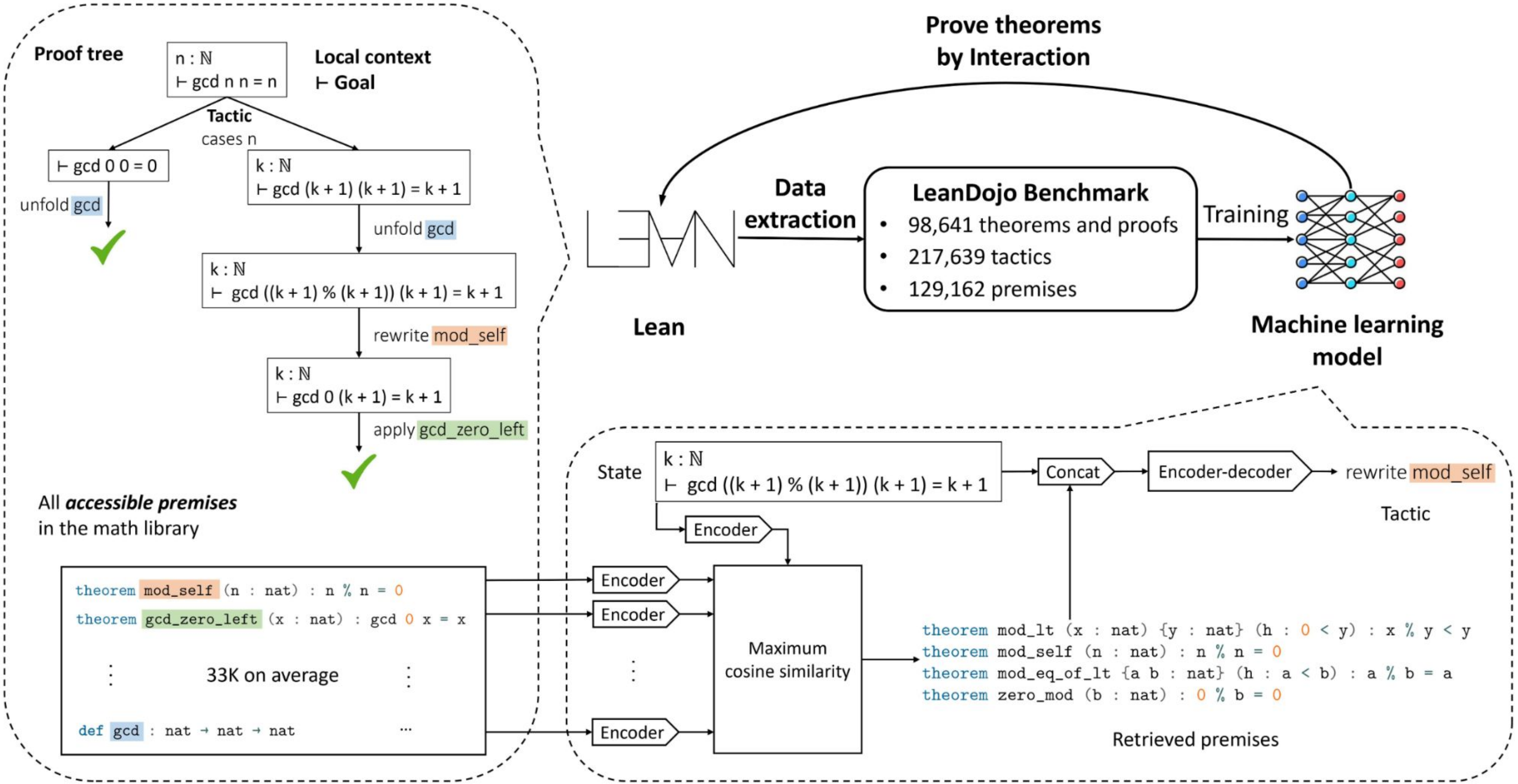
```
theorem  order_conjugate (G : Type*) [group G] (x g : G) :
  order x = order (g⁻¹ * x * g) :=
```

That's almost correct. Just replace `order` with `order_of`.

Lean enables AI for math

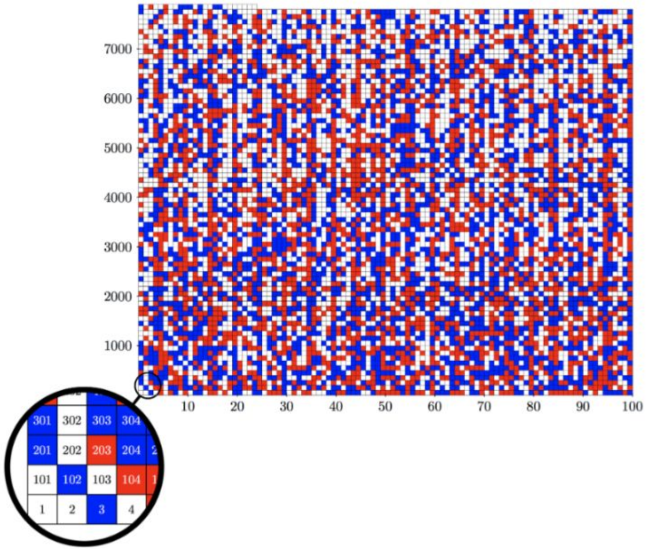
LeanAide by Ayush Agrawal, Siddhartha Gadgil, Navin Goyal, Anand Tadipatri




Proof tree
n : ℕ
⊢ gcd n n = n
Local context
⊢ Goal
Tactic
cases n
⊢ gcd 0 0 = 0
unfold gcd
k : ℕ
⊢ gcd (k + 1) (k + 1) = k + 1
unfold gcd
k : ℕ
⊢ gcd ((k + 1) % (k + 1)) (k + 1) = k + 1
rewrite mod_self
k : ℕ
⊢ gcd 0 (k + 1) = k + 1
apply gcd_zero_left
All *accessible premises* in the math library
theorem mod_self (n : nat) : n % n = 0
theorem gcd_zero_left (x : nat) : gcd 0 x = x
33K on average
def gcd : nat → nat → nat ...
Prove theorems by Interaction
Data extraction
LeanDojo Benchmark
98,641 theorems and proofs
217,639 tactics
129,162 premises
Training
Lean
Machine learning model
State
k : ℕ
⊢ gcd ((k + 1) % (k + 1)) (k + 1) = k + 1
Concat
Encoder-decoder
rewrite mod_self
Tactic
Encoder
Encoder
Encoder
Encoder
Maximum cosine similarity
theorem mod_lt (x : nat) {y : nat} (h : 0 < y) : x % y < y
theorem mod_self (n : nat) : n % n = 0
theorem mod_eq_of_lt {a b : nat} (h : a < b) : a % b = a
theorem zero_mod (b : nat) : 0 % b = 0
Retrieved premises

Beyond Large Language Models

Solving and Verifying the Boolean Pythagorean Triples problem via Cube-and-Conquer by Marijn J.H. Heule, Oliver Kullmann, and Victor Marek



Engineering

Yes, there is a lot of engineering.

Cloud build system.

Package manager (Mathlib is currently a mono-repo).

Documentation generators.

Continuous Integration (CI) for Lean and Mathlib.

Installation packages.

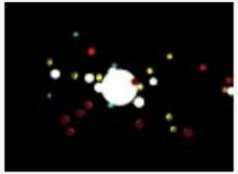
Diagnostic tools (essential when something goes wrong).

Community excitement

lean4 › Lean 4 as a scripting language in Houdini

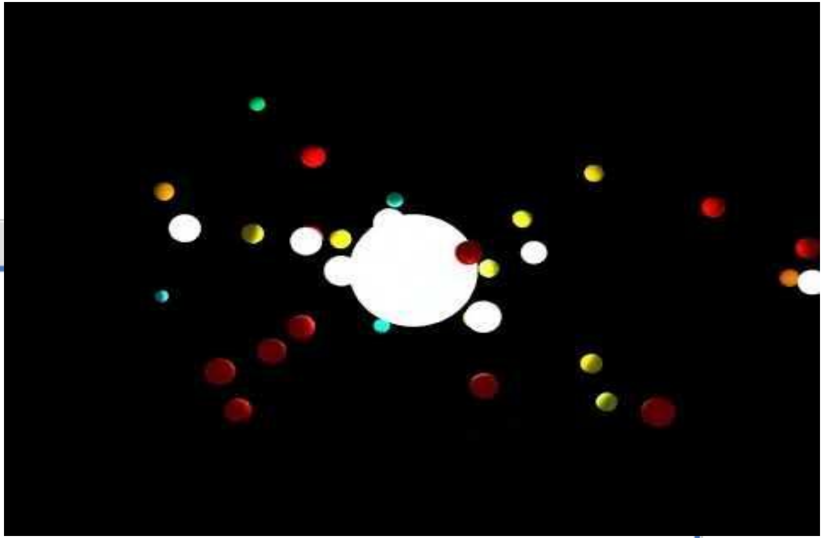
Tomas Skrivan EDITED

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



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

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



Auto refactoring / 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

LEAN http://leanprover.zulipchat.com

Focused Research Organization (FRO)

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

convergentresearch.org

The Lean FRO

Mission: address scalability, usability, and proof automation in Lean

~7 FTEs by end of year

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

lean-fro.org

Questions of Scale

“Can mathlib scale to 100 times its present size, with a community 100 times its present size and commits going in at 100 times the present rate? [...] Will the proofs be maintained afterwards [...]?”

– Joseph Myers on Lean Zulip

Conclusion

LEAN is an extensible theorem prover. http://leanprover.github.io

Decentralized collaboration.

The Mathlib community will change how mathematics is done and taught.

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 power.