

Lecture 1 on Interactive Proof Assistants :

What Are They?

Assaf Kfoury¹

kfoury@bu.edu

Boston University

16 March 2026

¹ <https://www.cs.bu.edu/~kfoury/>

NOTES for slide 1:

- Many slides are not numbered and without a grey banner (like this one). These contain my notes, which I intend to present verbally during the lecture – as much of them as I could within a lecture of one hour and a half – or in writing if a board is available. I intend the actual presentation with the overhead projector to be limited to the numbered slides with a grey banner.
- I have inserted a profusion of hyperlinks to places on the Web to back up many of my claims and statements. These are identified by a green frame like [this](#).
- I composed my slides for an audience which includes many (mostly?) *philosophers of mathematics* and *philosophical logicians*. This took a little maneuvering and speculation on my part, because my usual audiences are different – mostly consisting of *computer scientists* and/or *mathematicians*.

NOTES for slide 1:

- Instead of referring to “*computer scientists* and/or *mathematicians*” every time, I combine the two groups by just calling them *mathematicians*, among which I include other scientists for whom the concept of *mathematical proof* is central in their work and research – and this is the case for *theoretical economists*, for *theoretical physicists*, for some *engineers*, and for others.
- Finally, as a reminder before I start my presentation, this is the first lecture in a three-lecture series – the second and third lectures are:
 - **Lecture 2 on Proof Assistants: How Do We Use them?**
 - **Lecture 3 on Proof Assistants: Why Do They Work?**

So, the first lecture is more on the *history* of IPAs, the second lecture is more on the *pragmatics* of IPAs, and the third lecture is more on the theoretical *foundations* of IPAs.

- 1 New Ways in Mathematical Practices
- 2 Expressiveness vs. Automation
- 3 Frustrating Results, Spectacular Successes, Embarrassing Failures
- 4 Not All Parts of Mathematics Have Profited Equally
- 5 Successes of ATPs and IPAs Across Mathematical Domains
- 6 Lineages of Two Leading ATPs and Two Leading IPAs

Hopefully, I will have time to cover Sections 1, 2, 3, and perhaps 4 also, but most likely not Sections 5 and 6. I will make all of my slides available for download later.

What Are Proof Assistants?

or – better – to stress interaction with **human agents**:

What Are Interactive Proof Assistants?

What Are Proof Assistants?

or – better – to stress interaction with **human agents**:

What Are **Interactive** Proof Assistants?

They are integrated software ecosystems that assist mathematicians in constructing error-free mathematics, acting as force multipliers for efficiency and productivity.

► SKIP THE NOTES

NOTES for slide 4:

... but they will not make mathematicians disappear –
contrary to what some tycoons of the tech industry like to predict!
See for example what they say in *The San Francisco Consensus*.

- Before going further, I want to clarify what mathematicians mean when they have produced a written “proof” and what they expect from it.
- A mathematical proof is absolute. In theory, its correctness can be verified or certified by anyone because it can be broken down into tiny, irrefutable logical steps.^a While practicing mathematicians almost never spell out proofs to this extreme exhaustive level of detail, we operate on the assumption that it can always be done. When a theorem turns out to be false – which occasionally happens – it reveals a gap in that verification process, because the purported proof of the theorem contains an unverified (or unverifiable) step in its chain of rigorous logical reasoning.

^a Like the steps of a *Turing machine*, if you know what a *Turing machine* is.

NOTES for slide 4:

- One more comment to make explicit an important duality in the role of proofs:
 - ▶ Beyond their **verification** or **certification function** – or **narrow-scope** role – which is the rigorous step-by-step reasoning that gives us absolute certainty that a theorem is true,
 - ▶ proofs serve what we may call an **illuminating function** – or **global-scope** role – which varies from mathematician to mathematician. Even collaborators working on the same proof may experience this differently, shaped by their distinct backgrounds and expertise. This broader role goes beyond mechanical certification: it helps us build personal **understanding** and **appreciation** of why a theorem holds, reveal how new theorems fit into the wider mathematical landscape, and uncover familiar patterns hidden behind unfamiliar notation or unknown similarities between different areas of mathematics. Critically, what **illuminates** a concept or a theorem statement from the perspective of one mathematician – often described as better **“intuition”** and **“insights”** – may have little to no effect on another mathematician, reflecting their different backgrounds and interests.
- One further comment from Oded Goldreich on the **illuminating function** of proofs: This is a qualification of *“the main message, which is not stated explicitly. It is that proofs provide better understanding of the theorem itself and often also of related and even unrelated material.”*

NOTES for slide 4:

- In conclusion, IPAs excel at supporting the **verification/certification** function of proofs – the mechanical, step-by-step checking which is fundamentally the same process across all mathematical sciences. However, IPAs face inherent limitations in supporting the **illuminating** role of proofs: they cannot readily adapt (ever?) to the diverse expertise, goals, interests, and motivations of every human who chooses to pursue mathematical research.
- And the human dimension of being a mathematician is about other aspects far beyond the effects of the **illuminating** role of proofs. Although this falls outside the scope of today's lecture, let me note: all mathematicians derive deep pleasure and satisfaction from discovering new results – often couched as new “insights” and “intuition” which they eagerly share with colleagues, but not with emotionally inert software systems, even if the latter are programmed to simulate human emotions. More fundamentally, does it even make sense to “share” feelings with software systems, simulated or otherwise? And to what end? Is it to “befriend” a software system, or to find an automated surrogate therapist, or . . . ? Yet, many mathematicians fall into the trap of a “contradictory consciousness” when they join outside observers in asserting that mathematical practice by humans faces extinction, even as they continue to celebrate mathematics’ beauty with their colleagues and yet they say their practice is under threat – indeed, under threat by whom, other than those controlling the purse and their salaries?

NOTES for slide 4:

- One further comment from Oded Goldreich on the preceding bullet point, which focuses on the **social aspect** and also the **emotional aspect** of practicing mathematics and working out proofs: *“There is a pure **intellectual aspect** [in addition to the preceding two aspects]: mathematicians seek to understand mathematics. They are a type of humans who are interested in this subject area, and like many humans, they seek to understand the world (and change it . . .), not only to describe and predict some phenomena.”*

“Elaborating further: Consider a mental experiment in which you are given a device that predicts all you want. In what sense is it different from the world itself? It is only different in telling you what will happen ahead of time when you ask it, rather than in real time. But it gives you no understanding at all. The world remains totally mysterious/magical to you. You remain in the exact position of an animal that knows that an electric shock follows the ring of a bell (in an experiment to which it is subjected . . .). You may claim that you are in a better situation, since you can ask the box ahead of time when will the electric shock come”

New Ways in Mathematical Practices

Locating **proof assistants** in the ecosystem of software tools used by math scientists:

- Of course, they use *email*, *word processing*, *LaTeX*, *Web browsers*, etc.

New Ways in Mathematical Practices

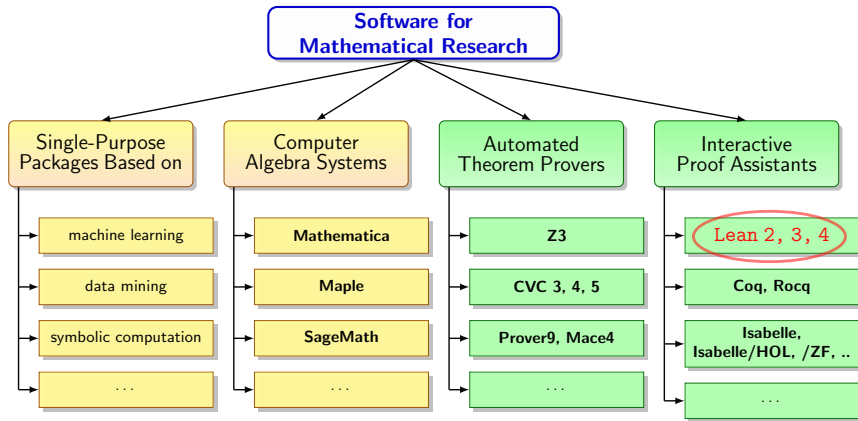
Locating **proof assistants** in the ecosystem of software tools used by math scientists:

- Of course, they use *email*, *word processing*, *LaTeX*, *Web browsers*, etc.
- But beyond the preceding, they also use some more advanced systems:

New Ways in Mathematical Practices

Locating **proof assistants** in the ecosystem of software tools used by math scientists:

- Of course, they use **email**, **word processing**, **LaTeX**, **Web browsers**, etc.
- But beyond the preceding, they also use some more advanced systems:



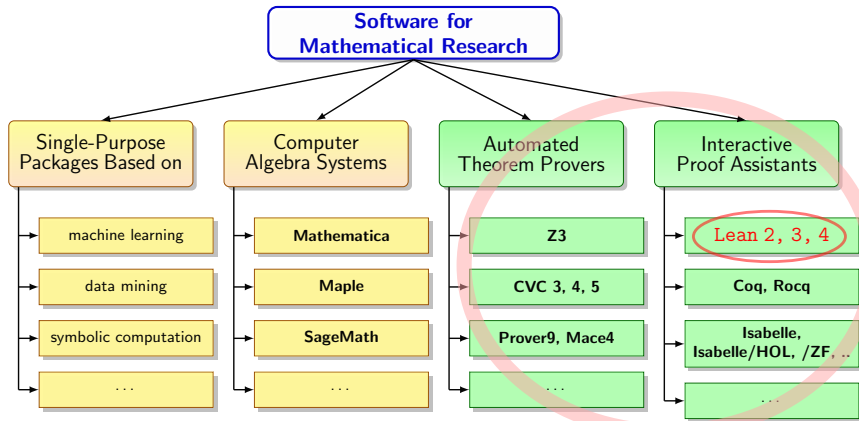
(*) More on **Lean 4** in Lectures 2 and 3

► SKIP THE NOTES

New Ways in Mathematical Practices

Locating **proof assistants** in the ecosystem of software tools used by math scientists:

- Of course, they use **email**, **word processing**, **LaTeX**, **Web browsers**, etc.
- But beyond the preceding, they also use some more advanced systems:



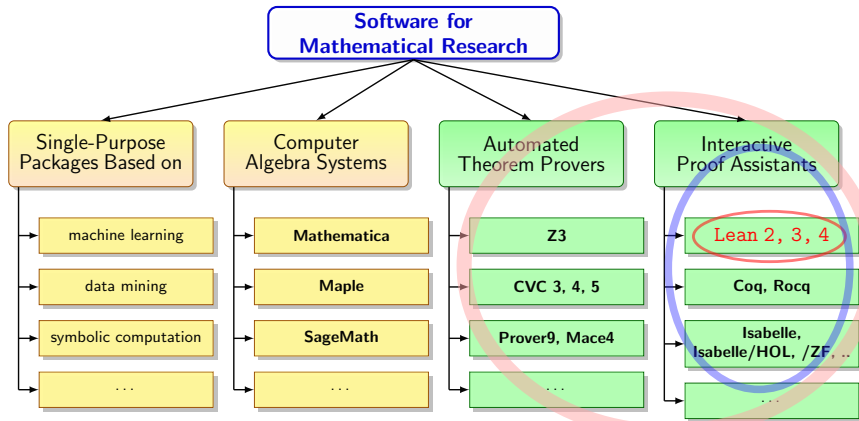
(*) More on **Lean 4** in Lectures 2 and 3

► SKIP THE NOTES

New Ways in Mathematical Practices

Locating **proof assistants** in the ecosystem of software tools used by math scientists:

- Of course, they use **email**, **word processing**, **LaTeX**, **Web browsers**, etc.
- But beyond the preceding, they also use some more advanced systems:



(*) More on **Lean 4** in Lectures 2 and 3

► SKIP THE NOTES

NOTES for slide 9:

- The classification on the preceding slide is an *approximation*, i.e., some of these systems are not pure of a kind. For example, many IPAs allow the user to invoke ATPs: **Lean 4**, **Coq/Rocq**, and **Isabelle/HOL** can invoke the SAT/SMT solvers **Z3** and **CVC4/cvc5**.
- Moreover, in recent years there have been good attempts to construct software packages where CASs and IPAs co-exist and collaborate.
- The preceding-slide classification is also an *approximation* as it omits:
 - Systems that cannot be neatly inserted in one of the 4 columns. There is currently a large variety of software systems combining elements from the 4 columns, perhaps foremost among these are **model checkers** and **verification-aware programming languages**.
 - Systems narrowly focused on numeric computing and solutions are also omitted – such as **MATLAB** and **NumPy** – which can be placed under the column CAS (the second **yellow column** from the left) even though they are much less general-purpose than Mathematica, Maple, etc.

NOTES for slide 9:

- Systems in the yellow columns are implemented in a variety of non-functional programming languages. For example, **Mathematica** is implemented with a combination of **C**, **C++**, **Java**, and a custom-designed programming language called **Wolfram Language**; **Maple** is primarily implemented with **C** and **Java**; and **SageMath** is primarily implemented with **Python** and **Cython** (a superset of **Python** that compiles to **C**). Most of the single-purpose software packages (often called *expert systems*) involve implementation in **C**.
- Systems in the yellow columns are implemented by software engineers and computer scientists who do not need much of a background in **math logic**, in contrast to systems in the green columns whose foundations are in **mathematical logic** and whose implementors should be so trained.

NOTES for slide 9:

- Among systems in the **green columns** there is an interesting distinction:
 - ▶ IPAs are primarily implemented using functional programming languages:
Lean 4 is a self-hosting, fully-featured functional programming language that is implemented in **Lean** itself and some **C++**; **Isabelle** is mostly implemented in **Standard ML** and **Scala**, which are both functional languages with the second incorporating object-oriented features; and **Coq/Rocq** is written in the **OCaml** functional language.
 - ▶ whereas ATPs are implemented with non-functional programming languages:
Z3 is primarily written in **C++**; **CVC4** and **cvc5** are entirely written in **C++**; and **Prover9** and **Mace4** are primarily implemented in **C**.
 - ▶ **This distinction between the implementation of IPAs and the implementation of ATPs is not accidental. We will get to explain it in Lecture 3.**

Expressiveness vs. Automation: Degrees of Interaction

- **Interactions with computers in proving theorems** – *i.e.*, executions of software packages, which are turned **on** and **off** by humans, in pursuit of producing **rigorous** (and sometimes **formal** too) proofs – span a wide spectrum of possibilities and degrees:
 - (1) **At one extreme**, computers are only used to check formal proofs produced by humans.
 - (2) **At the other extreme**, the provers/calculators may be highly automated and powerful, while still subject to some degree of human guidance and control, if only to be started and stopped.

²Notwithstanding what The San Francisco Consensus predicts.

Expressiveness vs. Automation: Degrees of Interaction

- **Interactions with computers in proving theorems** – i.e., executions of software packages, which are turned **on** and **off** by humans, in pursuit of producing **rigorous** (and sometimes **formal** too) proofs – span a wide spectrum of possibilities and degrees:
 - (1) **At one extreme**, computers are only used to check formal proofs produced by humans.
 - (2) **At the other extreme**, the provers/calculators may be highly automated and powerful, while still subject to some degree of human guidance and control, if only to be started and stopped.
- **Given practical and insuperable theoretical limitations of full automation**, it is likely that **interactive theorem proving** (of various degrees) with ATPs and IPAs will remain the only way, in the foreseeable future, for humans to thrive in doing mathematics² – to formalize most non-trivial theorems in mathematics or computer system correctness.

² Notwithstanding what The San Francisco Consensus predicts.

Expressiveness vs. Automation: Degrees of Interaction

- **Interactions with computers in proving theorems** – i.e., executions of software packages, which are turned **on** and **off** by humans, in pursuit of producing **rigorous** (and sometimes **formal** too) proofs – span a wide spectrum of possibilities and degrees:
 - (1) **At one extreme**, computers are only used to check formal proofs produced by humans.
 - (2) **At the other extreme**, the provers/calculators may be highly automated and powerful, while still subject to some degree of human guidance and control, if only to be started and stopped.
- **Given practical and insuperable theoretical limitations of full automation**, it is likely that **interactive theorem proving** (of various degrees) with ATPs and IPAs will remain the only way, in the foreseeable future, for humans to thrive in doing mathematics² – to formalize most non-trivial theorems in mathematics or computer system correctness.
- Among software systems in the **green columns**, IPAs are closer to (1) and ATPs closer to (2).
- All software systems in the **yellow columns** are closer to (2).
- **In conclusion – for pedagogical reasons at least, as well as natural (human) curiosity to understand and engage in rigorous reasoning – IPAs are probably better instruments than ATPs (in **green columns**) CASs (in **yellow columns**).**

► SKIP THE NOTES

² Notwithstanding what **The San Francisco Consensus** predicts.

NOTES for slide 12:

- This is a further comment on A.I. and the outlandish claims of the San Francisco Consensus:

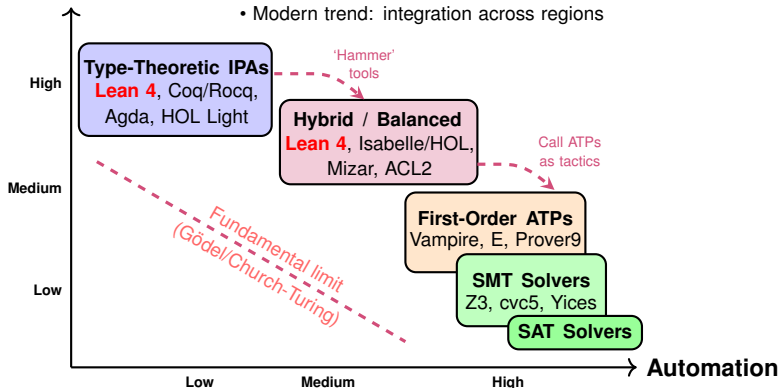
According to what Sam Altman, OpenAI CEO, said in an interview about a year ago: *"I suspect that in a couple of years on almost any topic, the most interesting, maybe the most empathetic conversation that you could have will be with an AI."*

- As if all that math scientists care about is to prove the next theorem . . . Math scientists do a lot of things as an academic community – *together in groups or individually, in cooperation or in rivalry, in celebration or in disappointment* – and many of these things have nothing to do with finding a proof for the next theorem or just knowing that the statement of the next theorem is true.

Expressiveness vs. Automation: A Schematic View

Expressiveness

- No system can be both maximally expressive *and* fully automated
- Modern trend: integration across regions



- ▶ **Vertical axis:** Expressiveness (higher = can encode more mathematics).
- ▶ **Horizontal axis:** Automation (higher = less human guidance needed).
- ▶ **Tradeoff:** Fundamental limit (Gödel/Church-Turing) prevents full automation in expressive systems.
- ▶ **Modern trend:** 'Hammers', ML, and hybrid architectures try to push toward more automation.

▶ SKIP THE NOTES

NOTES for slide 13:

Some of the references you may wish to consult, if you want to go deeper.

Foundational Work:

- **Harrison, J.** (2009). *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press.
- **Paulson, L.C.** (1989). The Foundation of a Generic Theorem Prover. *Journal of Automated Reasoning* 5(3), 363–397.
- **Wiedijk, F.** (2006). *The Seventeen Provers of the World*. Springer LNAI 3600.

Integration Approaches (“Hammers”):

- **Blanchette, J.C., Kaliszyk, C., Paulson, L.C., Urban, J.** (2016). Hammering towards QED. *Journal of Formalized Reasoning* 9(1), 101–148.
- **Kaliszyk, C., Urban, J.** (2013). HOL(y)Hammer: Online ATP Service for HOL Light. *LPAR-19*, 407–418.

Theoretical Limits:

- **Gödel’s Incompleteness Theorems** (1931): Expressive systems cannot be both complete and consistent.
- **Church-Turing**: Undecidability of first-order logic.

Frustrating Results

This is a sample – not a survey – of two results obtained with the help of software packages. They were both obtained by *brute-force searches*, but without shedding much insight, at least initially, into an underlying theory.

(1) **Graph theory** – *Four-Color Problem*.

Solution found in 1976 using a **single-purpose software system**, implemented with the “dinosaur” programming language FORTRAN.

Frustrating Results

This is a sample – not a survey – of two results obtained with the help of software packages. They were both obtained by *brute-force searches*, but without shedding much insight, at least initially, into an underlying theory.

(1) **Graph theory** – *Four-Color Problem*.

Solution found in 1976 using a **single-purpose software system**, implemented with the “dinosaur” programming language FORTRAN.

(2) **Combinatorial number theory** – *Boolean Pythagorean Triples problem*.

Solution found in 2016 using a **SAT solver**, a special type of ATP.

▶ SKIP THE NOTES

NOTES for slide 15:

Comments on *Four-Color Problem*:

- The mathematicians Kenneth Appel and Wolfgang Haken resolved the 4-color problem in 1976. They proved that any map can be colored using only four colors without adjacent regions sharing the same color, utilizing over 1,200 hours of computer time to verify 1,936 reducible configurations, marking the first major computer-assisted proof. Their program was written in **Fortran**, a dinosaur among programming languages, and was a single-purpose system (which can thus be placed in the leftmost yellow column in Slide 4).
- Appel's and Haken's methodology of analyzing thousands of map configurations with a computer program was a controversial method at the time because it could not be fully verified by hand.
- However, despite initial skepticism, the result was subsequently verified and upheld . . . it was formally verified using the proof assistant **Coq** in 2005.
- The Appel-Haken result was nevertheless a landmark achievement in graph theory, as it was the first major theorem proved with the aid of a computer.

NOTES for slide 15:

Comments on *Boolean Pythagorean Triples problem*:

- The *Boolean Pythagorean Triples* problem asks: Is it possible to color each of the positive integers either red or blue, so that no Pythagorean triple of integers a, b, c (satisfying $a^2 + b^2 = c^2$) consists of all red or all blue members? In other words, can the positive integers be partitioned into two sets such that no Pythagorean triple is monochromatic (entirely contained in one set)?
- Heule, Kullmann and Marek solved this problem in May 2016 through a computer-assisted proof, which showed that such a coloring is only possible up to number 7,824. More precisely, the set $\{1, \dots, 7824\}$ can be partitioned into two subsets, neither of which contains a Pythagorean triple; however, this is not possible for $\{1, \dots, 7825\}$.
- Scale of computation: The proof required 4 CPU-years of computation and generated a 200 terabyte propositional proof, which was later compressed to 68 gigabytes. This made it one of the largest computer-assisted proofs ever produced at the time.
- Formal verification: The result was not merely computed but formally verified. Heule, Kullmann and Marek produced a trace of their unsatisfiability proofs that was verified using an independently written checker. Subsequently, Cruz-Filipe and Schneider-Kamp formalized the problem in *Coq* and extracted a correct-by-construction proof checker that verified the 200 TB proof.

NOTES for slide 15:

More comments on *Boolean Pythagorean Triples problem*:

- Bridge between SAT solving and formal verification: This work provides a powerful framework in which we can rigorously verify proofs of other open conjectures in Mathematics that are obtained via an encoding into SAT. Future applications will only need to formalize the problem and its propositional encoding, and can afterwards apply these tools to verify results produced by SAT solvers.
- Recognition: The paper describing the proof won the best paper award at the SAT 2016 conference, and Ronald Graham's \$ 100 prize for the solution of the problem was awarded to Marijn Heule.
- The *Boolean Pythagorean Triples* problem thus represents an important milestone in demonstrating that massive computer-assisted proofs with ATPs can be rigorously verified using IPAs, bridging SAT solving (a case of ATPs) with formal verification.
- References:
 - Heule, M.J.H., Kullmann, O., and Marek, V.W. (2016). "Solving and Verifying the Boolean Pythagorean Triples Problem via Cube-and-Conquer." In *Proceedings of 19th Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT 2016)*, LNCS vol. 9710, pp. 228-245. Springer.
 - Cruz-Filipe, L. and Schneider-Kamp, P. (2017). "Formally Proving the Boolean Pythagorean Triples Conjecture." In *Proceedings of 21st Int'l Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-21)*, EPiC Series in Computing, vol. 46, pp. 509-522. [This paper formalizes the problem in Coq]
 - Cruz-Filipe, L., Heule, M.J.H., Hunt Jr., W.A., Kaufmann, M., and Schneider-Kamp, P. (2019). "Formally Verifying the Solution to the Boolean Pythagorean Triples Problem." *Journal of Automated Reasoning*, 63(3):695-722. [This is the comprehensive journal version of the formal verification]

NOTES for slide 15:

Final comments on **Four-Color Problem** and **Boolean Pythagorean Triples problem** – at least in the way they were initially clinched:

- **Why do I call them “frustrating” results?**
- To me at least, they come across as curiosities that were revealed – *not by having developed new theories* informing how planar graphs (resp. how Pythagorean triples) should occur or by relating them to some **prior theories** about planar graphs (resp. about adding squares of integers) – but by an exhaustive search for particular combinations of graph patterns (resp. three numbers) through large finite configurations of planar graphs (resp. finite sets of integers).
- At best, in the future, they will be experimental confirmations of what new theories – unknown and yet to be discovered – will predict.
- Here is a formulation and a comment re: **Boolean Pythagorean Triples** problem, by my colleague Michael Harris at Columbia:

*Let $Q(x, y, z)$ be a quadratic form in three variables with integer coefficients, and assume its coefficients have no common factors. Assume it has solutions mod p for every prime p , and assume it has a real solution. **Question:** Can the positive integers be partitioned into two subsets X and Y such that $Q(x, y, z)$ has no solution $Q(a, b, c) = 0$ with all of a, b, c either in X or in Y ? There is nothing special about $Q(x, y, z) = x^2 + y^2 - z^2$. But the general question seems contrived to me. It is possible that it has some application in analytic number theory but it has no bearing on algebraic number theory.*

Spectacular Successes

This is a sample – not a survey – of three big results obtained with IPAs, in chronological order.

(1) *Coq/Rocq: CompCert Verified C Compiler (2006-present)*

Verification of a production *compiler* used in safety-critical systems.

Spectacular Successes

This is a sample – not a survey – of three big results obtained with IPAs, in chronological order.

(1) ***Coq/Rocq: CompCert Verified C Compiler (2006-present)***

Verification of a production ***compiler*** used in safety-critical systems.

(2) ***Isabelle: seL4 Verified Operating System Kernel (2009)***

Verification of a complete ***operating system*** kernel.

Spectacular Successes

This is a sample – not a survey – of three big results obtained with IPAs, in chronological order.

(1) ***Coq/Rocq: CompCert Verified C Compiler (2006-present)***

Verification of a production **compiler** used in safety-critical systems.

(2) ***Isabelle: seL4 Verified Operating System Kernel (2009)***

Verification of a complete **operating system** kernel.

(3) ***Lean: The Liquid Tensor Experiment (2022)***

Verification of cutting-edge mathematical research.

► SKIP THE NOTES

NOTES for slide 18:

Comments and references on *Coq/Rocq: CompCert Verified C Compiler*.

- *CompCert* is a formally verified optimizing compiler from *Clight* (a large subset of the C programming language) to PowerPC assembly code (and later ARM, RISC-V, x86, and x86-64), using the *Coq* IPA, both for programming the compiler and for proving its correctness. For the development of *CompCert*, the 2021 ACM Software System Award was awarded to Xavier Leroy and the development team, recognizing “a software system that has had a lasting influence” and considered the highest award for software originating from computer science research
- *CompCert* is the first practically useful optimizing compiler targeting multiple commercial architectures that has a complete, mechanically checked proof of its correctness. The verification guarantees that the compiled code behaves exactly as prescribed by the semantics of the source program, ruling out the possibility of compiler-introduced bugs. Airbus uses *CompCert* to compile software for its planes, and the only bugs found in *CompCert* have been in the unverified frontend and in assembly code generation where a number was too large for an instruction field.
- References:
 - Leroy, Xavier (2009). “Formal verification of a realistic compiler.” *Communications of the ACM*, 52(7):107-115. DOI: 10.1145/1538788.1538814
 - Klein, Gerwin et al. (2014). “Comprehensive Formal Verification of an OS Microkernel.” *ACM Transactions on Computer Systems*, 32(1):2:1-2:70.

NOTES for slide 18:

Comments and references on *Isabelle: seL4 Verified Operating System Kernel*:

- The *seL4 project* achieved the first formal proof of functional correctness of a complete, general-purpose operating-system kernel, comprising 8,700 lines of C code and 600 lines of assembler code. The verification was performed in the **Isabelle/HOL** IPA, connecting an abstract operational specification down to the C implementation.
- The functional correctness proof means that if the assumptions are true, *seL4* will have: no code injection attacks, no buffer overflows, no NULL pointer access, no ill-typed pointer access, no memory leaks, no non-termination (all kernel calls terminate), no arithmetic exceptions, and all user input is checked and validated. The total effort for *seL4* was about 20 person-years for the proof, including 9 person-years invested in formal language frameworks, proof tools, and automation. The project demonstrated that formal verification of production-quality operating system code is feasible with modern theorem proving tools.
- References:
 - Klein, Gerwin, Kevin Elphinstone, Gernot Heiser, et al. (2009). "seL4: Formal Verification of an OS Kernel." In *Proceedings of 22nd ACM Symp. on Operating Systems Principles (SOSP)*, pp. 207-220.
 - Klein, Gerwin, June Andronick, Kevin Elphinstone, et al. (2014). "Comprehensive Formal Verification of an OS Microkernel." *ACM Transactions on Computer Systems*, 32(1):2:1-2:70.

NOTES for slide 18:

Comments and references on *Lean: The Liquid Tensor Experiment*:

- The Liquid Tensor Experiment was completed on July 14, 2022, providing a formal verification of the main theorem of liquid vector spaces using the Lean proof assistant. This was in response to a challenge posed by Fields Medalist Peter Scholze in December 2020.
- Peter Scholze stated: “I find it absolutely insane that interactive proof assistants are now at the level that within a very reasonable time span they can formally verify difficult original research.” This represented the formalization of a complex theorem about complex objects – complementary to previous major achievements like the *Kepler conjecture* and *odd order theorem* which dealt with complex theorems about simpler objects
- References:
 - Commelin, Johan et al. (2022). “Completion of the Liquid Tensor Experiment.” Lean Community Blog, July 15, 2022. Available at: [Completion of the Liquid Tensor Experiment](#) .
 - Hales, Thomas C. et al. (2022). The Liquid Tensor Experiment covered in *Nature* and *Quanta Magazine*.

Embarrassing Failures

This is a sample – not a survey – of two *embarrassing failures* with CASs.

- (1) **with *Mathematica*:** A disturbing example involving *Mathematica* was reported by Durán, Pérez, and Varona (2014). The authors accidentally discovered that *Mathematica* not only computes determinants of integer matrices incorrectly, but also produces different results if one evaluates the same determinant twice.

Embarrassing Failures

This is a sample – not a survey – of two *embarrassing failures* with CASs.

- (1) **with *Mathematica***: A disturbing example involving *Mathematica* was reported by Durán, Pérez, and Varona (2014). The authors accidentally discovered that *Mathematica* not only computes determinants of integer matrices incorrectly, but also produces different results if one evaluates the same determinant twice.
- (2) **with *Maple***: The *Maple Special Functions Defect (2019-2022)*: In a comprehensive verification study published in 2022, Greiner-Petter et al. tested the **Digital Library of Mathematical Functions** (DLMF), systematically against both *Mathematica* and *Maple*. The evaluation techniques discovered two errors in the DLMF and one defect in *Maple*. This *Maple* defect was related to special function computations, an area where CASs are relied upon heavily by physicists, engineers, and applied mathematicians.

► SKIP THE NOTES

► SKIP to last slide

NOTES for slide 20:

- The error with *Mathematica* was discovered by comparing *Mathematica*'s calculations with those of *Maple*. To determine which of the two CASs was at fault, they had to run both on multiple randomly generated inputs; they did not have at their disposal formally specified conditions under which the CASs can be safely used and return outputs with correctness guarantees.
- More disturbing still than an error whose source could not be identified or located (*Mathematica* and *Maple* are not open-source) was the fact that an earlier release (*Mathematica 7*) did not show the error, while later releases (*Mathematica 9* and *Mathematica 10*) did. This regression represents a fundamental failure of software engineering: improvements or additions to the codebase introduced bugs into previously correct functionality, and the absence of comprehensive formal verification meant these bugs went undetected until users encountered them in practice.
- This example illustrates the architectural differences between CASs and IPAs discussed earlier. In an IPA with an LCF-style architecture, the trusted kernel ensures theorems are correct by construction – bugs in peripheral code cannot compromise soundness. By contrast, CASs like *Mathematica* have no such architectural separation between trusted and untrusted code, meaning that changes anywhere in the codebase can introduce silent errors that produce incorrect results without warning. The opacity of closed-source systems compounds this problem: users cannot inspect the code and must resort to empirical testing across multiple systems – exactly the kind of unreliable verification that formal methods were designed to eliminate.

NOTES for slide 20:

- The study involving [Maple](#) was particularly significant because it involved translating 4,078 semantic LaTeX DLMF formulae to native CAS representations and then applying an automated scheme for symbolic and numerical testing and verification. The researchers developed a translation tool called [LaCAsT](#) that enabled systematic comparison between the mathematical library and the CAS implementations.

What makes this example particularly revealing is the scale of the verification effort: of 3,474 test cases, numeric evaluation failed for 1,784 cases. In the evaluation process, 655 computations timed out and 180 failed due to errors in [Mathematica](#). This demonstrates that CAS reliability issues are not isolated incidents but rather systematic problems affecting a substantial portion of mathematical computations.

The fact that this defect in [Maple](#) – a mature CAS that has been commercially developed since the 1980s – was only discovered through systematic automated verification in 2019, underscores a fundamental problem: without formal specifications and soundness guarantees, users have no way to know which computations are trustworthy. The closed-source nature of [Maple](#) (like [Mathematica](#)) means users cannot inspect the implementation to understand the source of errors or the conditions under which the system can be safely used.

- Reference for the [Mathematica](#) failure:
 - Durán, Antonio J., Mario Pérez, and Juan L. Varona (2014). "The Misfortunes of a Trio of Mathematicians Using Computer Algebra Systems. Can We Trust in Them?" *Notices of the American Mathematical Society*, 61(10):1249–1252. [Also available as arXiv preprint arXiv:1312.3270]
- Reference for the [Maple](#) failure:
 - Greiner-Petter, André, Howard S. Cohl, Abdou Youssef, Moritz Schubotz, Avi Trost, Rajen Dey, Akiko Aizawa, and Bela Gipp (2022). "Comparative Verification of the Digital Library of Mathematical Functions and Computer Algebra Systems." In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2022)*, Lecture Notes in Computer Science, vol. 13243, pp. 87–105. Springer. DOI: 10.1007/978-3-030-99524-9.5. [Also available as arXiv preprint arXiv:2201.09488]

Not All Parts of Mathematics Have Profited Equally

from *Computer Algebra Systems & Automated Theorem Provers & Interactive Proof Assistants*

A rough approximation of the difference between CASs, ATPs and IPAs:

- ▶ CASs are **super calculators**,
mostly of numbers, derived from equations and formulas.
- ▶ ATPs are **super search engines**,
mostly of patterns, similarities, and formal proofs, built from axioms and deduction rules.
- ▶ IPAs are . . . mostly **super search engines**,
but some recent extensions of IPAs can also invoke CASs – and thus act as **super calculators** – as well as invoke ATPs.

Not All Parts of Mathematics Have Profited Equally

from *Computer Algebra Systems & Automated Theorem Provers & Interactive Proof Assistants*

A rough approximation of the difference between CASs, ATPs and IPAs:

- ▶ CASs are **super calculators**,
mostly of numbers, derived from equations and formulas.
- ▶ ATPs are **super search engines**,
mostly of patterns, similarities, and formal proofs, built from axioms and deduction rules.
- ▶ IPAs are . . . mostly **super search engines**,
but some recent extensions of IPAs can also invoke CASs – and thus act as **super calculators** – as well as invoke ATPs.

This difference – between **super calculator** and **super search engine** – is reflected by the parts of mathematics that have most profited from CASs, ATPs, and IPAs, which are not the same parts for all three categories of software systems.

▶ SKIP THE NOTES

▶ SKIP to last slide

NOTES for slide 22:

- Below is a saying often attributed to **Alan Bundy** (mathematical logician), sometimes to **Tim Gowers** (Fields Medalist combinatorialist), sometimes to **Bruno Buchberger** (mathematician and logician, creator of the theory of Gröbner bases)
 - here paraphrased and expanded a little:
 - ▶ *automated theorem provers and proof assistants are **super search engines** – of patterns, of similar formalisms, of formal proofs built from axioms and deduction rules, in a vast space of logical possibilities to find a valid sequence of inferences constituting a rigorous proof, be it formal or semi-formal,*
 - ▶ *computer algebra systems are **super calculators** – mostly of numbers, derived from equations and formulas, excelling at performing complex, predefined algorithmic computations, such as symbolic integration and derivation, and matrix manipulation.*

Not All Parts of Mathematics Have Profited Equally

from *Computer Algebra Systems & Automated Theorem Provers & Interactive Proof Assistants*

Differences between parts of mathematics where IPAs excel come through when examining their respective libraries – taking the library `mathlib4` of **Lean 4** as a reference point:

- ▶ **Mathlib** (specifically `mathlib4`), comprehensive, user-maintained, community-driven library of formalized mathematics for Lean 4. It acts as the primary repository for definitions, theorems, and tactics, providing essential foundational knowledge for research-level mathematics and programming within Lean 4.
- ▶ **MathComp** (the Mathematical Components library of **Coq/Rocq**), primary, large-scale equivalent to `mathlib4`, often utilized for algebra and formalizing the *top 100 theorems*.³ Other specialized libraries include **Coquelicot** (for real analysis with Coq/Rocq).
- ▶ **AFP** (the Archive of Formal Proofs by **Isabelle**), which acts as the central, comprehensive repository for formalized mathematics, serving as the equivalent to `mathlib4`'s broad scope.

▶ SKIP THE NOTES

▶ SKIP to last slide

³ **The Top 100 Theorems** – a rather arbitrary list of theorems, most (not all) of which are elementary, used for comparing IPAs.

NOTES for slide 23:

The three libraries exhibit similar coverage gaps, but keep in mind that these are constantly growing libraries. **Mathlib** started in 2017; **MathComp** has been regularly released since 2008, though many of its components had been created in earlier years; and **AFP** was founded in 2004.

- Perhaps the most apparent gaps in the three libraries are in *algebraic topology*, *modern algebraic geometry*, *differential geometry*, and perhaps other areas of math which I didn't inspect.
- The three libraries exhibit different strengths – this is certainly a subjective opinion, reflecting my own intellectual background. The three are shaped by their respective histories and philosophical orientations:
 - ▶ **Mathlib** seems to have a comprehensive undergraduate coverage of many (most?) areas of math outside computer science, perhaps reflecting that many (most?) of its contributors are mathematicians outside computer science.
 - ▶ **MathComp** comes across as a powerhouse in areas of *finite algebra* and *constructive mathematics*, and also strong in areas of *computer science* with most of its maintainers being computer scientists.
 - ▶ **AFP** appears broader and more balanced than the other two, with many verification results in areas of *computer science*, perhaps reflecting its longer history and its maintainers (more computer scientists than mathematicians).

Not All Parts of Mathematics Have Profited Equally

from *Computer Algebra Systems & Automated Theorem Provers & Interactive Proof Assistants*

Cartoon Presentation:

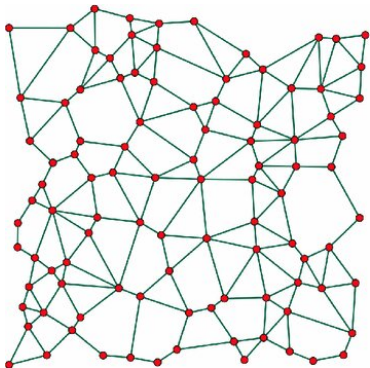
What IPAs can do as *super search engines*.

Suppose we have the following situation
an unboundedly large (possibly infinite) network of connections,
represented by a graph where each connection is an edge

Not All Parts of Mathematics Have Profited Equally

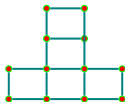
from *Computer Algebra Systems & Automated Theorem Provers & Interactive Proof Assistants*

Small Portion of Network X with +3,000,000 Nodes



Is it possible to embed Y in X ?

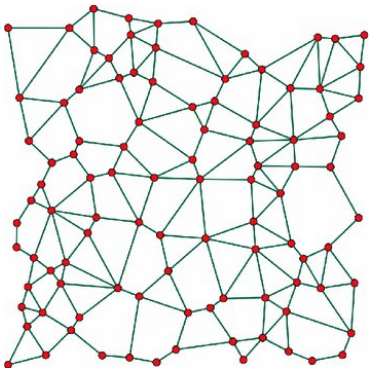
Pattern Y with 12 Nodes



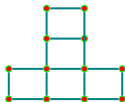
Not All Parts of Mathematics Have Profited Equally

from *Computer Algebra Systems & Automated Theorem Provers & Interactive Proof Assistants*

Small Portion of Network X with +3,000,000 Nodes



Pattern Y with 12 Nodes



Is it possible to embed Y in X ?

World-Class Mathematician

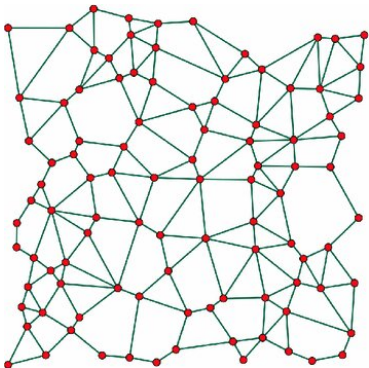


I can't answer, it's beyond my ability!

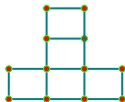
Not All Parts of Mathematics Have Profited Equally

from *Computer Algebra Systems & Automated Theorem Provers & Interactive Proof Assistants*

Small Portion of Network X with +3,000,000 Nodes



Pattern Y with 12 Nodes



Is it possible to embed Y in X ?

World-Class Mathematician

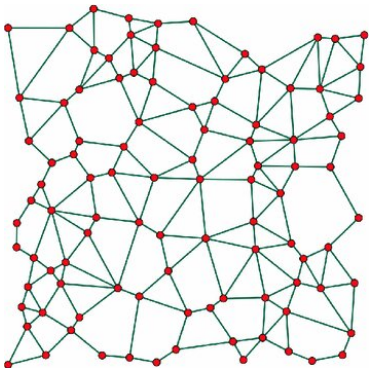


Let me Google for a software that may help . . .

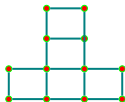
Not All Parts of Mathematics Have Profited Equally

from *Computer Algebra Systems & Automated Theorem Provers & Interactive Proof Assistants*

Small Portion of Network X with +3,000,000 Nodes



Pattern Y with 12 Nodes



Is it possible to embed Y in X ?

World-Class Mathematician

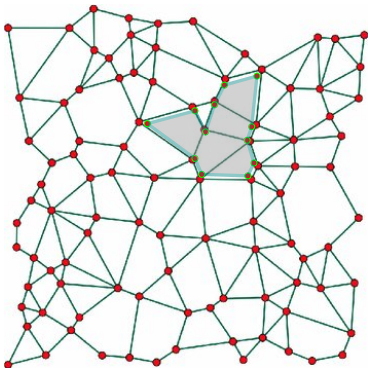


I think this software package will help . . .

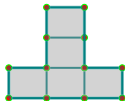
Not All Parts of Mathematics Have Profited Equally

from *Computer Algebra Systems & Automated Theorem Provers & Interactive Proof Assistants*

Small Portion of Network X with +3,000,000 Nodes



Pattern Y with 12 Nodes



Is it possible to embed Y in X ?

World-Class Mathematician



Great! My problem is solved: The answer is YES!

► SKIP to last slide

ATP Successes Across Mathematical Domains (1/2)

Where Automated Theorem Provers Have Excelled

TOP TIER: Overwhelming Success

1. Propositional and Boolean Logic (SAT)

- Modern SAT solvers handle millions of variables
- Applications: hardware verification, planning, scheduling
- **References:**
 - Biere et al. (2009). *Handbook of Satisfiability*. IOS Press.
 - Marques-Silva & Sakallah (1999). GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers* 48(5).

2. Linear Arithmetic

- Systems of linear equations and inequalities
- SMT solvers use Simplex, Fourier-Motzkin elimination
- Critical for program verification
- **References:**
 - Dutertre & de Moura (2006). A linear-arithmetic solver for DPLL(T). *CAV 2006*, LNCS 4144.
 - King & Barrett (2020). Effective bit-width and under-approximation. *FMCAD 2020*.

ATP Successes Across Mathematical Domains (1/2, cont.)

Where Automated Theorem Provers Have Excelled

3. Bit-Vector Arithmetic

- Models machine-level operations (AND, XOR, shifts)
- Essential for hardware and low-level software verification
- **References:**
 - Brummayer & Biere (2009). Boolector: An efficient SMT solver for bit-vectors and arrays. *TACAS 2009*, LNCS 5505.
 - Niemetz et al. (2020). Solving quantified bit-vectors using invertibility conditions. *CAV 2018*, LNCS 10981.

MIDDLE TIER: Moderate Success

4. Fragments of Number Theory

- **Presburger arithmetic** (addition, no multiplication): decidable
- Linear Diophantine equations
- General number theory much harder (see next slide)
- **References:**
 - Cooper (1972). Theorem proving in arithmetic without multiplication. *Machine Intelligence 7*.
 - Pugh (1991). The Omega test. *Communications of the ACM* 34(8).

ATP Successes Across Mathematical Domains (2/2)

Moderate Success and Where ATPs Have Struggled

MIDDLE TIER: Moderate Success (continued)

5. Finite Model Finding

- Finding counterexamples in finite domains
- Useful in algebra for small counterexamples to conjectures
- **References:**
 - Claessen & Sörensson (2003). Better techniques for MACE-style model finding. *CADE-19*.
 - Zhang & Zhang (1995). SEM: A system for enumerating models. *IJCAI 1995*.

6. First-Order Logic (Restricted)

- Resolution provers work on certain problem classes
- TPTP benchmark: 40–60% success on curated problems
- Highly domain-dependent

BOTTOM TIER: Disappointing to Dismal

7. Abstract Algebra (Groups, Rings, Fields)

- General theorems about infinite structures: very difficult
- Quantifier alternation causes exponential blowup
- *IPAs far more successful* (e.g., Feit-Thompson in Coq)

8. Analysis and Topology

- Real analysis, limits, continuity, differentiation
- ε - δ definitions with complex quantifiers
- ATPs almost completely ineffective; this is IPA territory

IPA Successes Across Mathematical Domains (1/2)

Where Interactive Proof Assistants Have Excelled

TOP TIER: Overwhelming Success

1. Abstract Algebra (Groups, Rings, Fields, Modules)

- Rich algebraic hierarchies, infinite structures, general theorems
- **Major achievements:**
 - Feit-Thompson Odd Order Theorem (Coq, 2012)
 - Substantial group theory, ring theory libraries
- **References:**
 - Gonthier et al. (2013). A machine-checked proof of the odd-order thm. *ITP 2013*, LNCS 7998.
 - Buzzard et al. (2020). Formalising perfectoid spaces. *CPP 2020*.

2. Mathematical Analysis (Real/Complex)

- Limits, continuity, derivatives, integrals, measure theory
- **Major achievements:**
 - Kepler Conjecture (Isabelle/HOL, 2014)
 - Substantial analysis libraries in all major IPAs
- **References:**
 - Hales et al. (2017). A formal proof of the Kepler conjecture. *Forum of Mathematics, Pi* 5.
 - Boldo et al. (2015). Coqelicot: A user-friendly library of real analysis for Coq. *Mathematics in Computer Science* 9(1).

IPA Successes Across Mathematical Domains (1/2, cont.)

Where Interactive Proof Assistants Have Excelled

3. Topology and Geometry

- Point-set topology, metric spaces, algebraic topology
- **Major achievements:**
 - Four Color Theorem (Coq, 2005)
 - Jordan Curve Theorem (multiple formalizations)
- **References:**
 - Gonthier (2008). Formal proof – The four-color theorem. *Notices of the AMS* 55(11).
 - Hales (2007). The Jordan curve theorem, formally and informally. *American Mathematical Monthly* 114(10).

MIDDLE TIER: Good Success

4. Type Theory and Logic Foundations

- Meta-theory of type systems, consistency proofs
- Naturally suited to systems that *are* type-theoretic
- **References:**
 - Coquand & Huet (1988). The calculus of constructions. *Information and Computation* 76(2-3).
 - Univalent Foundations Program (2013). *Homotopy Type Theory*. IAS.

IPA Successes Across Mathematical Domains (2/2)

Moderate Success and Where IPAs Have Struggled

MIDDLE TIER: Good Success (continued)

5. Number Theory

- Elementary and analytic number theory
- **Major achievements:**
 - Prime Number Theorem (Isabelle/HOL, 2005)
 - Dirichlet's theorem, many elementary results
- **References:**
 - Avigad et al. (2007). A formally verified proof of the prime number theorem. *ACM TOCL* 9(1).
 - Eberl (2018). Dirichlet's thm on primes in arithmetic progressions. *Archive of Formal Proofs*.

6. Combinatorics and Graph Theory

- Finite structures, counting arguments
- Four Color Theorem is the flagship example
- Good progress on basic results; harder for advanced theorems

BOTTOM TIER: Difficult or Disappointing

7. Probability Theory

- Measure-theoretic foundations are challenging
- Some success (Lean, Isabelle have libraries), but formalization burden is high
- Stochastic processes, advanced probability largely unexplored

8. Category Theory (Higher Categories)

- Basic category theory: formalized successfully
- Higher categories, ∞ -categories: extremely difficult
- Notation and abstraction mismatch with current systems

Lineages of Two Leading SAT/SMT Solvers: cvc5 & Z3

1950s: THE ROOTS

- Logic Theorist (1956)
- **DP/DPLL** (1960–62)

1970s: BREAKTHROUGH

- **Nelson-Oppen** (1979)
 - Theory combination
 - “Cooperating” in CVC

1990s: STANFORD

- **SVC** (1996)
 - Barrett, Dill, Levitt
 - Hardware verification

2000s: EVOLUTION

- CVC (2000–02)
- CVC Lite (2003)
- CVC3 (2007)

2010s–Present

- **CVC4** (2011–20)
- **cvc5** (2022–)
 - ML integration
 - Higher-order logic

1950s: THE ROOTS

- Logic Theorist (1956)
- **DP/DPLL** (1960–62)
 - Core SAT engine

1960s–70s: SPLIT

- Resolution (1965)
- **Nelson-Oppen** (1979)
 - Theory integration

1980s–90s: PREDECESSORS

- Shostak’s Method (1984)
- **Simplify** (Early 90s)
 - Detlefs, Nelson, Saxe
 - Simplex for arithmetic

2000s: MODERN SMT

- SVC → CVC → CVC3
- **Yices** (2006)
 - Leonardo de Moura
 - Pre-Z3 state-of-art

2007–Present

- **Z3** (2007–)
 - de Moura & Bjørner
 - Microsoft Research
 - CDCL + theories

Key: Both share DPLL and Nelson-Oppen foundations. CVC lineage emphasizes Stanford origins; Z3 incorporates Simplify/Yices innovations.

Lineages of Two Leading IPAs: Coq/Rocq & Lean 4

1930s–60s: FOUNDATIONS: Church's λ -calculus (1936), Curry-Howard (1958–69), Martin-Löf Type Theory (1972)

1970s–80s: AUTOMATH

- **Automath** (1967–80)
 - de Bruijn (Eindhoven)
 - First practical IPA

1984: BIRTH OF CoC

- **Calculus of Constructions**
 - Coquand & Huet (INRIA)
 - Unified types & proofs

1989–Present: COQ

- **Coq v1** (1989)
 - Implementing CoC
- **CIC** (1990s)
 - Calculus of Inductive Constructions
 - Added inductive types
- Major versions (2000s–)
 - Coq 8.x series
 - Four Color (2005)
 - Odd Order (2012)
- **Rocq** (2024)
 - Official rename
 - Same system, new name

1930s–60s: FOUNDATIONS: Church's λ -calculus (1936), Curry-Howard (1958–69), Martin-Löf Type Theory (1972)

1980s: CoC & IPAs

- Calculus of Constructions
- Early IPAs: Coq, HOL

2000s: de MOURA ERA

- **Z3** (2007)
 - Leonardo de Moura
 - SMT solver at MSR

2013–2021: LEAN 1–3

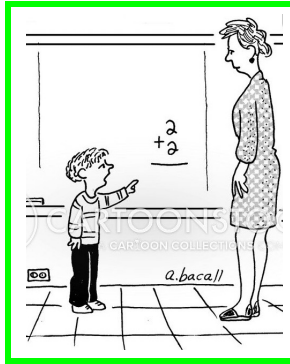
- **Lean** (2013)
 - de Moura at MSR
 - CIC-based, like Coq
- **Lean 2** (2015)
 - HoTT support
- **Lean 3** (2017)
 - mathlib begins
 - Community growth
 - Perfectoid spaces (2019)

2021–Present: LEAN 4

- **Complete rewrite**
 - Metaprogramming
 - Compiler optimizations
 - Self-hosting
 - Liquid Tensor (2022)

Key: Both rooted in type theory (Church, Curry-Howard, CoC). Coq: 35+ year evolution. Lean: rapid innovation with SMT insights.

Thank you!



Rather than learn how to solve that problem,
is it not better that we learn how to operate software that can solve it?