Programs and Proofs

Anthony Wang (xy)

2026-01-04

Contents

-

Lean bio:

- Created in 2013 by Leo de Moura at Microsoft Research
- Lean 4 released in 2023, rewritten in Lean itself
- Named because it was supposed to be fast and minimal, not named after the drug
- A very modern programming language that enables building powerful abstractions

Cool Lean projects

Mathlib4

Mathlib4 (This is how everyone will be doing math in 20 years (maybe)) https://eric-wieser.github.io/mathlib-import-graph/ https://leanprover-community.github.io/mathlib_stats.html

Video player

- Lean rickroll (You can choose your level of verification and safety)

Rupert

https://www.youtube.com/watch?v=jDTPBdxmxKw

https://jcreedcmu.github.io/Noperthedron/blueprint/dep_graph_document.html

SciLean

- https://lecopivo.github.io/scientific-computing-lean/Working-with-Arrays/Tensor-Operations/#Scientific-Computing-in-Lean--Working-with-Arrays--Tensor-Operations--Simple-Neural-Network (Blazingly fast and no GC)

Equational theories

- https://teorth.github.io/equational_theories/ “Math at web scale”

Raytracer

https://github.com/kmill/lean4-raytracer

Functorio

- https://github.com/konne88/functorio (If it compiles, it’s most likely correct and bug-free)

LeanTex

https://github.com/kiranandcode/LeanTeX/

HouLean

https://github.com/lecopivo/HouLean

Analysis

https://teorth.github.io/analysis/sec21/ (useful for teaching, instant feedback)

Erdős 707

https://borisalexeev.com/pdf/erdos707.pdf (Maybe can solve the LLM hallucination problem, since LLMs suck at reasoning)

Compile-time video player

- Lean rickroll in VSCode (Lean's metalanguage is just Lean)

Insertion sort

```
variable [LE α] [DecidableLE α]
[Std.IsLinearOrder α] [BEq α]
[LawfulBEq α] (xs : List α)

@[grind] def insert (a : α)
  | [] => [a]
  | x :: xs =>
    if a ≤ x then a :: x :: xs
    else x :: insert a xs

@[grind] def insertionSort : List α →
List α
  | [] => []
  | x :: xs => insert x (insertionSort
xs)

@[grind] def Sorted : List α → Prop
  | [] | [_] => True
  | x :: x' :: xs => x ≤ x' ∧ Sorted
(x' :: xs)

theorem insertCorrect x : (Sorted xs →
Sorted (insert x xs)) ∧ (x :: xs).Perm
(insert x xs) := by
  induction xs with
  | nil => grind
  | cons _ t => cases t <;> grind

theorem insertionSortCorrect : Sorted
(insertionSort xs) ∧ xs.Perm
(insertionSort xs) := by
  induction xs with
  | nil => grind
  | cons h t => grind [insertCorrect
(insertionSort t) h]
```