Emily Riehl

Johns Hopkins University

A new paradigm for mathematical proof?

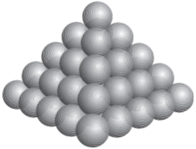
Mechanization and Mathematical Research

Recent developments in mathematics: problem solving

In 1998, Thomas Hales announced a proof of a 1611 conjecture of Johannes Kepler, via a "proof by exhaustion" involving the checking of many individual cases using a computer to solve linear programming problems. After four years, a panel of 12 referees reported they were 99% certain that the proof was correct, but could not check all the computer calculations.

THEOREM 1.1 (The Kepler conjecture). *No packing of congruent balls in Euclidean three space has density greater than that of the face-centered cubic packing.*

This density is $\pi/\sqrt{18} \approx 0.74$.



The unabridged version of the paper, which was published in 2005 in the *Annals of Mathematics*, came to 339 pages, with around 3 gigabytes of computer artifacts.

Recent developments in mathematics: problem solving

In 2016, Maryna Viazovska, resolved the analogous question in eight dimensions, proving that the optimal packing of spheres in 8-dimensional spaces positions their centers on the points of the E_8-lattice.

A week later, she and four collaborators generalized these results to dimension 24.

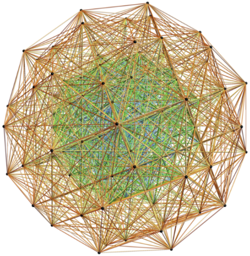
Annals of Mathematics **185** (2017), 991–1015
https://doi.org/10.4007/annals.2017.185.3.7

The sphere packing problem in dimension 8

By Maryna S. Viazovska

Abstract

In this paper we prove that no packing of unit balls in Euclidean space $\mathbb{R}^8$ has density greater than that of the E_8-lattice packing.



Visualization by J.G. Moxness (CC BY-SA 3.0).

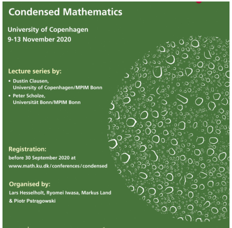
Recent developments in mathematics: theory building

In "The two cultures of mathematics" (the title alluding to C. P. Snow), Tim Gowers makes a distinction between:

> "mathematicians who regard their central aim as being to solve problems, and those who are more concerned with building and understanding theories."

Alexander Grothendieck's metaphorical rising sea evokes the power of theory building.

Some of the new ideas in the paper 356 paper of "Geometrization the local Langlands correspondence" by Laurent Fargues and Peter Scholze, to appear in *Asterisque*, involve a new framework to study objects that are both algebraic and topological in nature.



This framework, called condensed mathematics, has been developed over the course of several hundred pages of lecture notes by Dustin Clausen and Peter Scholze, which have been widely studied but remain unpublished.

A human-computer mathematical interaction

does the set of orthogonal matrices define a subspace of the vector space of all square mat

All Images Videos Shopping Web Forums Books More Tools

AI Overview

Learn more

Yes, the set of orthogonal matrices of a fixed size does define a subspace of the vector space of all square matrices of the same size.

Explanation:

Closure under addition:

If you add two orthogonal matrices, the resulting matrix will also be orthogonal, satisfying the closure property for addition in a subspace.

Closure under scalar multiplication:

Multiplying an orthogonal matrix by a scalar preserves the orthogonality condition, meaning the set is closed under scalar multiplication.

Contains the zero vector:

The zero matrix is considered an orthogonal matrix, ensuring the zero vector is included in the set.

Key point: An orthogonal matrix is a square matrix where the transpose is equal to its inverse, which guarantees that the rows (or columns) form an orthonormal set of vectors.

4.11: Orthogonality - Mathematics LibreTexts

Sep 16, 2022 — The span of these vectors is a subspace W of Rn. ... Thus a matrix is orthogonal if its rows (or...

Math LibreTexts

Orthogonal Matrix - an overview | ScienceDirect Topics

Any set of n nonzero orthogonal [orthonormal] vectors in ℝ n is an orthogonal [orthonormal] basis for ℝ n . ... If a...

ScienceDirect.com

Orthogonal Matrix: Definition, Types, Properties and Examples

A square matrix with real numbers or values is termed as an orthogonal matrix if its transpose is equal to the...

toppr.com

Do orthogonal matrices form a subspace?

My student asked Gemini:

"Does the set of orthogonal matrices define a subspace of the vector space of all square matrices?"

In the 2×2 case, orthogonal matrices have the form

$$\begin{bmatrix} s & -t \\ t & s \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} s & t \\ t & -s \end{bmatrix} \quad \text{with} \quad s^2 + t^2 = 1,$$

while square matrices have the form $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ for any real numbers a, b, c, and d.

The set of 2×2 matrices forms a vector space because you can add them and multiply them by a real number to get another 2×2 matrix.

This question asks whether the sum or scalar multiple of orthogonal matrices is again orthogonal and whether the zero matrix is an orthogonal matrix.

Gemini's illogical "thinking"

"Does the set of orthogonal matrices define a subspace of the vector space of all square matrices?"

Gemini answered "Yes" with the following explanation:

- "If you add two orthogonal matrices, the resulting matrix will also be orthogonal, satisfying the closure property for addition in a subspace."

  This is incorrect.

- "Multiplying an orthogonal matrix by a scalar preserves the orthogonality condition, meaning the set is closed under scalar multiplication."

  This is incorrect.

- "The zero matrix is considered an orthogonal matrix, ensuring the zero vector is included in the set."

  This is incorrect.

Despite the fact that Gemini gave correct definitions of subspace and of orthogonal matrices (as those matrices whose transpose is the inverse), it answered wrong.

Vibe proving

Is it true that an ∞-category is idempotent complete (in the ∞-categorical sense) if and only if its homotopy category is idempotent complete (in the 1-categorical sense)?

Show thinking

Yes, it is true that an ∞-category $\mathcal{C}$ is idempotent complete (in the ∞-categorical sense) if and only if its homotopy category $ho(\mathcal{C})$ is idempotent complete (in the 1-categorical sense).

Let's establish the definitions and then prove the equivalence.

...

Conclusion:
The two conditions are equivalent. The definitions of idempotent completeness are "homotopy-invariant" in a way that translates directly between the ∞-category and its homotopy category. This result is also confirmed by the fact that $h(Kar(\mathcal{C})) \cong Kar(h\mathcal{C})$, where $Kar(-)$ denotes the Karoubi envelope (idempotent completion) and $h(-)$ denotes taking the homotopy category. An (∞)-category is idempotent complete if and only if the canonical functor into its Karoubi envelope is an equivalence.

Your answer uses the "incoherent" definition of an idempotent in an ∞-category? Can you answer the same question using the coherent definition of an idempotent in an ∞-category?

Show thinking

You're right to press on the subtleties of "coherence" in ∞-categories. Let's clarify how these definitions are typically handled and why the previous conclusion still holds.

...

If a framework requires a more elaborate "coherent idempotent object" (e.g., an object in a category of idempotents, or an algebra for a certain monad), the definition of "idempotent completeness of $\mathcal{C}$" itself usually still boils down to the splitting condition for any e satisfying $e \circ e \simeq e$. The power of ∞-category theory is that such "weak" conditions, when interpreted within a coherently structured environment, often suffice and lead to robust theories.

Thus, the answer remains the same: **Yes, an ∞-category is idempotent complete (in the standard ∞-categorical sense) if and only if its homotopy category is idempotent complete (in the 1-categorical sense).** The "coherence" is sufficiently handled by the definitions used.

The omitted text contains a clearly written argument with a subtle error, the sort of optimistic mistake often made by human mathematicians. The conclusion is incorrect.

How do we defend the mathematical literature against vibe proving?

A lot of the mathematical text generated by top “reasoning” models is pretty good and all of it looks good.*

*Caveat: large language models tend to get even the simplest numerical calculations horribly wrong.

But pretty good is not good enough: one minor error could make an entire logical argument reach the wrong conclusion: true instead of false.

As a journal editor specializing in a subfield where papers tend to be long, technical, and difficult to referee, I am extremely concerned about AI generated mathematical “proofs.”

Proof as peer review

Why has mathematics largely avoided the replication crisis
that has confronted other fields?

Peer review in theory: careful refereeing should lead to an error-free publications.

Unfortunately, the mathematical literature contains famous mistakes as well as contradictory theorems.

Peer review in practice: in theory any proof should be reproducible by any reader — allowing the reader to understand for themselves why the result is true.

When papers have enough readers, mistakes are eventually caught.

Importantly: human mathematicians are careful in claiming they have a proof.

Proof as peer review?

But sometimes these ideals break down:

One Fields medalist was dismayed to find mistakes in his published, well-studied papers:

> *"A technical argument by a trusted author, which is hard to check and looks similar to arguments known to be correct, is hardly ever checked in detail."*
>
> *— Vladimir Voevodsky*

Another Fields medalist expressed doubts about a particular proof he had discovered — and also doubted that anyone else would check it:

> *"...while I was very happy to see many study groups on condensed mathematics throughout the world, to my knowledge all of them have stopped short of this proof. (Yes, this proof is not much fun...)"*
>
> *— Peter Scholze*

A new paradigm for mathematical proof?

THE EQUIVARIANT MODEL STRUCTURE ON CARTESIAN CUBICAL SETS

STEVE AWODEY, EVAN CAVALLO, THIERRY COQUAND, EMILY RIEHL, AND CHRISTIAN SATTLER

Abstract. We develop a constructive model of homotopy type theory in a Quillen model category that classically presents the usual homotopy theory of spaces. Our model is based on presheaves over the cartesian cube category, a well-behaved Eilenberg–Zilber category. The key innovation is an additional equivariance condition in the specification of the cubical Kan fibrations, which can be described as the pullback of an interval-based class of uniform fibrations in the category of symmetric sequences of cubical sets. The main technical results in the development of our model have been formalized in a computer proof assistant.

Contents

Date: June 27, 2024.

Software programs called computer proof assistants can certify the correctness of a mathematical proof that has been written in a precise formal language.

- Today such proofs are laboriously encoded by human mathematicians (formalization).
- In principle, generative AI could be trained to output text in a format that could be checked by a computer proof assistant (autoformalization).

Computer proof verification

```
{-
--------------------------------------------------------------------------------------------
Formalization of an equivariant cartesian cubical set model of type theory
--------------------------------------------------------------------------------------------

This formalization accompanies the article

  The equivariant model structure on cartesian cubical sets.
  Steve Awodey, Evan Cavallo, Thierry Coquand, Emily Riehl, & Christian Sattler.
  https://arxiv.org/abs/2406.18497

The contents of the formalization are outlined in Appendix A of the article.

The formalization defines a model of homotopy type theory inside an extensional type
theory augmented with a flat modality and axioms postulating *shapes* (among them an
*interval*) and a cofibration classifier. The results can in particular be externalized in
the category of cartesian cubical sets.

The code has been tested with Agda version 2.6.4.
The source is available at

  github.com/ecavallo/equivariant-cartesian

and there is an HTML interface at

  ecavallo.github.io/equivariant-cartesian

For reference (see the file equivariant.agda-lib in the source), the formalization is
compiled with the flags

  --with-K
  --cohesion --flat-split
  --no-import-sorts
  --rewriting

In particular, the --with-K flag enables axiom K (uniqueness of identity proofs), while
the --cohesion and --flat-split flags enable the flat modality (see the module
axiom.flat for more information).

-}
```

Our 87 page preprint is accompanied by a library of formalized proofs checked by the computer proof assistant `Agda`.

The paper, submitted to a journal in Sept. 2024, is still awaiting a referee report.

```
--↓ The equivariance condition on local filling structures associated to a shape
--↓ homomorphism σ : S → T. Filling an open box over T and then composing with σ should be
--↓ the same as composing the box with σ and then filling over S.

LocalEquivariance : {S T : Shape} (σ : Shape[ S , T ]) {A : ⟨ T ⟩ → Type ℓ}
  → LocalFillStr T A → LocalFillStr S (A ∘ ⟪ σ ⟫) → Type ℓ
LocalEquivariance σ liftT liftS =
  ∀ r box s →
  reshapeFiller σ (liftT (⟪ σ ⟫ r) box) .fill s .out
  ≡ liftS r (reshapeBox σ box) .fill s .out

Equivariance : {S T : Shape} (σ : Shape[ S , T ]) {Γ : Type ℓ} (A : Γ → Type ℓ')
  → FillStr T A → FillStr S A → Type (ℓ ⊔ ℓ')
Equivariance {T = T} σ {Γ} A fillT fillS =
  (γ : Γ ^ T) → LocalEquivariance σ (fillT γ) (fillS (γ ∘ ⟪ σ ⟫))

--↓ Definition of an equivariant fibration structure.

record FibStr {Γ : Type ℓ} (A : Γ → Type ℓ') : Type (ℓ ⊔ ℓ') where
  constructor makeFib
  field
    --↓ We have a filling structure for every shape.

    lift : (S : Shape) → FillStr S A

    --↓ The filling structures satisfy the equivariance condition.

    vary : ∀ S T (σ : Shape[ S , T ]) → Equivariance σ A (lift T) (lift S)
```

The main definition takes just a few lines to encode ⇝

What are computer proof assistants?

A computer proof assistant or interactive theorem prover — such as `Agda`, `HOL Light`, `Isabelle`, `Lean`, `Mizar`, or `Rocq` (née `Coq`) — is a computer program that:

- knows the rules of a logical formal system (e.g., a foundation for mathematics), which a trusted core program (the kernel) uses to check the correctness of proofs
- is programmed (via the elaborator) to interpret statements written in an expressive formal language (the vernacular) used to encode definitions, theorems, and proofs.

Formal proofs, written by a human user, are developed interactively with the computer.

- The mathematician inputs each line of their proof in a precise syntax.
- The computer checks that the logical argument supplied by the user produces a valid deduction of the claimed mathematical statement.

Aside: modern proof assistants often use a newer formal system — dependent type theory — in place of traditional Zermelo-Fraenkel set theory and first order logic.

How do computer proof assistants check proofs?

Aside: modern proof assistants often use a newer formal system — dependent type theory — in place of traditional Zermelo-Fraenkel set theory and first order logic.

How does the computer verify that the supplied logical argument produces a valid deduction of the claimed mathematical statement?

Formally:

- The statement of a mathematical theorem defines a type.
- The proof purports to define an element of that type.
- The logical form of the type dictates which rules may be used to construct elements — introduction rules to construct elements and elimination rules to use elements.
- The proof can be parsed as an iterative application of logical rules, which are checked inductively against the claimed type: proof verification as "type-checking."

Example. The statement "there is no surjective function from $\mathbb{N}$ to $\mathfrak{P}(\mathbb{N})$" corresponds to the type: $(\forall f : \mathbb{N} \to \mathfrak{P}(\mathbb{N})), (\forall S : \mathfrak{P}(\mathbb{N}), \exists n : \mathbb{N}, f(n) = S) \to \bot$.

A valid proof may start by assuming the antecedents and deriving a contradiction.

Type-checking Cantor's diagonalization argument

$$\text{cantor} : (\forall f : \mathbb{N} \to \mathfrak{P}(\mathbb{N})), (\forall S : \mathfrak{P}(\mathbb{N}), \exists n : \mathbb{N}, f(n) = S) \to \bot$$

A proof starting "assume f is a surjection from $\mathbb{N}$ to $\mathfrak{P}(\mathbb{N})$" provides hypotheses

$f : \mathbb{N} \to \mathfrak{P}(\mathbb{N})$ and $h : (\forall S : \mathfrak{P}(\mathbb{N}), \exists n : \mathbb{N}, f(n) = S)$ with the goal to prove $\bot$.

The user defines $D :\equiv \{k : \mathbb{N} \mid k \notin f(k)\}$ and the proof assistant checks that $D : \mathfrak{P}(\mathbb{N})$. The user says "applying hypothesis h to D, there exists $d : \mathbb{N}$ so that $f(d) = D$" so the proof assistant checks that $h(D) : \exists n : \mathbb{N}, f(n) = D$ provides $d : \mathbb{N}$ and $s : f(d) = D$. The user says "we derive a contradiction by applying the classical tautology

$$t : (\forall P : \text{Prop}), (P \leftrightarrow \neg P) \to \bot$$

to the proposition $d \in D$." Now it remains to check the user's argument that

$$((d \in D) \to (d \notin D)) \wedge ((d \notin D) \to (d \in D)).$$

The proof assistant expects a pair of proofs, one for each implication, such as:

- "Assume $d \in D$. Then by $s : f(d) = D$, $d \in f(d)$ so $d \notin D$."
- "Assume $d \notin D$. Then by $s : f(d) = D$, $d \notin f(d)$ so $d \in D$." $\square$

A new paradigm for proof writing

Computer formalization is a new and not yet widely practiced method of interactively developing and communicating rigorous mathematical proofs using a computer proof assistant. The formalization must either:

- be entirely self contained, including formalizations of all prerequisite definitions and theorems statements[1]
- or may refer to a library of formalized mathematics.

To a human user of an interactive theorem prover, writing a formal proof feels like writing code in a programming language, but with useful real-time feedback:

- typos or conceptual mistakes may be pointed out by "type-checking errors"
- the proof assistant often communicates the standing assumptions and yet-to-be proven objectives midway through a complex proof.

Warning: the proof assistant cannot check whether the formalized definitions or theorem statements accurately capture the mathematical ideas intended by the user!

[1]It is often possible to "assume" or "admit" some theorems without proof — "sorry" — but even if proofs are omitted those theorems must be stated formally.

A formalized proof of a true theorem

To illustrate, we give a formal proof in Lean that symmetric matrices define a subspace.

```
import Mathlib.Data.Matrix.Basic
import Mathlib.Data.Real.Basic

open Matrix

variable {n : Type}

/-- A matrix is `symmetric` if its `i j` entry equals its `j i` entry. -/
def symmetric (A : Matrix n n ℝ) : Prop :=
  (i j : n) → A i j = A j i

/-- A proof that the subset of symmetric matrices is a subspace. -/
def SymmetricMatrixSubspace : Subspace ℝ (Matrix n n ℝ) := sorry
```

A matrix $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ is symmetric if $A_{12} = A_{21}$.

More generally, an $n \times n$ matrix A is symmetric if $A_{ij} = A_{ji}$ for all indices i and j.

A formalized proof of a true theorem

Lean's Infoview keeps track of assumptions and objectives at each stage of a proof.

```
import Mathlib.Data.Matrix.Basic
import Mathlib.Data.Real.Basic

open Matrix

variable {n : Type}

/-- A matrix is `symmetric` if its `i j` entry equals its `j i` entry. -/
def symmetric (A : Matrix n n ℝ) : Prop :=
  (i j : n) → A i j = A j i

/-- A proof that the subset of symmetric matrices is a subspace. -/
def SymmetricMatrixSubspace : Subspace ℝ (Matrix n n ℝ) where
  carrier := sorry
  add_mem' := sorry
  smul_mem' := sorry
  zero_mem' := sorry
```

```
▼SymmetricSubspace.lean:21:0
 ▼Expected type
   n : Type
   ⊢ Subspace ℝ (Matrix n n ℝ)
▶All Messages (4)
```

Lean automatically generates the proof obligations. To complete the proof, we must replace each "sorry" with code that satisfies Lean's proof checker.

A formalized proof of a true theorem

By filling in the carrier, we tell Lean that the elements of the subspace are the symmetric matrices.

```
import Mathlib.Data.Matrix.Basic
import Mathlib.Data.Real.Basic

open Matrix

variable {n : Type}

/-- A matrix is `symmetric` if its `i j` entry equals its `j i` entry. -/
def symmetric (A : Matrix n n ℝ) : Prop :=
  (i j : n) → A i j = A j i

/-- A proof that the subset of symmetric matrices is a subspace. -/
def SymmetricMatrixSubspace : Subspace ℝ (Matrix n n ℝ) where
  carrier := symmetric -- The elements of the subspace are symmetric matrices.
  add_mem' := by
    sorry
  smul_mem' := sorry
  zero_mem' := sorry
```

```
▼SymmetricSubspace.lean:16:4
 ▼Tactic state
  1 goal
   n : Type
   ⊢ ∀ {a b : Matrix n n ℝ}, a ∈ symmetric → b ∈ symmetric → a + b ∈ symmetric
▶All Messages (3)
```

Lean tells us that to prove closure under addition,
we must show that if A and B are symmetric, then $A + B$ is symmetric.

A formalized proof of a true theorem

Lean tells us that to prove closure under addition,

we must show that if A and B are symmetric, then $A + B$ is symmetric.

```
import Mathlib.Data.Matrix.Basic
import Mathlib.Data.Real.Basic

open Matrix

variable {n : Type}

/-- A matrix is `symmetric` if its `i j` entry equals its `j i` entry. -/
def symmetric (A : Matrix n n ℝ) : Prop :=
  (i j : n) → A i j = A j i

/-- A proof that the subset of symmetric matrices is a subspace. -/
def SymmetricMatrixSubspace : Subspace ℝ (Matrix n n ℝ) where
  carrier := symmetric -- The elements of the subspace are symmetric matrices.
  add_mem' := by -- A proof that if `A` and `B` are symmetric matrices, so is `A + B`.
    intro A B Asym Bsym i j
    sorry
  smul_mem' := sorry
  zero_mem' := sorry
```

▼SymmetricSubspace.lean:17:4

▼Tactic state

1 goal

```
n : Type
A B : Matrix n n ℝ
Asym : A ∈ symmetric
Bsym : B ∈ symmetric
i j : n
⊢ (A + B) i j = (A + B) j i
```

▶ All Messages (3)

Thus for symmetric matrices A and B and indices i and j,

we must show that $(A + B)_{ij} = (A + B)_{ji}$.

A formalized proof of a true theorem

By definition of matrix addition, $(A+B)_{ij} = A_{ij} + B_{ij}$ and $(A+B)_{ji} = A_{ji} + B_{ji}$.

```lean
import Mathlib.Data.Matrix.Basic
import Mathlib.Data.Real.Basic

open Matrix

variable {n : Type}

/-- A matrix is `symmetric` if its `i j` entry equals its `j i` entry. -/
def symmetric (A : Matrix n n ℝ) : Prop :=
  (i j : n) → A i j = A j i

/-- A proof that the subset of symmetric matrices is a subspace. -/
def SymmetricMatrixSubspace : Subspace ℝ (Matrix n n ℝ) where
  carrier := symmetric -- The elements of the subspace are symmetric matrices.
  add_mem' := by -- A proof that if `A` and `B` are symmetric matrices, so is `A + B`.
    intro A B Asym Bsym i j
    simp only [add_apply]
    sorry
  smul_mem' := sorry
  zero_mem' := sorry
```

```
▼SymmetricSubspace.lean:18:4
 ▼Tactic state
  1 goal
  n : Type
  A B : Matrix n n ℝ
  Asym : A ∈ symmetric
  Bsym : B ∈ symmetric
  i j : n
  ⊢ A i j + B i j = A j i + B j i
▶All Messages (3)
```

Thus for symmetric matrices A and B and indices i and j,

we must show that $A_{ij} + B_{ij} = A_{ji} + B_{ji}$.

A formalized proof of a true theorem

Since A and B are symmetric, $A_{ij} = A_{ji}$ and $B_{ij} = B_{ji}$ so this equation holds:

```
import Mathlib.Data.Matrix.Basic
import Mathlib.Data.Real.Basic

open Matrix

variable {n : Type}

/-- A matrix is `symmetric` if its `i j` entry equals its `j i` entry. -/
def symmetric (A : Matrix n n ℝ) : Prop :=
  (i j : n) → A i j = A j i

/-- A proof that the subset of symmetric matrices is a subspace. -/
def SymmetricMatrixSubspace : Subspace ℝ (Matrix n n ℝ) where
  carrier := symmetric -- The elements of the subspace are symmetric matrices.
  add_mem' := by -- A proof that if `A` and `B` are symmetric matrices, so is `A + B`.
    intro A B Asym Bsym i j
    simp only [add_apply]
    rw [Asym, Bsym]
  smul_mem' := sorry
  zero_mem' := sorry
```

```
▼SymmetricSubspace.lean:18:20
 ▼Tactic state
  No goals
 ▼Expected type
  n : Type
  ⊢ Subspace ℝ (Matrix n n ℝ)
▶All Messages (2)
```

Now Lean tells us that there are no goals!

So we may move on to the remaining proof obligations …

A formalized proof of a true theorem

```lean
import Mathlib.Data.Matrix.Basic
import Mathlib.Data.Real.Basic

open Matrix

variable {n : Type}

/-- A matrix is `symmetric` if its `i j` entry equals its `j i` entry. -/
def symmetric (A : Matrix n n ℝ) : Prop :=
  (i j : n) → A i j = A j i

/-- A proof that the subset of symmetric matrices is a subspace. -/
def SymmetricMatrixSubspace : Subspace ℝ (Matrix n n ℝ) where
  carrier := symmetric -- The elements of the subspace are symmetric matrices.
  add_mem' := by -- A proof that if `A` and `B` are symmetric matrices, so is `A + B`.
    intro A B Asym Bsym i j
    simp only [add_apply]
    rw [Asym, Bsym]
  smul_mem' := by -- A proof that if `k ∈ ℝ` and `A` is a symmetric matrix, so is `k • A`.
    intro k A Asym i j
    simp only [smul_apply]
    rw [Asym]
  zero_mem' := by -- A proof that the zero matrix is symmetric.
    intro i j
    simp only [zero_apply]
```

▼SymmetricSubspace.lean:26:0

▼Tactic state

No goals

▼Expected type

```
n : Type
⊢ Subspace ℝ (Matrix n n ℝ)
```

▶All Messages (0)

A new paradigm for proof checking

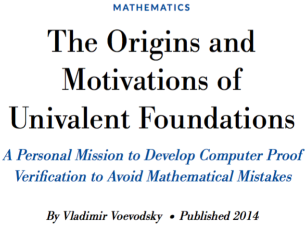
"A technical argument by a trusted author, which is hard to check and looks similar to arguments known to be correct, is hardly ever checked in detail."

— Vladimir Voevodsky

"...while I was very happy to see many study groups on condensed mathematics throughout the world, to my knowledge all of them have stopped short of this proof. (Yes, this proof is not much fun...)"

— Peter Scholze

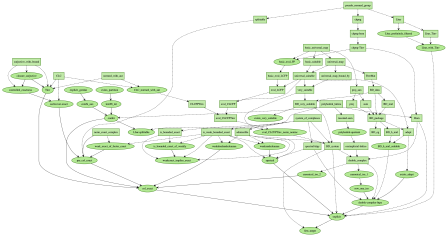
Voevodsky and Scholze both turned to computer formalization to resolve their doubts about the veracity of their own proofs.


MATHEMATICS

The Origins and Motivations of Univalent Foundations

A Personal Mission to Develop Computer Proof Verification to Avoid Mathematical Mistakes

By Vladimir Voevodsky • Published 2014



Large scale computer-verified proofs

When human referees failed to fully certify his proof of the Kepler conjecture, Hales launched a project to verify the result himself in a computer proof assistant. Eleven years later, the full proof was formally verified in the proof assistants `HOL Light` and `Isabelle`. The formalization was described in an accompanying 29 page paper with 22 authors.

A FORMAL PROOF OF THE KEPLER CONJECTURE

THOMAS HALES[1], MARK ADAMS[2,3], GERTRUD BAUER[4], TAT DAT DANG[5], JOHN HARRISON[6], LE TRUONG HOANG[7], CEZARY KALISZYK[8], VICTOR MAGRON[9], SEAN MCLAUGHLIN[10], TAT THANG NGUYEN[7], QUANG TRUONG NGUYEN[1], TOBIAS NIPKOW[11], STEVEN OBUA[12], JOSEPH PLESO[13], JASON RUTE[14], ALEXEY SOLOVYEV[15], THI HOAI AN TA[7], NAM TRUNG TRAN[7], THI DIEP TRIEU[16], JOSEF URBAN[17], KY VU[18] and ROLAND ZUMKELLER[19]

Last year, Kevin Buzzard launched a project to verify a modern proof of Fermat's last theorem—that there are no positive integer solutions to the equation $x^n + y^n = z^n$ for $n \geq 3$—in the computer proof assistant `Lean`, motivated in part by the question: "is there any one person who completely understands a proof of Fermat's Last Theorem?"

Moral: proofs at the frontier of mathematics are formalizable

...but only with monumental human effort via large-scale collaborations.

Interactive theorem proving, in pursuit of greater rigour and clarity

The practice of explaining a mathematical proof to a computer requires absolute precision, in particular regarding the exact definitions of mathematical terms. In my experience at least, this level of pedantry is both deeply frustrating and unexpectedly seductive, making it easier to achieve and sustain a flow state of deep focus.

- There is no point in attempting to write anything if you aren't thinking perfectly clearly because it will be rejected by the computer proof assistant.
- Paradoxically, this much steeper demand of my attention makes it easier for me to achieve that level of concentration.

The interactions with the computer proof assistant also activate reward mechanisms.

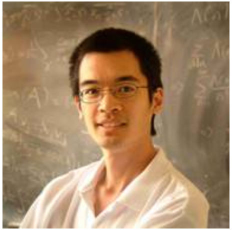
- During a typical research day, I make no quantifiable progress towards proving anything. But in the practice of formalization, the user periodically asks the proof assistant whether what is done so far is correct. When it says yes, this feels great.

Practitioners sometimes describe formalizing as a gamification of mathematical research.

There's more to mathematics than rigour and proofs

But the demands of greater rigour enforced by formalization runs counter to the vision of mathematics presented in a famous blog post of Terry Tao:

> *It is of course vitally important that you know how to think rigorously, as this gives you the discipline to avoid many common errors and purge many misconceptions. Unfortunately, this has the unintended consequence that "fuzzier" or "intuitive" thinking (such as heuristic reasoning, judicious extrapolation from examples, or analogies with other contexts such as physics) gets deprecated as "non-rigorous" ...*



> *The point of rigour is not to destroy all intuition; instead, it should be used to destroy bad intuition while clarifying and elevating good intuition. It is only with a combination of both rigorous formalism and good intuition that one can tackle complex mathematical problems; one needs the former to correctly deal with the fine details, and the latter to correctly deal with the big picture.*

On computer-verified proof and progress in mathematics

In a famous 1994 essay "On proof and progress in mathematics," Bill Thurston opens with the question

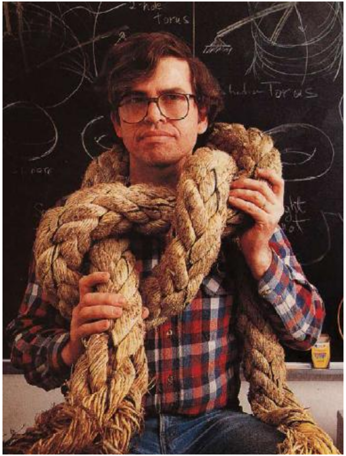
> "What is it that mathematicians accomplish?"

which he rephrases in a more leading form as

> "How do mathematicians advance human understanding of mathematics?"

He continues:

> "This question brings to the fore something that is fundamental and pervasive: that what we are doing is finding ways for *people* to understand and think about mathematics. "



On human understanding of mathematics

Thurston's point is that mathematics is about more than just definitions, theorems, and proofs or getting to the "right answers":

> *The rapid advance of computers has helped dramatize this point, because computers and people are very different. For instance, when Appel and Haken completed a proof of the 4-color map theorem using a massive automatic computation, it evoked much controversy. I interpret the controversy as having little to do with doubt people had as to the veracity of the theorem or the correctness of the proof. Rather, it reflected a continuing desire for human understanding of a proof, in addition to knowledge that the theorem is true.*
>
> *On a more everyday level, it is common for people first starting to grapple with computers to make large-scale computations of things they might have done on a smaller scale by hand. They might print out a table of the first 10,000 primes, only to find that their printout isn't something they really wanted after all. They discover by this kind of experience that what they really want is usually not some collection of "answers"—what they want is understanding.*

A new paradigm for mathematical proof?

Right now computer proof assistants are very difficult for the working mathematician to use in an expedient way:

the gap between the "post-rigorous" proofs in the mathematical literature and fully formal mathematics in enormous.

However, with advances in:

- domain-specific foundation systems, which bring the formal definitions of the central mathematical objects much closer to mathematician's intuitions and
- AI-powered autoformalization

it seems entirely plausible that the everyday practice of mathematics will change dramatically in the coming decades.

How would this shift affect human understanding of mathematics — the real point of what we do?