

Beyond QED:  
AI, Theorem Proving, and the Quest for  
Beautiful Proofs

Natarajan Shankar

Computer Science Laboratory  
SRI International  
Menlo Park, CA

April 10, 2026

- Artificial intelligence (AI) aims to solve problems that typically require human intelligence.
- AI techniques can be classified as algorithmic, symbolic, and statistical:
  - Theorem proving, or more broadly, automated reasoning, is a universal approach to symbolic AI.
  - Algorithmic approaches seek bespoke solutions to specific classes of problems, e.g., planning, perception, natural language understanding.
  - Statistical approaches fit models to data.
- The dramatic evolution of statistical AI, fuelled by the convergence of data and compute, opens up new opportunities for synergy between AI and Theorem Proving.
- How can we leverage statistical AI for the formalization of mathematics to go beyond QED to
  - 1 Efficiently and insightfully pinpoint fallacious reasoning in human or AI-generated proofs, and
  - 2 Assist in the production of beautiful formalizations and proofs that capture simple, repeatable abstract patterns of reasoning?

# Beautiful Formal Proofs

- In mathematics, beautiful proofs not only capture the inevitability of a claim in a compelling manner, they reveal patterns and methods that are broadly reusable.
- But, beauty in formalization? *Do we care?*
- Yes, we do. A beautiful formalization or proof captures the crux of the argument with uniformity and generality, simple and direct inference steps, and minimal irrelevance or surprise.
- Formalization is the process of explaining mathematics to a machine and to other humans.  
*Science is what we understand well enough to explain to a computer. Art is everything else we do. —Donald Knuth*
- Formalization is hard, and ugly formalizations exacerbate the pain and build technical debt.

# Why Beautiful Proofs?

*The development of mathematics toward greater precision has led, as is well known, to the formalization of large tracts of it, so one can prove any theorem using nothing but a few mechanical rules.*

*—Kurt Gödel*

A beautiful proof delivers value beyond merely the validity of the claim that it established — it leverages language and mechanization to create an architecture for scaling the abilities of humans.

*In mathematics, you don't understand things. You just get used to them.*

*John von Neumann*

# What Makes a Proof Beautiful?

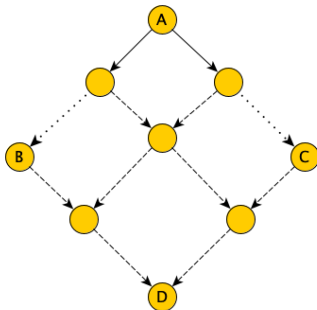
- Mathematics, many believe, is inherently beautiful.
- Erdős talked about Proofs from The Book where God wrote down the perfect proofs.
- In *Proofs from THE BOOK*, Aigner and Ziegler admit “We have no definition or characterization of what constitutes a proof from The Book ...”.
- A beautiful informal proof can seem appealing to the senses, but this attraction often doesn't survive formalization.
  - Dijkstra's proof of the Pigeonhole Principle: Given  $N + 1$  pigeons assigned to  $N > 0$  holes, the maximum number of pigeons per hole is at least the average, which is  $\frac{N+1}{N} > 1$ .*
- The above proof appears beautiful but brings in concepts like counting, maximum, and average that are more complicated than needed — all you want is that there is no injection from holes to pigeons!
- By going beyond QED, we can uncover deep value in proofs from their structural elegance.

# Some Personal Pre-History

- I started working on formalization in early 1982 at UT Austin under the supervision of Bob Boyer and J Moore.
- The goal was a complete formalization of Gödel's incompleteness theorem using the Boyer–Moore theorem prover **Thm**.
- Since this was the first nontrivial formalization attempted; there were no precedents nor roadmaps for this kind of undertaking.
- Failure was a very realistic option.
- Was this formalization beautiful? *Absolutely. The outcome would've been a failure otherwise.*
- The steps in the incompleteness proof contain many hidden delights including the verification of a SAT solver (with unsatisfiability proof objects).

# A Detour into Lambda Calculus

- In 1984, I paused work on the incompleteness theorem to take a brief detour to formalize the Church-Rosser theorem.
- After several failures, at Boyer's repeated suggestion, I decided one night to try using the de Bruijn representation for  $\lambda$ -terms.
- The proofs I did that night leading to the Diamond Property of Walks leverage the sheer elegance of the de Bruijn representation.



- By the mid-1980s, there were enough successful results to proclaim *Boyer's Theorem*:

*Large-scale formalization is no longer a major challenge.*

- Meaning: Getting to QED is a foregone conclusion, failure is unlikely, The focus is on demonstrating that formalization has *intrinsic* (clarity, insight) and *extrinsic* (reusability) value.
- From 1988 Notices of AMS: *Of all the sciences, mathematics is the one that has relied least on technology. Generations of mathematicians have practiced their art and communicated their ideas with only the simplest writing implements. The traditional mathematician perhaps views computer proof checkers to be unnecessary diversions that are unlikely to have any significant, positive impact on the field. . . . , I would like to suggest that proof checkers can have a useful role to play . . . in mathematical practice.*

# PVS and the Computational Sapir–Whorf Hypothesis

- In the 1980s, there were few challengers to the Boyer–Moore prover **Thm**, but SRI's EHDM system had an interesting combination of higher-order logic and SMT solving.
- However, these and other systems neglected the real pain point of formalization: 95% of your time is spent debugging erroneous definitions, conjectures, and proof attempts, and your neurons.
- They also largely ignored the role that the formal language plays in formalization.
- The latter point led to the observation of the *Computational Sapir–Whorf Hypothesis: Formal language influences formal thought*. (e.g., Hoare's invention of the recursive Quicksort was inspired by recursion in Algol60)
- PVS is explicitly designed as a medium for capturing beautiful, scalable formalizations by exploiting the synergy between language and mechanization.

# The Proof Lifecycle Beyond QED

The 1992 CADE paper introducing PVS captures the motivation for many of the design choices in PVS for supporting the scalable construction of robust proofs:

*Our experience with mechanical verification of complex designs and algorithms has led us to conclude that, just as with software, there is a lifecycle to a mechanically-checked proof. In the initial exploratory phase of proof development, we are mainly interested in debugging the specification and putative theorems, and in testing and revising the key, high-level ideas in the proof. An important requirement in this phase is early and useful feedback when a purported theorem is, in fact, false. Once the basic intuitions have been acquired and the formalization is stable, the proof checking enters a development phase where we take care of the details and construct the proof in larger leaps. Efficiency of proof development is a key requirement here. In the third, presentation phase, the proof is honed and polished for presentation in order to be scrutinized by the social process. Readability and intellectual perspicuity of the output is the goal here. The final phase is generalization where we carefully analyze the finished proof, weaken and generalize the assumptions, extract the key insights and proof techniques, and make it easier to carry out subsequent verifications of a similar nature. Maintenance is a special application of generalization, where we adapt a verification to slightly changed assumptions or requirements. Robustness of the proof procedure is a useful attribute here.*

# Proof Exploration: Cantor–Schroder-Bernstein Theorem

- Given injections  $f : A \rightarrow B$  and  $g : B \rightarrow A$ , there is a bijection  $h : A \leftrightarrow B$ .
- For any subset  $X$  of  $A$  (resp.,  $Y$  of  $B$ ), let  $f[X] \subseteq B$  (resp.  $g[Y] \subseteq A$ ) be the image of  $X$  w.r.t.  $f$  (resp.  $Y$  w.r.t.  $g$ ).
- Let  $\bar{X}$  be the complement of  $X$  in  $A$ . Define  $P(X) = \overline{g[f[X]]}$ .
- Since the image operator is monotone, and complement is antitone,  $P$  has a fixpoint  $X_0$ , i.e.,  $P(X_0) = X_0$ .
- If  $Y_0 = \overline{f[X_0]}$ , then  $R(Y_0) = Y_0$ , where  $R(Y) = \overline{f[g[Y]]}$ , since  $R(Y_0) = \overline{f[g[\overline{f[X_0]}]]} = \overline{f[P(X_0)]} = \overline{f[X_0]} = Y_0$ .
- Note that  $g[Y_0] = g[\overline{f[X_0]}] = \overline{X_0}$ .
- Since  $f$  is a bijection between  $X_0$  and  $\overline{Y_0}$  and  $g$  is a bijection between  $Y_0$  and  $\overline{X_0}$ , the bijection  $h$  can be defined as

$$h(x) = \begin{cases} f(x) & \text{for } x \in X_0 \\ g^{-1}(x) & \text{for } x \in \overline{X_0}. \end{cases}$$

- Initial proof completed in one sitting (no props) from a vague recollection of informal argument.

# Computational Sapir–Whorf in Action: N-Queens

- Define and verify a function that checks if there is a placement of  $N$  queens on an  $N \times N$  chessboard.
- This was a challenge problem in a 2014 Verified Software Competition (that I posed!).
- Rustan Leino, who participated in the competition, challenged me to verify an N-Queens search program.
- The formalization to follow was completed in less than an hour.
- This formalization is a good example of language and automation conspiring to yield an elegant, easy-to-prove formalization of a tricky problem.

# N-Queens in PVS

The nqueens theory is parametric in the number of rows/columns. A board maps each row  $i$  to the column  $j$  where a queen is placed. The extends predicate checks that two boards agree on queen positions in rows  $0 \dots i - 1$ .

```
nqueens  [N: nat ]  : THEORY
  BEGIN
    %Boards are just N-element arrays on the subrange 0..N-1.
    board : TYPE = [below(N)->below(N)]
    A, B, queen, new_queen: VAR board

    i, j, k: VAR upto(N)

    %Board queen extends A from point i onwards
    extends(i, A, queen): bool =
      (FORALL (j: below(i)): A(j) = queen(j))

    p: VAR [board -> bool]
    :
  END nqueens
```

# N-Queens in PVS

The `qlift?` operation on a predicate  $p$  and a lifted board  $x$  is critical. It provides semantics to search failure (bottom and success `up(y)`). A good extension  $B$  of  $A$  is an extension that satisfies the search predicate  $p$ .

```
%A lifted element of type T is either bottom or up(y), and qlift?(p)
%ensures that bottom is returned only when there is no board
%satisfying p, and up(y) is returned only when y satisfies p.
```

```
qlift?(p)(x : lift[board]): bool =
  CASES x OF
    bottom: (FORALL queen: NOT p(queen)),
    up(queen): p(queen)
  ENDCASES
```

```
%A good extension is one that satisfies p
good_extension?(i, A, p)(B): bool =
  (p(B) AND extends(i, A, B))
```

# N-Queens in PVS

search at row  $i$  goes through column positions until it finds one that can be extended to a full solution satisfying predicate  $p$ .

```
%search is a higher-order operation that iterates over the values
%of j assigned to position i of the board to check if f can be
%used to find a good extension

search((i: below(N)), A, p,
      (j | (FORALL (k: below(j), B):
            NOT good_extension?(i+1, A WITH [i:= k], p)(B))),
      (f: [B: board -> (qlift?(good_extension?(i+1, B, p)))]))
: RECURSIVE
(qlift?(good_extension?(i, A, p))) =
(IF j = N THEN bottom
 ELSE LET B = A WITH [i := j]
      IN CASES f(B) OF
          bottom: search(i, A, p, j+1, f),
          up(C): up(C)
      ENDCASES
ENDIF)
MEASURE N - j
```

scan looks for a position for the queen at row *i* invoking search. Note how the mutual recursion between scan and search is resolved through a continuation argument to search. The type annotations allow correctness to be derived from local type compatibility checks.

```
%scan iterates over the positions until a good board extending the
%input is found

scan(i, p)(queen): RECURSIVE (qlift?(good_extension?(i, queen, p)))
=
(IF i = N THEN IF p(queen) THEN up(queen) ELSE bottom ENDIF
 ELSE search(i, queen, p, 0, scan(i+1, p))
  ENDIF)
  MEASURE N - i
```

Finally, the `goodqueen?` predicate is defined as the search predicate. The proofs are all in the 21 TCC proof obligations, five of which are proved interactively in 5 to 10 steps each, mostly using `grind` and `friends`.

```
%findboard starts that scan at 0 to return a good board when
%one exists.
findboard(p): (qlift?(p)) =
  scan(0, p)(LAMBDA (i: below(N)): 0)

goodqueen?(queen): bool =
  (FORALL (i, j: below(N)): i /= j IMPLIES
    (queen(i) /= queen(j) AND
     (i - j /= queen(i) - queen(j)) AND
     (j - i /= queen(i) - queen(j))))

nqueen: (qlift?(goodqueen?)) = findboard(goodqueen?)
```

# Proof explanation with Claude

Me: *Can you explain how the attached PVS theory solves the N-queens problem?*

Claude: Certainly! The attached PVS theory `nqueenb` provides a functional, higher-order solution to the classic n-queens problem: placing  $N$  queens on an  $N \times N$  chessboard so that no two queens threaten each other.

⋮

`scan(i, p)(queen)`: Tries to extend a partial board `queen` (with queens placed in rows  $< i$ ) to a full solution. If  $i = N$ , returns the board (all rows filled). Otherwise, calls `search` to try all possible columns for row  $i$ .

`search(i, p, A, j, f)`: For a given partial board `A`, tries to place a queen in row  $i$  at column  $j$ , and recursively continues. If  $j = N$ , returns bottom (no solution found for this row). If placing at  $j$  is safe ( $p(B)(i)$ ), recursively tries to extend this board. Otherwise, tries the next column.

This is impressive, but the explanation is quite hand-wavy and misses the significance of the type-constrained lifted type and continuation parameter to `search`.

# Synergy between AI and Theorem Proving

- Mathematical problem solving requires a combination of soft reasoning (examples, explanations, motivation, analogies, similarities) coupled with hard reasoning (simplification, rewriting, constraint solving, induction, proof tactics).
- AI can help with all four phases (exploration, development, presentation, and maintenance):
  - ① **Exploration:** Auto-formalization maps loose informal presentations to precise formal ones exploiting affordances in the target language.
  - ② **Development:** Proof recommender systems help to guide proof construction, retrieve lemmas, assess progress, and diagnose failures.
  - ③ **Presentation:** Automatically clean up the formalization, streamline proofs, and generate structured documents with coherent explanations (auto-informalization).
  - ④ **Maintenance:** Refactor proofs to extract theories and proof strategies, and build out libraries.

- A 1999 paper from Sir Timothy Gowers entitled *Rough Structure and Classification* contains an eerily prescient dialog between a mathematician and an AI agent, followed by the depressing conclusion:

*...I expect computers to be better at proving theorems than humans in 2099. I do not feel happy about this, but I expect the impact on my own mathematical life to be entirely positive, because a semi-intelligent database of the kind I have described would be a wonderful resource and would take a great deal of drudgery out of research. There might even be a golden age where computers are good at exercises but not good at having deep insights. Then, instead of wasting a week not noticing that a hoped-for lemma had a simple counterexample, one could get the computer to check it. [H]owever, such a golden age, if it occurs, is unlikely to last for long. ...the work of the mathematician would be to simply learn how to use theorem-proving machines effectively and to find interesting applications for them. This would be a valuable skill, but hardly the pure mathematics as we know it today.*

# Where do we go from here?

- We might already be in Gowers' *Golden Age*: AI models are already doing well on math competitions, solving nontrivial open problems, and driving large-scale formalization.
- Big questions:
  - ① Will these models amplify human intelligence or replace it?
  - ② Will they even need proof assistants for ground truth?
  - ③ Are we on the verge of realizing Leibniz' dream of resolving disputes by collectively acknowledging ground truths, or are we slowly untethering ourselves from reality?
- Leibniz' hope for symbolic logic was that *"humanity would have a new kind of instrument increasing the powers of reason far more than any optical instrument has ever aided the power of vision"*.
- Can we find a way to use this technology (built mostly on our creative output) to expand *humanity's powers of reason*?

- We need to use AI to augment the formalization lifecycle to go beyond QED to
  - ① Create proofs, tools, and libraries that are easy to understand, use, and extend
  - ② Amplify human effort by helping novices rapidly gain new skills, and enable experts to tackle more complex problems.
- Delegating problem solving and proving entirely to AI might save labor but runs the risk of building technical debt, deskilling human talent, accelerating inequality, and drowning the world with AI slop.

*Beautiful formalizations help humans (and machines) get used to mathematics.*