# Viewpoint Column

by

Anca Muscholl and Stefan Schmid

Bordeaux University, France
and TU Berlin, Germany
anca@labri.fr and stefan.schmid@tu-berlin.de

# Adding "Verified" Badges to Our Paper, One Lemma at a Time

Matthias Függer and Thomas Nowak

LMF, CNRS, ENS Paris-Saclay, Université Paris-Saclay

**Abstract**

We describe our experience using AI coding agents to formally verify proofs from a theoretical computer science paper in Lean 4. Using a two-agent workflow where one agent generates proofs and another checks their faithfulness to the paper, we formalized several of the paper's results, including the computational model. We did not know Lean before starting. We reflect on what worked, what did not, what infrastructure is missing, and where this might lead for TCS and for mathematics more broadly.

## 1 Introduction

Formal verification of mathematical proofs has a long history. The goal of these undertakings was to express valid proof steps through explicit symbolic rules, making mathematical reasoning precise and easier to verify systematically. Early examples include Frege's *Begriffsschrift* (1879) [7], Hilbert's formal axiomatic program (1928) [16], and *Principia Mathematica* by Whitehead and Russell (1910) [22]. These developments eventually led to foundational work connecting symbolic rules with computability and proof theory by Gödel [13], Church [2], Turing [20], and Gentzen [12]. One of the earliest computer-assisted formal verification projects was de Bruijn's Automath system in the late 1960s [5], and the idea that computers should check our proofs is nearly as old as the idea that computers should help us find them. The four-color theorem was proved with computer assistance in 1976 and formally verified in Coq (now Rocq) by Gonthier and Werner in 2005 [14]. Hales's proof of the Kepler conjecture, after years of controversy over its computer-assisted parts, was formally verified in the Flyspeck project in 2014 [15].

Yet after more than half a century, the vast majority of theorems in mathematics and theoretical computer science remain verified only by human peer review. The reason is not that formal verification does not work. It is that it has cost too much.

Formalizing a single nontrivial theorem in a proof assistant like Lean [6], Rocq [1], or Isabelle [18] has traditionally required a specialist.

This is changing. AI coding agents can now interact with proof assistants autonomously: they write proof terms, compile them, read the error messages, and iterate. The human cost of formalization is dropping, and this is starting to reshape how mathematical knowledge is recorded and trusted, in TCS and beyond.

This column is an account of our own experiment in this direction. We recently had AI agents formally verify some of the proofs in a paper on distributed computing [8]. The paper studies averaging algorithms under oblivious message adversaries, introduces a notion of asymptotic subspace consensus, and gives a characterization of solvability. Several of the results now carry green "Verified in Lean" badges in the PDF, each linking to a compiled Lean 4 proof. Neither of us knew Lean before starting.

## 2 The Two-Agent Workflow

The story begins with a social media post. Jukka Suomela, whose work in distributed computing many readers of this Bulletin will know, posted about a workflow he uses: ask a chatbot to solve a math problem, then ask it to formalize the proof in Lean 4. The chatbot discovers mistakes in its own reasoning through the formalization attempt, fixes them, and tries again. Suomela pointed out that, in principle, he does not even need to look at the Lean code or know what Lean is. The formalization attempt acts as a kind of rubber duck, forcing the model into rigor.

We tried this with ChatGPT, and the iterative self-correction worked as described. But we were suspicious: was the chatbot actually producing valid Lean code? ChatGPT confirmed that it does not have access to the Lean compiler. So we installed Lean and tried to build the code. It did not build.

What followed was tedious but instructive. We went back and forth with ChatGPT, pasting compiler errors, and it identified issues each time. This was slow, because we were acting as a human compile-paste loop. The obvious next step was to use an AI coding agent that could run the Lean compiler itself.

We launched Codex, OpenAI's coding agent, in a Lean project directory and asked it to prove one of the lemmas from the paper. Codex can on its own explore relevant parts in the tex source, run shell commands, compile its own code, read the errors, and iterate. After a few minutes, it produced a Lean proof that compiled. But the Lean statement did not match the English statement of the lemma. Codex had introduced additional hypotheses not present in the paper.

This is a natural failure mode, and in hindsight an obvious one: an agent optimizing for "make it compile" will strengthen hypotheses or weaken conclusions

until Lean accepts the proof. The resulting theorem may be trivially true and bear no relation to the intended claim.

The fix was to use a second agent to check the first one's work. We set up Claude Code, Anthropic's coding agent, to review Codex's output against the paper's LaTeX source. Claude reads both the Lean file and the corresponding section of the paper, and checks for additional hypotheses not present in the paper, weakened conclusions, `sorry` placeholders that mark incomplete proofs or missing arguments, and mismatches between the Lean definitions and the paper's notation. When Claude finds issues, it writes a detailed report specifying what is wrong and how the Lean code deviates from the paper. Codex reads this report and tries again.

This two-agent loop runs with minimal human intervention. In practice, it required several iterations per lemma. Codex would often stop early with an incorrect formalization, and Claude would catch the discrepancy. Sometimes the issue was subtle: a universal quantifier over all vectors replaced by a quantifier over unit vectors, or a strict inequality weakened to a non-strict one. Sometimes it was gross: an entire hypothesis added that made the theorem trivial. The checking agent caught both kinds.

The workflow has a clear separation of concerns. Codex's job is to produce Lean code that compiles. Claude's job is to ensure that what compiles is faithful to the paper. The human's job is to write the paper, to decide which results to formalize, and to verify that the Lean definitions capture the intended concepts.

The agent instructions, which specify the protocol, are available on GitHub [11], and a short demo video of the workflow is also online [9].

# 3 What We Learned

Our paper involves processes that update their state by taking weighted averages of received values, subject to a communication pattern controlled by an adversary. The main results concern convergence properties and impossibility results characterized by graph-theoretic properties of the communication patterns.

We started with a warm-up: a basic geometric fact about the distance from a point to an open halfspace defined by a hyperplane. This was straightforward, and Codex handled it in a single pass. It gave us confidence that the workflow could produce real results.

The hardest part of the entire formalization was not a proof but the definitions. Formalizing the computational model in Lean, the processes, the rounds, the message adversary, the averaging updates, required making dozens of modeling choices that have no obvious right answer. How should the states be represented? What type should the communication graph have? These choices are invisible in the paper, where we write "each process updates its state by averaging the values it

received" and move on. In Lean, every implicit convention must be made explicit, and this took more iterations, and more human oversight, than any of the proofs.

Once the model was in place, several convergence lemmas went through without major issues. These are geometric arguments involving convex combinations, projections, and distances to halfspaces. Lean's mathematical library Mathlib [3] had good coverage of the relevant linear algebra, and Codex could find the right lemmas and chain them together. We also formalized an impossibility theorem, a result showing that certain convergence guarantees cannot be achieved under certain adversary classes. This is a combinatorial argument about the structure of communication graphs, and it required a different style of Lean proof from the geometric lemmas. The Lean code is nontrivial, and the agents produced all of it, with human oversight on the modeling choices.

Our paper also contains results on Steiner symmetrization of convex bodies, which we did not formalize. Our objective was to verify some of the paper's results, not all of them, and we stopped before tackling the symmetrization arguments.

The benefits of the workflow extend beyond the badges that end up in the final PDF. On a separate paper of ours on chemical reaction networks, we attempted the same two-agent loop and did not produce a formalization we were happy with, so we submitted the paper without badges. The attempt was still worthwhile: along the way, the formalization uncovered errors in the manuscript and suggested how to fix them. Even an unfinished formalization can serve as a careful proofreader.

The cost was modest. The entire formalization took days, not months, and required no prior Lean expertise from either of us. The monetary cost amounted to tens of euros. For comparison, a human Lean expert would likely have done a better and more reusable job, but would have taken longer and cost more. The agents gave us a quick, if somewhat brittle, path to compiled proofs.

To make formal verification visible in the paper, we developed a small LaTeX package called `verified-badges` [10]. It provides commands that add a clickable badge next to a theorem header, linking directly to the proof source. The package supports eleven proof assistants and is released under CC0. For example, the warm-up halfspace lemma mentioned at the start of this section appears in our paper as

**Lemma** (VERIFIED Lean)**.** *Let point $q \in \mathbb{R}^d$ and normal vector $v \in \mathbb{R}^d \setminus \{0\}$ define the half-space $H = \{h \in \mathbb{R}^d \mid \langle h - q, v\rangle < 0\}$. Then* $\operatorname{dist}(\{z\}, H) = \langle z - q, v\rangle / \|v\|$ *for all points $z \in \mathbb{R}^d \setminus H$.*

and the package likewise produces VERIFIED Rocq for Rocq, VERIFIED Isabelle for Isabelle, and so on. A blue variant, such as FORMALIZED Lean, marks statements whose formalization is complete but whose proof uses `sorry`. The same idea has been used independently in the Rocq community: Leray and Winterhalter's recent paper [17] marks each

formalized lemma and theorem with a small Rocq icon next to its header. A green badge indicates that the statement and proof have been translated into a formal language and that the result type-checks. This is a different claim than an English proof alone, even if it is not an absolute guarantee: the formalization gap remains.

# 4 The Missing Libraries

The problem of formalizing the right statements is not specific to our paper, or to distributed computing, or even to computer science. Mathematics rests on informal consensus about what definitions mean. When we write "let $G$ be a graph," we rely on a shared understanding that has been stable for a century. A proof assistant does not have that shared understanding unless someone has encoded it.

For core mathematics, Mathlib now provides a large and well-maintained library: groups, rings, topological spaces, measure theory, and much more. It contains over two million lines of Lean code and is growing fast. When we needed a fact about inner products or convex combinations, Mathlib had it, and Codex could find it. This is why the geometric parts of our formalization went smoothly: someone had already done the hard work of defining the right abstractions.

But beyond core mathematics, the coverage is patchy. Physics, economics, and theoretical computer science build on mathematical models that, when formalized at all, are formalized once for a specific result rather than being assembled into widely-adopted, Mathlib-scale curated libraries. Pieces exist but they are scattered: Mathlib has Turing machines; Isabelle's Archive of Formal Proofs hosts a recent formalization of Nash equilibria for finite games, of the Cook–Levin theorem, of monotone-circuit lower bounds, and of the Chandy–Lamport snapshot algorithm; and Rocq has the deepest coverage of computational systems, with verified compilers, separation-logic frameworks, and cryptographic frameworks among others. But there is no standard shared library for, say, the LOCAL or CONGEST models, for population protocols, or for Lagrangian mechanics that a researcher can pick up and build on. In our case, we performed ad-hoc formalizations: each definition was written for this specific paper. This works once, but the resulting Lean code may not be reusable.

CSLib [4] is a recent initiative aiming to build a Lean library for computer science foundations: computational models, complexity analysis tools, verified algorithms and data structures, and deductive verification techniques. It targets, for computer science, the role Mathlib plays for pure mathematics.

Research communities are well-placed to drive this work themselves. Each TCS subcommunity has some standard models that its members already largely share, even if formalizing them surfaces choices that papers usually paper over. A shared formalization of those models, with families of interoperable variants

where conventions differ, is a natural service for a conference or a special-interest group to offer its authors, alongside proceedings and artifact evaluation. The hard part is not the technical work but the governance: who maintains the library, who arbitrates incompatible encodings, and how this labor earns credit in the usual academic accounting.

The same principle applies to any field with a mathematical backbone. What is needed is broad coverage: the standard models of computation, the standard models of physics, the standard models of economic theory, all formalized in a common framework so that results can interoperate. A small group of researchers, possibly aided by AI agents, could formalize the standard models from foundational textbooks and maintain them as an open library. This would be a contribution not just to formal verification but to the foundations of each field: the process of formalizing a definition forces you to resolve every ambiguity, and the resulting library becomes a reference that is precise in a way that no textbook can be. It would also have a pedagogical dimension, since students could trace any claim back to the axioms mechanically, turning the library into a teaching tool as well as a reference.

# 5 Looking Forward

Today, the proofs in our papers are checked by peer reviewers who read them, think about them, and try to convince themselves that they are correct. This process is valuable but imperfect. Published proofs are sometimes found to be wrong years or decades after the fact, and human proof-checking does not scale to the complexity of modern mathematics. Voevodsky, who discovered an error in a 1989 paper of his own, is a well-known case: he became one of the most prominent advocates for formal verification and devoted much of the latter part of his career to developing univalent foundations and the UniMath library [21].

Formal verification offers a complementary check. It does not replace peer review, since reviewers also evaluate significance, clarity, novelty, and the appropriateness of the model. But it can provide a machine-checked guarantee that the mathematical reasoning is correct, conditional on the formalization being faithful. Until recently, the cost of formalization made this impractical for most papers. AI agents have changed this.

Consider arXiv. It is already a public pool of papers with mathematical claims, and authors or referees can opt to put any of them through a formalization attempt. The `paper-badger` tool [19] is one concrete instance: given an arXiv identifier, it extracts the theorem statements and asks an AI agent to formalize them in Lean and Mathlib. Used as an opt-in diagnostic by authors, it will not catch every error—the formalization gap is real, and theorem statements often depend on macros, hidden

conventions, and definitions spread across the paper, so false negatives will be common. But the mere attempt at formalization would surface ambiguities in definitions and statements that English prose hides, and would catch some errors that currently slip through.

Or consider peer review. A conference or journal could assign, alongside its human reviewers, an AI reviewer whose sole job is to formalize the claimed results and check the proofs. This reviewer would not evaluate novelty, significance, or writing quality. It would answer one question: does the proof, as written, imply the claimed theorem in a formal sense? A report from such a reviewer would not be a verdict but a diagnostic. It might say: "Lemma 3 compiles as stated, but the proof of Theorem 1 uses a stronger induction hypothesis than the one stated in the paper." Human reviewers could then focus their energy on the parts of reviewing that require judgment, the parts that machines cannot do, rather than spending hours line-checking algebraic manipulations.

Beyond individual papers, the corpus of published mathematics is a large system running on trust: each paper assumes the correctness of the papers it cites, and the chain of trust extends back to the axioms. Occasionally a link breaks and the consequences propagate forward. A formally verified mathematical library would replace this chain of trust with a chain of proof. Mathlib already covers large parts of undergraduate and graduate mathematics. It is growing fast, and AI agents are accelerating the pace. It is not hard to imagine a future where the standard theorems of mathematics exist in a formal library, and new results are expected to build on that library rather than reprove things from scratch in English. This would be a shift comparable to the introduction of rigorous proof itself in the nineteenth century.

As AI-assisted formalization becomes routine, a finer taxonomy of mathematical knowledge will become visible. Some results are *formalized*: their statements and proofs exist in a proof assistant and type-check. Others are *easily formalizable*: their proofs are clear and detailed enough that an AI agent can formalize them with minimal human guidance, even if no one has bothered to do so yet. And then there are results that are *not obviously formalizable*.

This stratification has always existed, but formalization has been too expensive for it to be empirically observable at scale. As the cost drops, the boundaries will become visible, and they will tell us something about our own mathematical practice. A result that resists formalization is not necessarily wrong, but it is a result whose proof rests on conventions and intuitions that have not been made explicit. This is useful information. It points to places where our understanding is less complete than we thought, where the informal consensus may be hiding ambiguity or error. In the long run, the most valuable contribution of AI-assisted formalization may not be the proofs it verifies but the gaps it reveals.

This is not science fiction. The individual components exist today. What is

missing is infrastructure: the curated libraries of formal definitions, the integration with submission systems, and the community norms that would make this feel natural rather than imposed.

Natural-language proofs will continue to be valuable. The agents need the English proof as a guide, and readers benefit from understanding the proof, not just knowing that it is correct. A Lean term certifies that the reasoning holds, but it does not, by itself, convey how the author thought about the problem. The English proof carries the intuitions and analogies that let a reader spot the next variant, the next generalization, the next theorem. As the cost of formal verification drops, it can become a routine check rather than a special project, and natural-language proofs can keep doing what they are best at.

# References


[1] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer, Heidelberg, 2004. doi:10.1007/978-3-662-07964-5.

[2] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936. doi:10.2307/2371045.

[3] The Mathlib Community. Mathlib: a unified library of mathematics formalized in Lean 4. https://leanprover-community.github.io/mathlib4_docs/.

[4] CSLib. A focused effort on formalizing computer science in Lean, 2026. https://www.cslib.io/.

[5] N. G. de Bruijn. The mathematical language AUTOMATH, its usage, and some of its extensions. In M. Laudet, D. Lacombe, L. Nolin, and M. Schützenberger, editors, *Proceedings of the Symposium on Automatic Demonstration*, volume 125 of *Lecture Notes in Mathematics*, pages 29–61, Heidelberg, 1970. Springer. doi:10.1007/BFb0060623.

[6] Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In André Platzer and Geoff Sutcliffe, editors, *Proceedings of the 28th International Conference on Automated Deduction (CADE 2021)*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635, Cham, 2021. Springer. doi:10.1007/978-3-030-79876-5_37.

[7] Gottlob Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Verlag von Louis Nebert, Halle, 1879.

[8] Matthias Függer and Thomas Nowak. Asymptotic subspace consensus in dynamic networks, 2026. arXiv:2602.19121 [cs.DC]. To appear in the Proceedings of the 5th Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2026). doi:10.48550/arXiv.2602.19121.

[9] Matthias Függer and Thomas Nowak. A short demo of the two-agent formalization workflow, 2026. Video. https://youtu.be/rE5_Zc2AMwM.

[10] Matthias Függer and Thomas Nowak. Verified badges: a LaTeX package for proof-assistant badges, 2026. https://github.com/BioDisCo/verified-badges.

[11] Matthias Függer and Thomas Nowak. Vibe formalizing: an AI-assisted formal verification workflow, 2026. https://github.com/BioDisCo/vibe-formalizing.

[12] Gerhard Gentzen. Untersuchungen über das logische Schließen. I. *Mathematische Zeitschrift*, 39(1):176–210, 1935. doi:10.1007/BF01201353.

[13] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38(1):173–198, 1931. doi:10.1007/BF01700692.

[14] Georges Gonthier. Formal proof—the four-color theorem. *Notices of the American Mathematical Society*, 55(11):1382–1393, 2008. https://www.ams.org/notices/200811/tx081101382p.pdf.

[15] Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Le Truong Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, Quang Truong Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, Thi Hoai An Ta, Nam Trung Tran, Thi Diep Trieu, Josef Urban, Ky Vu, and Roland Zumkeller. A formal proof of the Kepler conjecture. *Forum of Mathematics, Pi*, 5:e2, 2017. doi:10.1017/fmp.2017.1.

[16] David Hilbert. Die Grundlagen der Mathematik. *Abhandlungen aus dem Mathematischen Seminar der Hamburgischen Universität*, 6:65–85, 1928. doi:10.1007/BF02940602.

[17] Yann Leray and Théo Winterhalter. Encode the Cake and Eat It Too: Controlling Computation in Type Theory, Locally. *Proceedings of the ACM on Programming Languages*, 10(POPL):62:1–62:30, 2026. doi:10.1145/3776704.

[18] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, Heidelberg, 2002. doi:10.1007/3-540-45949-9.

[19] Thomas Nowak. paper-badger: automated Lean 4 formalization and verification badges for arXiv papers, 2026. https://github.com/BioDisCo/paper-badger.

[20] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1936. doi:10.1112/plms/s2-42.1.230.

[21] Vladimir Voevodsky. The origins and motivations of univalent foundations, 2014. The Institute Letter, Institute for Advanced Study, Summer 2014. https://www.ias.edu/ideas/2014/voevodsky-origins.

[22] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*, volume 1. Cambridge University Press, Cambridge, 1910.