Why Proof Assistants Are Suddenly Practical

Proof assistants have a reputation for being powerful but slow: the kind of thing you reach for when you want absolute certainty, and you're willing to pay for it in time and friction. Terence Tao's recent experiment is interesting because it attacks that tradeoff directly, aiming for a workflow where "formal-ish" verification is fast enough to be used as part of everyday problem solving, not just as a final archival step.

A proof assistant that feels like a REPL

Tao describes iterating from a proof-of-concept verifier into a more flexible, extensible proof assistant that deliberately mimics Lean in key ways, while being implemented in Python and powered by SymPy. The interaction model is intentionally simple: run Python in interactive mode, load an exercise, and drive the proof forward by applying tactics to the current proof state.

The examples make the point quickly. A goal is presented as a structured state (variables, hypotheses, and a target), and a single tactic like `Linarith()` can close a linear arithmetic goal, with an optional verbose mode that exposes the underlying feasibility check. It's the same idea as modern theorem provers: humans provide high-level moves, the system does the tedious bookkeeping.

Why this matters: the "boring" middle of proofs

What makes this direction compelling is not that it replaces full formalization. It's that it tries to cover the annoying middle ground: proofs that are conceptually straightforward but algebraically messy, where most human error hides. Tao explicitly leans toward semi-automated interactive proofs, where the user provides tactics and the tool pushes calculations through until the goal is discharged.

This is also why the tool's focus on estimates is a good litmus test. Real analysis and asymptotics are full of manipulations that are easy to get wrong in small ways, and expensive to formalize end-to-end in a fully verified foundation. A lightweight assistant that can reliably certify these steps could be a practical bridge between pen-and-paper reasoning and fully formal proof libraries.

Asymptotics, SymPy, and a pragmatic foundation

The post gets especially fun when it moves from propositional/linear reasoning into asymptotic estimates. Tao sketches an approach where orders of magnitude (like `Theta(X)`) are represented in a way that plays nicely with SymPy's symbolic machinery, and then verified via a log-linear arithmetic solver. In other words: take a domain where humans often hand-wave, and build tactics that turn the hand-waving into checkable structure.

There's an important honesty here too. Tao notes the tool is not designed to be fully formally sound—Python and SymPy are not certified kernels—so the promise is not "trusted down to axioms." The promise is closer to: get something that works, build a corpus of tactic-driven proofs, and eventually translate or export certificates into a fully verified system (he mentions Lean as a target direction) once the workflow is stable enough to justify the cost.

The bigger picture

One way to read this is as a bet on proof tooling evolving like developer tooling. A strict, verified kernel is like a compiler backend; a flexible, interactive layer is like the frontend that makes humans productive. If proof assistants are going to matter outside of niche communities, they need to feel less like writing a second proof and more like debugging: iterative, inspectable, and fast enough that you actually use it.

Contributor: Alessandro Linzi