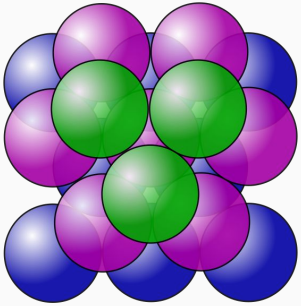
On the Origin of False Proofs by Means of Bugs in Proof Assistants

Yannick Forster

Cambium Team, Inria Paris

Séminaire automath, IHP, 16 October 2026

Kepler conjecture



Source: Wikipedia

1611 Conjecture by Johannes Kepler

1998 Proof by Tom Hales: 300 pages, 40,000 lines of code

1999 12 reviewers, 99% certain

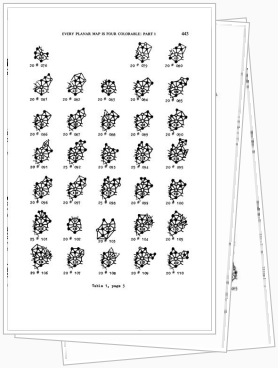
1999–02 Continued reviewing, still 99% certain

2005 Published in Annals of Mathematics, "without complete certification from the referees"

2003–15 Formal proof project in Isabelle and HOL Light: "Flyspeck"

2017 Published formal proof in Forum of Mathematics

Four colour theorem



Appel & Haken's unavoidable set, on microfiche

1852 Conjecture by Francis Guthrie

1879 Initial proof by Alfred Kempe

1890 Percy Heawood finds a mistake (5 colour theorem)

1976 Proof by Kenneth Appel and Wolfgang Haken

Proof idea: find an "unavoidable", "reducible" set of configurations

Reducibility: Checked by computer program, 15k lines on punch cards, over 1200 hours of computer time

Unavoidability: 400 pages of microfiche, checked manually by Appel, Haken, and Haken's teenage daughter Dorothea Blostein

1981 Mistakes found by Master's student, but fixed

2004 Proof in Rocq by Georges Gonthier with Benjamin Werner

BusyBeaver(5)



quantamagazine.org

lemonde.fr

1962 Tibor Radó introduces "The busy beaver game":
BB(n) is the maximal number of steps a Turing machine with n states can take
BB(15) decides a Collatz-like conjecture of Erdős (Stérin, Woods),
BB(643) is independent of ZFC (Ridenour)

1966 Allen Brady discovers 4 state machine taking 107 steps

1974 Brady proves BB(4) = 107
But there are 17 trillion possible 5 state Turing machines...

1989 Heiner Marxen and Jürgen Buntrock find 5 state machine taking 47,176,870 steps

2020 Scott Aaronson conjectures BB(5) = 47,176,870

2021 Busy beaver challenge started by Tristan Stérin

BusyBeaver(5)



quantamagazine.org

lemonde.fr

2021 Busy beaver challenge started by Tristan Stérin

2023 Only few "holdouts" left – but those are the hardest machines

July 2023 Maja Kadziolka: "Hi there! I'm interested in formalizing the algorithms used to decide halting in a proof assistant such as Coq. Has anything like this been attempted before?", starts BusyCoq project

May 2024 Anonymous contributor `mxdys` publishes full Coq proof, using BusyCoq as a library, solving all remaining holdouts

From Tristan Stérin <tristan@prgm.dev>

To Yannick Forster

Cc Théo Zimmermann

5/31/24, 10:51 AM

Subject **Coq expertise for bbchallenge.org**

Hello Yannick,

Théo (in CC) gave me your contact as I was discussing with him the need for Coq expertise for a massively collaborative research project I created 2 years ago, bbchallenge.org.

The goal of bbchallenge is to collaboratively prove the conjecture on the 5th busy beaver value, "BB(5) = 47,176,870", open since the 1980s, more details are given on https://bbchallenge.org/story.

Very recently, and to our biggest surprise, a new contributor, mxdys (also known as ccz181078), has formalised the proof of "BB(5) = 47,176,870" in Coq, using (and sometimes improving) many other contributions made by other bbchallenge collaborators: https://github.com/ccz181078/Coq-BB5. The theorem compiles in Coq in about 10 hours and it only uses one assumption, an axiom from the standard library, `functional_extensionality_dep`.

As a maintainer of the project with little Coq experience, I am seeking for independent Coq experts that could review the Coq theorem statement (not the full proof, for which the bbchallenge community knows the used techniques) and agree that it indeed corresponds to "BB(5) = 47,176,870". Collaborators of bbchallenge with more Coq experience than me have said it would only boil down to auditing the ~160 lines at the beginning of TM.v that define Turing Machines. Note that we don't believe that we are in an adversarial scenario as there is no prize of financial incentive for the proof and the contributor mxdys is trusted by the community.

Would you be interested in helping us with this audit and give your expert opinion on Coq-BB5 theorem statement?

I would like to mention two related elements:

- Quanta magazine is preparing a story on this collaborative proof of BB(5) and their reporter, Ben Brubaker, would love to interact with the Coq experts that will look at Coq-BB5. We are aiming for releasing the Quanta article at the end of June, which means that it would be amazing to collect expert opinions before then.

- Apart from the Coq proof, we are working on an arxiv+conference+journal paper to explain the result and the used techniques to the wider scientific community and of course, we would invite all Coq reviewers to be co-authors, as it is crucial to us to benefit from your expertise for this article (but we are committed to do the vast majority of the writing work and would not incur much additional work to you).

I'd be more than happy to chat about all this with you in a visio-call if your are interested :)

Thank you very much in advance,

Sorry for the long email!

Dr Tristan Stérin
Co-founder & CRO @ prgm.dev
m: 06 260 15 260
e: tristan@prgm.dev

prgm

PRGM DEV
SAS au capital de 999,99 €

Coq-BB5

Public

Watch 8 Fork 12 Star 329

main Go to file t Code

ccz181078 Merge pull request #19 from tcosm... b295161 · 13 hours ago 305 Commits

BusyCoq	update READMEs, delete root make...	10 months ago
CoqBB5	Decider ID uniformization	2 days ago
.gitignore	ignore archive	8 months ago
README.md	TYPO: proves theoremS	7 months ago

README

Coq-BB5

Coq-BB5 (author: mxdys) proves theorems in Coq (v8.20.1) about Busy Beaver values, including the following results:

Original results:

- `BB(5) = 47,176,870`, see CoqBB5/BB5/
- `BB(2,4) = 3,932,964`, see CoqBB5/BB2x4/

About

No description, website, or topics provided.

Readme
Activity
329 stars
8 watching
12 forks
Report repository

Releases 1

Coq-BB5 release v1.0.0 Latest
12 hours ago

Packages

No packages published

Contributors 5





Common Criteria: the international standard for security certification of IT products.
ANSSI's 2021 checklist: how an evaluator should trust a Coq proof.

BusyBeaver(5)



quantamagazine.org

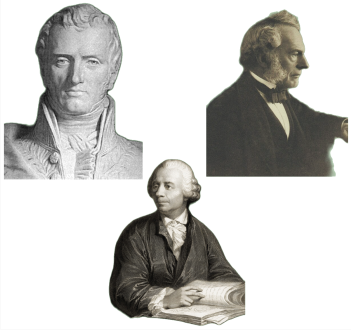
lemonde.fr

May 2024 Anonymous contributor `mxdys` publishes full Coq proof, using BusyCoq as a library, solving all remaining holdouts

2024 I was asked to review the Coq proof by Tristan – took me about 10 minutes to verify that the statement is correct, did not have to check the proof

2026 Paper at STOC 2026 by The bbchallenge Collaboration, Justin Blanchard, Daniel Briggs, Konrad Deka, Nathan Fenner, Yannick Forster, Georgi Georgiev (Skelet), Matthew L. House, Rachel Hunter, Iijil, Maja Kadziolka, Pavel Kropitz, Shawn Ligocki, mxdys, Mateusz Naściszewski, savask, Tristan Stérin, Chris Xu, Jason Yuen, Théo Zimmermann

Euler and Navier–Stokes



lemonde.fr

1757 Leonhard Euler publishes equations describing behavior of fluids

1822–45 Navier and Stokes work on generalisation of viscosity and thermal conductivity

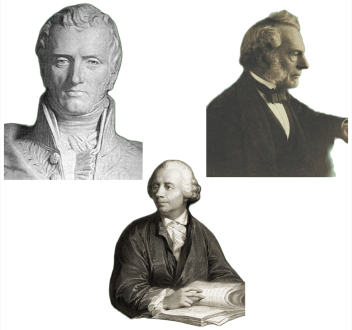
1934 Precise open problem stated by Jean Leray

2000 Clay institute determines Navier–Stokes as one out of 7 Millennium Prize Problems

2021 Córdoba and Martínez-Zoroa start program to find solution through "rough forcing"

2026 Buckmaster and Alpöge publish forced Euler blowup, based on smooth forcing.

Euler and Navier–Stokes



lemonde.fr

2026 Buckmaster and Alpöge publish forced Euler blowup, based on smooth forcing.

"We believe we also have blowup for hypo-dissipative Navier-Stokes. We are not releasing that paper today: unlike the above, the Lean verification has not yet finished. We do not yet have anything resembling a presentable writeup. I mention it because it is suggestive of a path to unforced Euler."

2026 OpenAI announces a solution to Navier–Stokes via unforced Euler blowup, with AI-generated 100 page pre-print and proof verified in Lean

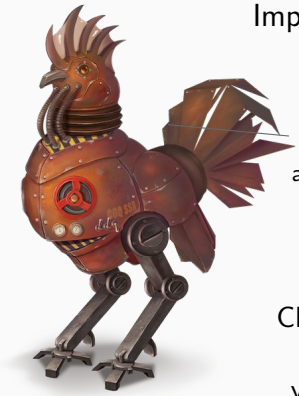
Oct 2026 Bastounis, Circelli, Hansen: Intermediate lemmas in pdf and Lean disagree – overall the right theorem, but a different proof was formalised

Poll by show of hands: In your opinion, does this mean the theorem is proved?

Bugs in proof assistants

$$\bot \to A$$

What do we need to trust?



Implementation / Small Core

Around 20.000 lines of code

around 1 critical bug per year

always found by developer

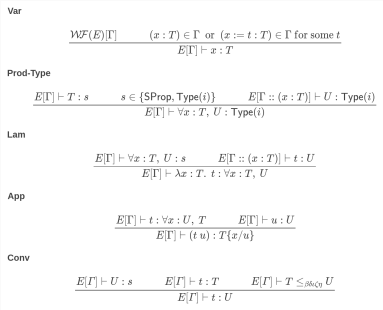
Changing the trusted core requires a PhD in type theory and years of development experience

Underlying theory

no problem in 40 years

image from Ilya Sergey's lecture notes, courtesy of Lilia Anisimova

Foundations matter



- In practice, mathematics is carried out in a typed system. There are numbers, functions, algebraic structures, and they are not the same thing.
- But set theory allows asking non-sensical questions "Some time ago, the question was raised (perhaps by Peter Freyd), on the categories discussion list, whether a finite simple group could be a zero of the Riemann zeta function. I believe someone checked that, with the usual set-theoretic codings of such entities, the answer is negative. Whew!" – Andreas Blass
- Two options arise: Higher-order logic or dependent type theory.
- Mild evidence that type theory yields more elegant mathematics formalisation.
- Open research question: How much dependent types are needed for maths and program verification?
- Fundamentally, type theory comes with a built-in programming language.

termination checker accepts infinite loop #20413

New issue

Closed #20415

ccz181078 opened on Mar 28

Description of the problem

`F 1 0 f = F 0 1 (fun x=>F x 0 f) = S (F 1 0 f)` doesn't terminate, but the definition of F is accepted. This can be used to prove False.

Small Rocq / Coq file to reproduce the bug

```
Fixpoint F(sz c:nat)(f:nat->nat):nat :=
match sz with
| S sz0 =>
    F sz0 (S c) (fun sz' => F sz' 0 f)
| O => S (f c)
end.

Lemma F_S sz0 c f:
  F (S sz0) c f = F sz0 (S c) (fun sz' => F sz' 0 f).
Proof.
  reflexivity.
Qed.

Lemma F_O c f:
  F O c f = S (f c).
Proof.
  reflexivity.
Qed.
```

Assignees

No one assigned

Labels

kind: bug kind: inconsistency

Projects

No projects

Milestone

9.0.1

No due date

Relationships

None yet

Development

Code with agent mode

Guard checker: don't forget to check non-structural arguments of fix

rocq-prover/rocq

Sound reasoning about programming languages

- `let rec f x = 1 + f x` is valid OCaml
- Also $\forall n \in \mathbb{N}.\ n \neq n + 1$ is a theorem
- Consequently, we have to forbid the definition of `f` in Rocq
- Chosen approach: Every program has to terminate
- The termination checker is the single biggest source of bugs in Rocq
- Contrast Lean: Everything is elaborated away. Pro: much safer. Con: No computation in proof assistant.



Tristan Sterin ⚡HALT 2/19/26, 8:51 AM

ok with all this proving power

what do we prove 

Enjoy your travel!

We can chat when you're back



Tristan Sterin ⚡HALT 2/19/26, 5:04 PM

we could task it to find critical bugs in the guard checker of the rocq kernel

since ppl said its where must bugs must be

I would most likely need your help to design that well though

- **Guard checker soundness bug: cross-calls in nested mutual fixpoints** kind: bug kind: inconsistency part: fixpoints
  #21682 · by tcosmo was closed 3 weeks ago · 9.2.0 — 1 — 4
- **Guard checker soundness bug: higher-order recursive call through fixpoint** kind: bug kind: inconsistency part: fixpoints
  #21683 · by tcosmo was closed 3 weeks ago · 9.2.0 — 1 — 6
- **Module Alias := M through multi-step alias chain corrupts delta-resolver, allowing proof of False** kind: bug kind: inconsistency
  #21685 · by tcosmo was closed 3 weeks ago · 9.2.0 — 1 — 2
- **Missing stack conversion for caseinvert** kind: bug kind: inconsistency part: kernel
  #21690 · by SkySkimmer was closed 3 weeks ago · 9.2.0 — 1 — 1
- **Incorrect discharge of squashing info** kind: bug kind: inconsistency part: kernel part: sections part: sort polymorphism
  #21694 · by SkySkimmer was closed 3 weeks ago · 9.2.0 — 1 — 1
- **Guard checker unsoundness: argument-less recursive calls don't restrict uniform arguments computation** kind: bug kind: inconsistency part: fixpoints
  #21701 · by yannl35133 was closed 3 weeks ago · 9.2.0 — 1
- **Incorrect `with Definition` universe constraint handling** kind: inconsistency part: modules part: universes
  #21702 · by SkySkimmer was closed 3 weeks ago · 9.2.0 — 1 — 4
- **Proof of False with universe polymorphism, VM/native and Register Inline** kind: inconsistency part: universes part: VM
  #21736 · by SkySkimmer was closed 2 weeks ago · 9.2.0 — 1
- **Modules forget elimination constraints** kind: bug kind: inconsistency part: modules part: sort polymorphism
  #21750 · by yannl35133 was closed 2 weeks ago · 9.2.0 — 1 — 1
- **Inconsistency due to uniform parameters in the guard condition** kind: bug kind: inconsistency part: fixpoints
  #21797 · by ppedrot was closed 5 days ago · 9.2.0 — 1 — 3



Tristan Stérin
Computer Scientist

In search of falsehood

March 5th 2026

Following up on the previous post where we used Opus 4.6 to prove things in Rocq and Lean4, we turned the tables: what if we asked it to *break* things instead? With the help of expert tips, we pointed Opus 4.6 at the kernels of Rocq and Lean4 (and some of Lean's non-official kernels) and asked it to find soundness bugs – bugs that allow deriving a proof of `False`.

A proof of `False` is arguably the worst possible bug in a proof assistant: from `False` you can prove anything, and the guarantees provided by formal verification vanish entirely.

The results:

	Proofs of `False`	Other bugs
Rocq (official)	1, 2, 3, 4, 5, 6, 7	8, 9, 10
Lean4 (official)	0	1, 2, 3, 4
nanoda (Lean, non-official)	1, 2	0
lean4lean (Lean, non-official)	1	0

Mirimanoff's paradox in type theory

- **Mirimanoff (1917)**: The collection of well-founded sets (no infinite chain $x_0 \ni x_1 \ni \cdots$) is not a set. Assume W would be. Then W is well-founded, since any chain from W enters a well-founded set and stops. Hence $W \in W$, which gives the chain $W \ni W \ni W \ni \cdots$: W is not well-founded. Contradiction.
- **Girard (1972)**: the same with ordinals (Burali-Forti) in System U: `Type : Type` lets one build the type of all ordinals, an ordinal larger than itself.
- **Coquand (1992)**: instead use trees branching over universe with `Type : Type`, closer to Mirimanoff.
- **Hurkens (1995)**: well-foundedness defined without inductive types, only two impredicative universes.
- Rocq: expressive universes, good for user, bad for soundness.
  Lean: limited universes, code duplication for user, good for soundness.
  Algebraic universes in Rocq coming up (Sozeau et al.).



Leonardo de Moura

Who Watches the Provers?

2026-03-16

In 2026, a skilled researcher using Claude (Opus 4.6), with real-time support from the Rocq development team, found seven kernel bugs in Rocq, a widely used proof assistant. Seven independent paths to proving something false.

Rocq is a serious system with decades of foundational work. But even Rocq is vulnerable when AI applies sustained adversarial pressure to a single implementation. The question is not whether AI will find the bugs, but whether the architecture is built to survive them.

I started Z3, the most widely used SMT solver in the world, and bugs that produce incorrect results are still regularly found by ordinary users after nearly two decades of development. Z3 does not have independent kernels, and its trusted base is large.

The problem compounds: many verification tools (e.g., Dafny, F*, and Verus) are built on top of Z3, and bugs arise not only in Z3 itself but also in these tools when they generate the verification conditions they send to Z3. Each layer in the stack is another place where correctness can silently break. This is the normal state of affairs for verification tools.

I built Lean not to have this problem.

Collatz

A Lean formalization of the Collatz map and its global behavior.

The mathematical statement is fixed in `Collatz/Definitions.lean` using Lean's standard natural numbers. The remaining modules use Mathlib for iteration, parity, and descent arguments:

- `Collatz/Iteration.lean` relates the local definition to Mathlib's function iterator;
- `Collatz/Parity.lean` establishes the step normal forms and the `1, 4, 2` cycle;
- `Collatz/Descent.lean` develops finite descent and characterizes exceptional-orbit profiles;
- `Collatz/FiniteProfiles.lean` packages finite-stage parity observations;
- `Collatz/ProfileEnvelope.lean` builds the stable dependent envelope;
- `Collatz/Limit.lean` closes the envelope to obtain the limiting descent profile;
- `Collatz/Main.lean` exports the result.

Result

The limiting profile yields a positive orbit which never reaches `1`:

```
Collatz.exists_nonterminating_orbit : ∃ n, Collatz.Diverges n
```

Consequently:

```
Collatz.not_conjecture : ¬ Collatz.Conjecture
```

The argument is witness-free: it establishes existence of an exceptional orbit without selecting a concrete natural number.

Finding bugs with AI

- In summer 2026, Daniel Selsam at OpenAI finds many more bugs in both Rocq and Lean with AI
- Russel-style issues: $P \leftrightarrow \neg P$ is inconsistent
- `datatype F = C (F -> empty)`
- Easily provable $(\exists f : F.\top) \leftrightarrow \neg(\exists f : F.\top)$
- Allowed datatypes have to be restricted ("strict positivity")
- This is not hard in principle, but intricate datatypes are required by users, in particular nested inductive types lead to issues (Lamiaux, Forster, Sozeau, Tabareau 2026)

October 6, 2026

- OpenAI announces solution to hundreds of maths problems
- About half of them come with proofs in Lean
- relies on Comparator, tool by Henrik Böving checking that a proof proves exactly a given reference statement

yforster Update README.md

eac95b0 · now History

Preview Code Blame 22 lines (17 loc) · 590 Bytes Raw

Following an idea by Alex Meiburg on Zulip

`Challenge.lean`:

```
def LeanIsInconsistent : Prop := False

theorem iCanProveIt : LeanIsInconsistent := by
  sorry
```

`comparator.json`:

```
{
    "challenge_module": "Challenge",
    "solution_module": "Solution",
    "theorem_names": ["iCanProveIt"],
    "definition_names": ["LeanIsInconsistent"],
    "permitted_axioms": ["propext", "Quot.sound", "Classical.choice"]
}
```

Computation in proof assistants

- Rocq is optimised to allow computing in proofs / the proof assistant
- Lean avoids computing, similar to HOL: Proofs like BB(5) are out of reach
- Can lead to bugs, e.g. compiled result of functions was forgeable in Rocq
- Open research question: How much computation is needed?

A genealogy of critical bugs

- Mixing programming language and logic (termination checking, affects Rocq)
- Efficient computation (affects Rocq, unavailable in Lean)
- Excluding Russel-style issues with datatypes, especially hard in nested inductive types (affects Lean, also Rocq)
- Excluding Mirimanoff / Burali-Forti-style issues (Rocq)
- Pure implementation oversights (both)

What do we need to trust?



Implementation
/ Small Core

Around 20.000 lines of code

around 1 critical bug per year

~~always found by developer~~

this year: dozens, many found with LLMs

Changing the trusted core requires
a PhD in type theory and
years of development experience

Underlying theory

no problem in 40 years

image from Ilya Sergey's lecture notes, courtesy of Lilia Anisimova

How can we trust proof assistants?

How to ensure that proof assistants are absolutely bug-free?



mathematical
correctness proof

checked by a  proof assistant

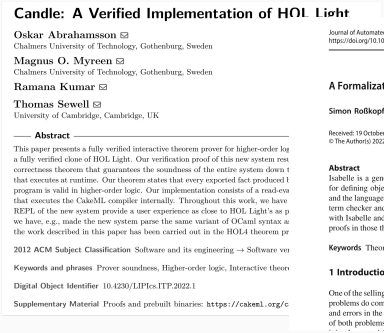
with mathematical
correctness proof

checked by a 

But what about Gödel?

- Second incompleteness theorem: a consistent, sufficiently strong system cannot prove its own consistency
- So what does "PA verified in PA" even mean?
- Theorem V in Gödel's paper: a sufficiently strong system can models its own *proof* predicate
- "PA verified in PA": The *proof* predicate is decided by a program

Self-verification in proof assistants



Candle: A Verified Implementation of HOL Light

Oskar Abrahamsson
Chalmers University of Technology, Gothenburg, Sweden
Magnus O. Myreen
Chalmers University of Technology, Gothenburg, Sweden
Ramana Kumar
Thomas Sewell
University of Cambridge, Cambridge, UK

Abstract

This paper presents a fully verified interactive theorem prover for higher-order log
a fully verified clone of HOL Light. Our verification proof of this new system rest
correctness theorem that guarantees the soundness of the entire system down t
that executes at runtime. Our theorem states that every exported fact produced l
program is valid in higher-order logic. Our implementation consists of a read-eva
that executes the CakeML compiler internally. Throughout this work, we have
REPL of the new system provide a user experience as close to HOL Light's as p
we have, e.g., made the new system parse the same variant of OCaml syntax a
the work described in this paper has been carried out in the HOL4 theorem pr

2012 ACM Subject Classification Software and its engineering → Software ver

Keywords and phrases Prover soundness, Higher-order logic, Interactive theor

Digital Object Identifier 10.4230/LIPIcs.ITP.2022.1

Supplementary Material Proofs and prebuilt binaries: https://cakeml.org/c

Journal of Automated Reasoning (2023) 67:1
https://doi.org/10.1007/s10817-022-09648-w

A Formalization and Proof Checker for Isabelle's Metalogi

Simon Roßkopf[1] · Tobias Nipkow[1]

Received: 19 October 2021 / Accepted: 20 July 2022 / Published online: 12 December 2022
© The Author(s) 2022

Abstract

Isabelle is a generic theorem prover with a fragment of higher-order logic a
for defining object logics. Isabelle also provides proof terms. We formalize t
and the language of proof terms in Isabelle/HOL, define an executable (but ine
term checker and prove its correctness w.r.t. the metalogic. We integrate the
with Isabelle and run it on a range of logics and theories to check the correct
proofs in those theories.

Keywords Theorem proving · Higher-order logic · Isabelle · Proofchecker · M

1 Introduction

One of the selling points of proof assistants is their trustworthiness. Yet in practi
problems do come up in most proof assistants. Harrison [15] distinguishes erro
and errors in the implementation (and cites examples). Our work contributes t
of both problems for the proof assistant Isabelle [35]. Isabelle is a generic th

Lean4Lean: Towards a formalized metatheory for the Lean theorem prover

Mario Carneiro
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

In this paper we present a new "external verifier" for the Lean theorem prover, written in Lean itself. This is the first complete verifier for Lean 4 other than the reference implementation in C++ used by Lean itself, and our new verifier is competitive with the original, running between 20% and 50% slower and usable to verify all of Lean's mathlib library, forming an additional step in Lean's aim to self-host the full elaborator and compiler. Moreover, because the verifier is written in a language which admits formal verification, it is possible to state and prove properties about the kernel itself, and we report on some initial steps taken in this direction to formalize the Lean type theory abstractly and show that the kernel correctly implements this theory, to eliminate the possibility of implementation bugs in the kernel and increase the trustworthiness of proofs conducted in it. This work is still ongoing but we plan to use this project to help justify any future changes to the kernel and type theory and ensure unsoundness does not sneak in through either the abstract theory or implementation bugs.

2012 ACM Subject Classification Mathematics of computing → Mathematical software performance; Software and its engineering → Formal methods; Social and professional topics → Systems analysis and design

Keywords and phrases Lean, proof assistant, external verifier, implementation, metatheory, type theory, proof theory

The fall of verification attempts

- Candle: HOL light is verified in HOL since 2022
- Lean4Lean was stuck for a long time: Lean's system has too many minor bugs, currently being fixed by Arthur Adjedj.
- Lean4Lean now making progress, based on POPL paper by Thierry Coquand, Mario Carneiro, Adrien Frabetti Mathieu, Meven Lennon-Bertrand, Paul-André Melliès, Stephanie Weirich
- MetaRocq was stuck for a long time: Rocq's type system has too many features.
- MetaLean by Jeremy Chen provides incomplete but sound checker for Lean against hand-written theory
- Con-Leche by Joachim Breitner provides incomplete but sound checker against set-theoretic model

MetaRocq





MetaCoq is developed by (left to right) Abhishek Anand, Danil Annenkov, Simon Boulier, Cyril Cohen, Yannick Forster, Meven Lennon-Bertrand, Kenji Maillard, Gregory Malecha, Jakob Botsch Nielsen, Matthieu Sozeau, Nicolas Tabareau and Théo Winterhalter.

and 29 more contributors, notably Pierre-Marie Pédrot, Jason Gross, Marcel Ullrich, Yann Leray, Loïc Pujet, Yee-Jian Tan.
Continuing a long line of work, see e.g. Coq in Coq by Bruno Barras and Benjamin Werner (1997).

From a clean specification to a running checker



- Soundness: there is a function that takes in a context and a term t and potentially produces a term A that is the type of t
- Completeness: if t is typeable, the function always returns a result (Lennon-Bertrand, Botsch Nielsen)
- To *run* the verified checker, we extract it to OCaml
  $\Rightarrow$ extraction has to be verified too (Forster, Sozeau, Tabareau, PLDI 2024)

Take homes

- I personally trust LLM-generated theorems only if they come with a proof in a proof assistant
- Still, requires verification by humans: Theorem correct? Proof structure as intended? Tools used correctly?
- Open research questions:
  - How much dependent types are needed for mathematics formalisation?
  - Is computation needed?
  - What features to have in the kernel, and what to elaborate away?
  - What's the best tradeoff for native computation?
  - What's the best tradeoff for universes?
  - What's the best tradeoff for nested inductive types?

Thank you! Questions?