Computational Logic in the Era of AI

Introduction

Sergey Mechtaev

mechtaev@pku.edu.cn

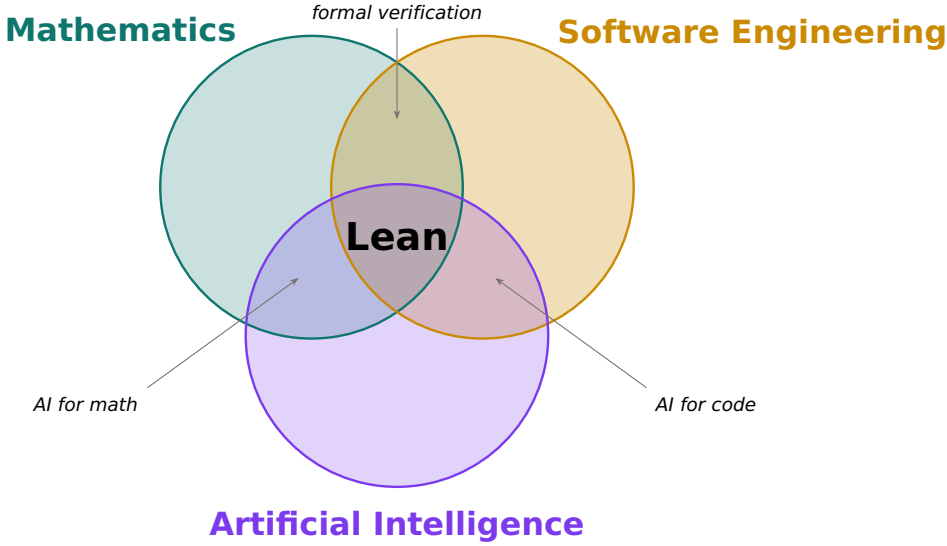
Term 1 • 26–27

Who is teaching, and where to ask

WeChat group
人工智能时代的计算逻辑 26–27

Sergey Mechtaev
Lecturer

Sijie Liang
Teaching assistant



One language for code, proofs, and AI.

Can you trust AI?

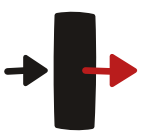
Hidden bugs

Hallucination

Bad reasoning

Overconfidence

Sheer volume



Black box

Lean’s foundation: the Curry–Howard correspondence

```
theorem S (p q r : Prop) :
  (p → q → r) → (p → q) → (p → r) :=
  fun f g a => f a (g a)
```

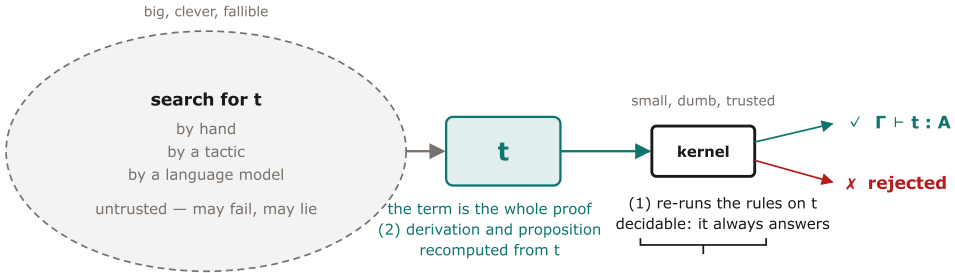
Theorem. For all propositions p, q, r:

$$(p \to q \to r) \to (p \to q) \to (p \to r)$$

Proof. $\lambda f.\lambda g.\lambda a.\ f\,a\,(g\,a)$. □

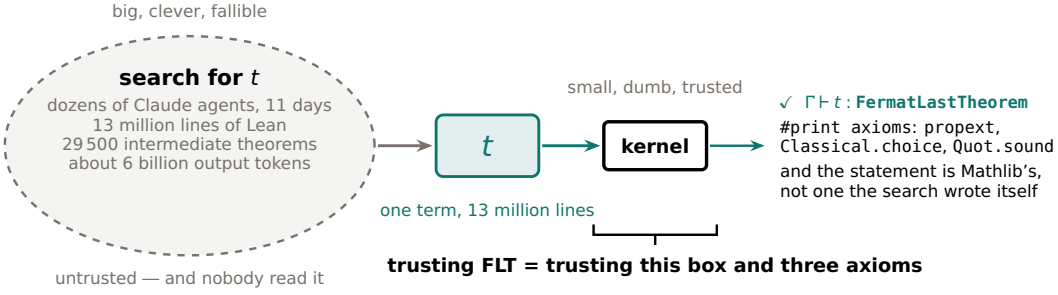
A proposition is a type; a proof is a term — so proof checking is type checking: $\Gamma \vdash t : A$.

The de Bruijn criterion: why a Lean proof is trusted



however t was found — the de Bruijn criterion. The checker does not care where t came from.

The criterion at scale: Fermat's Last Theorem (2026)



"... proves Fermat's Last Theorem with no assumptions other than the axioms of mathematics." — Kevin Buzzard, on the checked proof. anthropic.com/research/formalizing-fermats-last-theorem, 4 Sept 2026.

Small is not the same as correct

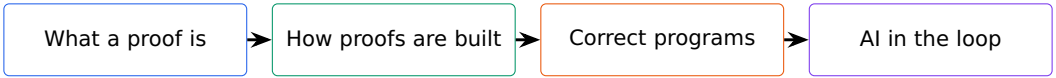
25 July 2026: an AI-assisted, `sorry`-free disproof of the Collatz conjecture — and the kernel accepted it.

```
structure  C where b : Bool
inductive  W where | mk (b : Bool)
inductive  T : Bool → Prop where | mk : T true
inductive  L (α : Type) (b : Bool) where | mk    -- α, b are phantom

-- pushed past the kernel with addDecl; the elaborator rejects it
inductive  E : W → Type where
  | mk (w : W) (l : L (E w) (w.1.1)) : E w       -- w.1.1 : a C out of a W
-- L's phantom parameters vanish from the generated auxiliary type,
-- so w.1.1 is never type-checked

theorem boom : False := nomatch (bad : T false)  -- bad : T false, from E
#print axioms boom                               -- no axioms
```

“This is an implementation bug, not a hole in Lean’s meta-theory.”

Leonardo de Moura, postmortem for #14576. Photo: David Monniaux, CC BY-SA 4.0.

Course content



From $\Gamma \vdash t : A$ to a program that ships with a machine-checked proof.

Why the course covers all of this

```
def bonus (x : Nat) : Nat :=
  if x % 2 == 0 then 2 * x else x
def score : List Nat → Nat
  | []      => 0
  | x :: xs => bonus x + score xs

theorem score_append (l : List Nat) (x : Nat) :
    score (l ++ [x]) = score l + bonus x := by
  induction l with
  | nil => simp [score]
  | cons y ys ih => simp [score, ih]; omega

def scoreInto (xs : List Nat) : StateM Nat Unit
    requires s => s = 0
    ensures _ s => s = score xs
  := do
    for x in xs invariant pref _ s => s = score pref do
      modify (· + bonus x)
  where finally
    | spec =>
      case vc1 => simp_all [score]
      case vc3 => simp_all [score_append]
```

The scoring rule
Every item scores its value, and even items count double; a list scores the total.

← a lemma, proved by induction

← a monad

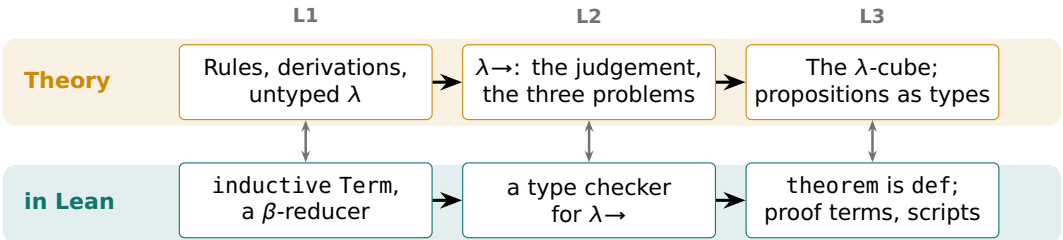
← the contract

← the loop invariant

← tactics

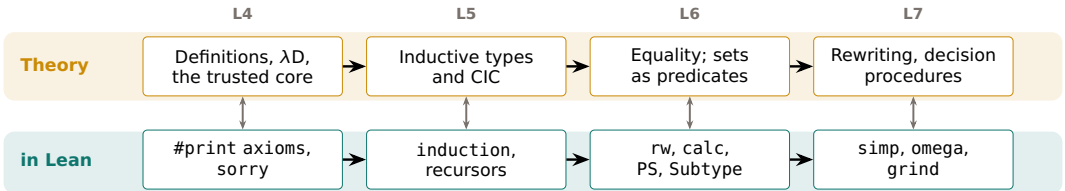
Phase I — What a proof is L1–L3

What exactly is $\Gamma \vdash t : A$, and why is checking it decidable while finding t is not?



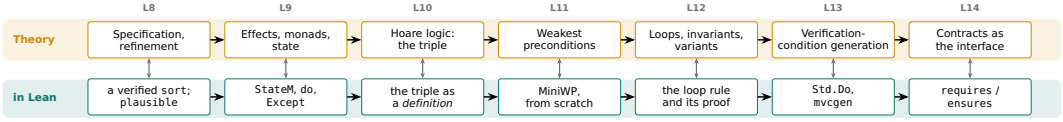
Phase II — How proofs are built L4–L7

Definitions, inductive types, equality, automation: the machinery between a judgement and a usable proof.



Phase III — Correct programs L8–L14

What is a specification, and how does a program come to satisfy one — pure, effectful, looping, and finally contract-annotated?



Phase IV — AI in the loop L15

Where does a generator help, where does the kernel filter, and where can a verified claim still be false?