WHO VERIFIES THE VERIFIER?

Lean, logical consistency, and the 2026 kernel soundness bugs

CS 846

Ende Jin

David R. Cheriton School of Computer Science

Slide is here



UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS

WHAT HAPPENED?

Aug 31 The Lean developers open #14977 *Meta issue: Lean Kernel Logic renovation*.

Quote

> *This summer's crop of implementation issues affecting the soundness of the kernel suggests that we should consider making changes to the kernel logic that make it easer to implement a bug-free kernel, that make verification of the kernel more feasible...*

WHAT HAPPENED?

Aug 31 The Lean developers open *#14977 Meta issue: Lean Kernel Logic renovation*.

Quote

> *This summer's crop of implementation issues affecting the soundness of the kernel suggests that we should consider making changes to the kernel logic that make it easer to implement a bug-free kernel, that make verification of the kernel more feasible...*

Is that a hole in Lean's metatheory, or not?

WHAT HAPPENED?

Aug 31 The Lean developers open #14977 *Meta issue: Lean Kernel Logic renovation*.

Quote

> *This summer's crop of implementation issues affecting the soundness of the kernel suggests that we should consider making changes to the kernel logic that make it easer to implement a bug-free kernel, that make verification of the kernel more feasible...*

Is that a hole in Lean's metatheory, or not?

What is Lean? **What supports Lean?**

WHAT IS LEAN?

- A **foundational**, **general-purpose**, **type-theoretic** theorem prover.
- A functional programming language and a proof assistant.
- Architecture: a large, untrusted *elaborator* produces certificates; a small *kernel* proof checks them.

Tactics, automation, AI $\Longrightarrow$ certificate $\Longrightarrow$ **kernel**: accept / reject

WHY FOUNDATIONAL?

Logic	Type theory
proposition P	type A
proof of P	term $t : A$
$P \to Q$	$A \to B$
$P \wedge Q$	$A \times B$
$P \vee Q$	$A + B$
$\bot$	`Nothing`

THE CURRY–HOWARD CORRESPONDENCE

A program that swaps a pair:

```
def pair_swap (α β : Type) :
    α × β → β × α :=
  fun ⟨a, b⟩ => ⟨b, a⟩
```

A proof that $\wedge$ commutes:

```
theorem and_swap (p q : Prop) :
    p ∧ q → q ∧ p :=
  fun ⟨hp, hq⟩ => ⟨hq, hp⟩
```

WHY FOUNDATIONAL?

Logic	Type theory
proposition P	type A
proof of P	term $t : A$
$P \to Q$	$A \to B$
$P \wedge Q$	$A \times B$
$P \vee Q$	$A + B$
$\bot$	`Nothing`
predicate $P\,x$	type family $P\,x$
$\forall x, P\,x$	$(x : A) \to P\,x$
$\exists x, P\,x$	$\{x : A \mathbin{//} P\,x\}$
$a = b$	path type

THE CURRY–HOWARD CORRESPONDENCE

A program that swaps a pair:

```
def pair_swap (α β : Type) :
    α × β → β × α :=
  fun ⟨a, b⟩ => ⟨b, a⟩
```

A proof that $\wedge$ commutes:

```
theorem and_swap (p q : Prop) :
    p ∧ q → q ∧ p :=
  fun ⟨hp, hq⟩ => ⟨hq, hp⟩
```

Proof checker is type checker.

WHY GENERAL-PURPOSE? INDUCTIVE TYPES & QUOTIENTS

- Inductive types (with their induction principles) and Quotient covers most conventional mathematics.

```
inductive Nat where
  | zero : Nat
  | succ : Nat → Nat

-- Mathlib: ℝ is a quotient of Cauchy sequences of ℚ
structure Real where ofCauchy ::
  cauchy : CauSeq.Completion.Cauchy (abs : ℚ → ℚ)

def Cauchy :=
  @Quotient (CauSeq _ abv) CauSeq.equiv
```

GENERAL PURPOSE, AND JUDGED BY MANY COMMUNITIES

Formal Verification Communities

- Multi-modal program verifiers
  Gladshtein et al., POPL 2026: verifiers whose soundness is proved in Lean.
  doi:10.1145/3776719
- Iris-Lean
  de Medeiros et al., 2026: ports the Iris concurrent separation logic framework from Rocq to Lean. arXiv:2609.24252

Math Community

- Terence Tao
  Formalized the proof of the Polynomial Freiman–Ruzsa conjecture in Lean, weeks after the paper.
- Kevin Buzzard
  Leads the project to formalize Fermat's Last Theorem in Lean.

The type theory community's view is *mixed*.

WHAT IS LOGICAL CONSISTENCY?

- A theory is **consistent** if there is no closed proof of `False`.
- Why it matters: `False` proves everything.

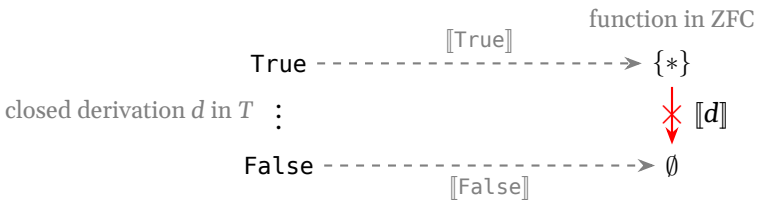
```
inductive False : Prop        -- no constructors

theorem anything (P : Prop) (h : False) : P :=
  False.elim h
```

- One kernel-accepted proof of `False` proves all statements.
- Conflating truth and lies.

HOW DO WE PROVE CONSISTENCY?

- We prove a theory T is consistent **relative to ZFC**.
- Build a **model** of T in ZFC:
  1. Each statement $A \mapsto$ a set $[\![A]\!]$.
  2. Each derivation from A to $B \mapsto$ a function $[\![A]\!] \to [\![B]\!]$.



No function $\{*\} \to \emptyset$ exists, so no closed derivation of `False` exists.

Lean has such a model, so: if ZFC is consistent, Lean is consistent.

Carneiro, *The Type Theory of Lean*, 2019, Corollary 6.8 (ZFC + n inaccessibles, every n).

WHO VERIFIES TYPE THEORY, THEORETICALLY?

- On a computer, consistency is not enough.
- We need an algorithm that checks whether a given proof is valid.
- Proof checking is type checking (Curry–Howard).

So we need **decidability of type checking**.

(The mental model of) Lean does **not** have it.

Carneiro, *The Type Theory of Lean*, 2019, §3.1: definitional equality is undecidable.

THE GAP

If Lean is consistent $\longrightarrow$ why are there kernel soundness bugs?

If type checking is undecidable $\longrightarrow$ what runs in every Lean executable?

THE GAP

If Lean is consistent $\longrightarrow$ why are there kernel soundness bugs?

If type checking is undecidable $\longrightarrow$ what runs in every Lean executable?

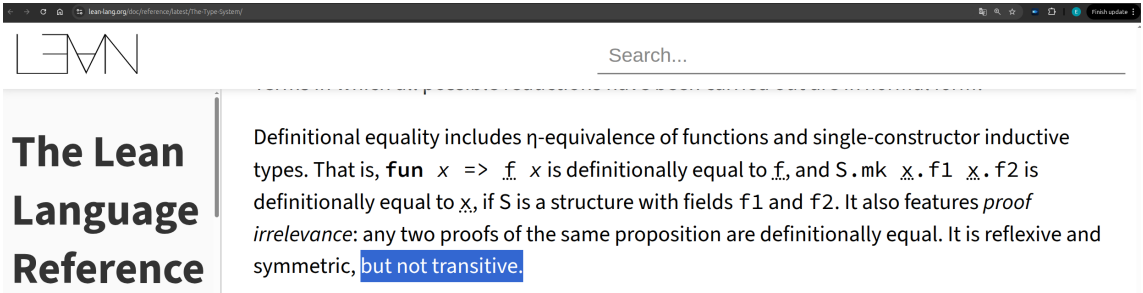
Both theorems are about **the mental model of Lean**.

	Mental model	Implementation
Equality checker[1]	transitive	**not** transitive
Type checking	undecidable	terminating
Consistency	proved (Cor. 6.8)	2026 kernel bugs

[1] The correct terminology for equality checker in metatheory is *judgemental equality* or *conversion checking*. Judgemental equality is to distinguish from *propositional equality*. We will not get into the details here.

THE GAP: TRANSITIVITY OF EQUALITY CHECKER

Is non-transitivity a bug? **No. It is a feature.**



Screenshot taken 9 November 2026

```lean
def True_mod_eq := Quot (α := True) (· = ·)
def img_true : True_mod_eq := Quot.mk (· = ·) trivial
def choose_img_true : True_mod_eq := Classical.choice ⟨img_true⟩
def mk42 (e : True_mod_eq) := Quot.lift (fun _ ⇒ 42) (fun _ _ _ ⇒ rfl) e

def a := mk42 img_true
def b := mk42 choose_img_true
def c := 42
#check (rfl : b = a) -- rfl is asking the internal equality checker to decide
#check (rfl : a = c)
#check (rfl : b = c) -- rfl fails here
#check ((Eq.trans (rfl : b = a) (rfl : a = c)) : b = c)
-- we have to manually write down more proof terms, like hints so Lean can accept

-- we can lift the above observation to programs
variable (R : Nat → Type)

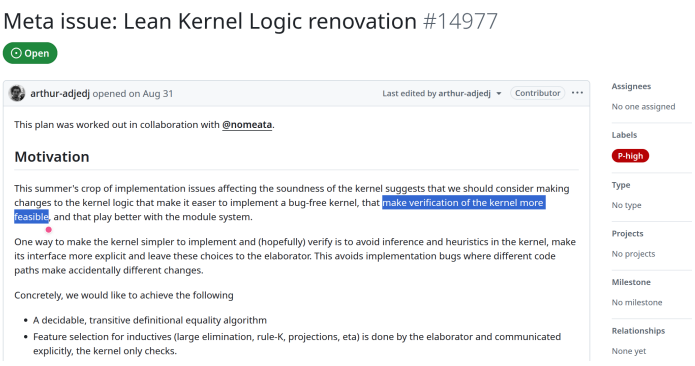
def f1 (x : R b) : R a := x      You, 19 seconds ago • Uncommitted changes
def f2 (x : R a) : R c := x
def f3 (x : R b) : R c := x -- this will fail
def f4 (x : R b) : R c := cast (congrArg R ((Eq.trans (rfl : b = a) (rfl : a = c)) : b = c)) x
-- again, more proof terms are needed to convince Lean
```

Tested on `lean4:v4.31.0`

THE GAP, THE FIX

How long has this non-transitivity been around? **At least since 2022.**

https://leanprover-community.github.io/archive/stream/113488-general/topic/subject.20reduction.html



Meta issue: Lean Kernel Logic renovation #14977

Open

arthur-adjedj opened on Aug 31 — Last edited by arthur-adjedj — Contributor

This plan was worked out in collaboration with @nomeata.

Motivation

This summer's crop of implementation issues affecting the soundness of the kernel suggests that we should consider making changes to the kernel logic that make it easer to implement a bug-free kernel, that make verification of the kernel more feasible, and that play better with the module system.

One way to make the kernel simpler to implement and (hopefully) verify is to avoid inference and heuristics in the kernel, make its interface more explicit and leave these choices to the elaborator. This avoids implementation bugs where different code paths make accidentally different changes.

Concretely, we would like to achieve the following

- A decidable, transitive definitional equality algorithm
- Feature selection for inductives (large elimination, rule-K, projections, eta) is done by the elaborator and communicated explicitly, the kernel only checks.

Assignees: No one assigned

Labels: P-high

Type: No type

Projects: No projects

Milestone: No milestone

Relationships: None yet

WHO VERIFIES LEAN, IN A VERIFIER?

Work	Proves	Assumes
Rocq Rocq Correct! / **MetaRocq** POPL 2020; JACM 2025 in Rocq	Rocq kernel checker sound (2020) and complete (2025) w.r.t. PCUIC	Strong normalization of PCUIC (axiom)
McTT ICFP 2025 in Rocq	Core MLTT ($\mathbb{N}$, Π, universes): normalization, consistency; checker sound and complete	Nothing (in Rocq, which additionally has impredicative `Prop`)
Lean4Lean TYPES 2025 in Lean	No consistency	Nothing
Con-Leche Lean FRO, 2026 in Lean **not peer-reviewed**	Never accepts a proof of `False`; set-theoretic model No transitivity Not reflecting the real implementation	ZF + ω universes (inaccessibles); Lean runtime `Nat`

Gödel's second incompleteness theorem: a consistent, expressive system cannot prove its own consistency.
Gap between host and object: Blue

WHO VERIFIES LEAN, IN A VERIFIER??!

GAP

Work	Object	Gap to the real prover
Rocq Rocq Correct! / MetaRocq	PCUIC, close to the Rocq kernel	No modules, template polymorphism, or η *Performance: no published benchmarks*
McTT	Core MLTT: $\mathbb{N}$, Π, universes	No general inductive types, no `Prop` *Performance: N/A*
Lean4Lean	Full Lean 4 kernel; checks Mathlib	Small *Performance: ∼30% slower than C++*
Con-Leche	Full Lean 4 kernel; production ready	Empirically small, formally unknown Incompatible internal representation *Performance: faster than C++ as standalone checker* *Performance: unknown as an interactive kernel*

WHO VERIFIES LEAN, IN A VERIFIER??!

TRUSTWORTHINESS



Talia Ringer (UIUC) on Con-Leche

The de Bruijn criterion

"A Mathematical Assistant satisfying the possibility of independent checking by a small program is said to satisfy the de Bruijn criterion." Barendregt & Wiedijk, 2005

- ✓ **Separation**: "Separate the proof checker ('simple') from the proof engine ('powerful')." Geuvers, 2019, p. 9
- **Independent checking**: "[Certificates] should be public and written in such a formalism that a reasonably simple proof-checker can verify them." Barendregt, 1997
- **Small kernel**: "Their programs must be small, so small that a human can (easily) verify the code by hand." Barendregt, 2007

WHO VERIFIES LEAN, ENGINEERING-WISE? INDEPENDENT CHECKER

- Lean Kernel Arena: 36 checkers, e.g. official (C++), nanoda (Rust), Lean4Lean (Lean), Con-Leche (Lean).

Does Lean achieve independent checking? **Counter-arguments:**

1. No official specification: every checker imitates the official checker.
2. Con-Leche accepts, official rejects: which one is right?
3. In practice the official checker is the reference. What if its mental model is wrong?

WHO VERIFIES LEAN, ENGINEERING-WISE?

The dependent type theory we actually use:

1. The Curry–Howard core
2. Proof-irrelevant `Prop`
3. Cumulative universe hierarchy
4. Inductive types, indexed inductive types, …

5*. Quotient types

6*. Extensionality: function, propositional, …

*Possible to leave out of the kernel, e.g. as schemas.

SMALL SIMPLE KERNEL

Kernel size:

- Lean (`v4.31.0`): ∼8k lines of C++
- Rocq (`V9.1.0`): ∼42k lines of OCaml

Strong normalization

hard to prove; necessary for decidable type checking; not simple to check

NOT REFERENCE

BECAUSE ALL THE EARLIER REFERENCES HAVE BEEN HYPERLINKED

- https://leodemoura.github.io/blog/2026-8-1-postmortem-for-kernel-soundness-bug-14576/
  Postmortem of a kernel soundness bug in nested inductive types: how it was reported, fixed, and hardened against.
- https://discuss.ocaml.org/t/why-lean-4-replaced-ocaml-as-my-primary-language/17109/5
  Performance of Lean.
- https://mathoverflow.net/questions/513742/are-we-stuck-with-lean
  Asks whether recent soundness bugs make a stronger-guarantee alternative, such as Metamath Zero, worth backing.
- https://mathstodon.xyz/@jonmsterling/116246695464035143
- https://proofassistants.stackexchange.com/questions/1183/how-much-of-trouble-is-leans-failure-of-normalization-given-that-logical-consi
  Asks how serious it is that impredicative `Prop` breaks normalization, given consistency and canonicity still hold.
- https://leanprover.zulipchat.com/#narrow/channel/113488-general/topic/What.20is.20the.20subject.20reduction.20debate.3F/with/274334065
- https://leanprover-community.github.io/archive/stream/113488-general/topic/subject.20reduction.html
- https://leodemoura.github.io/blog/2026-3-16-who-watches-the-provers/
  Argues trust should come from several independent small kernels, in different languages, that must all accept a proof.
- https://zhuanlan.zhihu.com/p/2084655794042283523
- https://x.com/TaliaRinger/status/2098492831354531851
- https://terrytao.wordpress.com/2026/10/09/what-mathematicians-should-know-about-the-lean-theorem-proverquestions-of-reliability-and-ai/
  Hales: a Lean proof is only as reliable as its kernel, statement, and type theory; calls for multiple and verified kernels.
- https://www.lesswrong.com/posts/zHhGa3iQBQcdSifFL/a-critical-look-at-lean4-1
  Argues Lean 4 should not be fully trusted: no known consistency proof relative to ZFC, nested inductives, repeated kernel bugs.