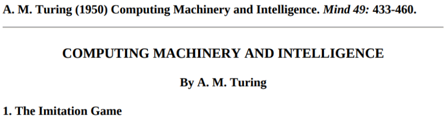
Emily Riehl

Johns Hopkins University

Can machines think logically?

APL Colloquium

Can machines think?



A. M. Turing (1950) Computing Machinery and Intelligence. *Mind 49:* 433-460.

COMPUTING MACHINERY AND INTELLIGENCE

By A. M. Turing

1. The Imitation Game

In a famous 1950 paper entitled "Computing machinery and intelligence," Alan Turing opens with the question

"Can machines think?"

Sidestepping the task of giving precise meaning to the term "think," Turing instead describes what he calls "the imitation game" and asks whether an interrogator in another room would be able to distinguish between a man and machine impersonating a man through a series of typewritten questions.[a]

[a] Actually, the imitation game is much queerer than this: Turing asks whether a machine would do better than a man at impersonating a woman.

The Turing test

Turing predicted that in 50 years time, machines would be able to pass what is now known as the Turing test.

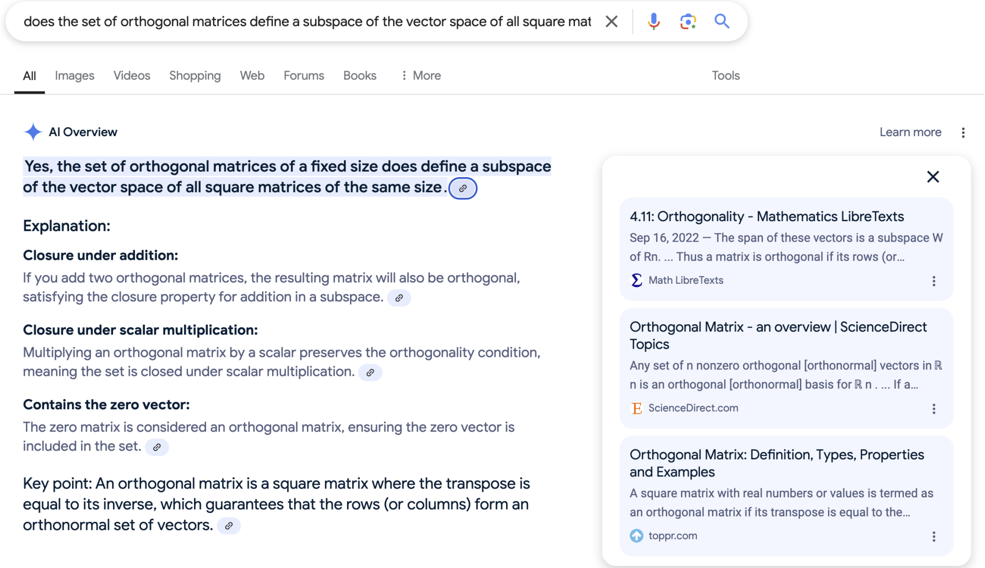
This is arguably the case for today's large language models.

But I want to ask a different question:

"Can machines think logically?"

In realms, such as mathematics, where logical reasoning can distinguish what is true from what is false, can generative AI reliably give us the correct answer?

A question from my practice final in Math 201: Linear Algebra



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

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."

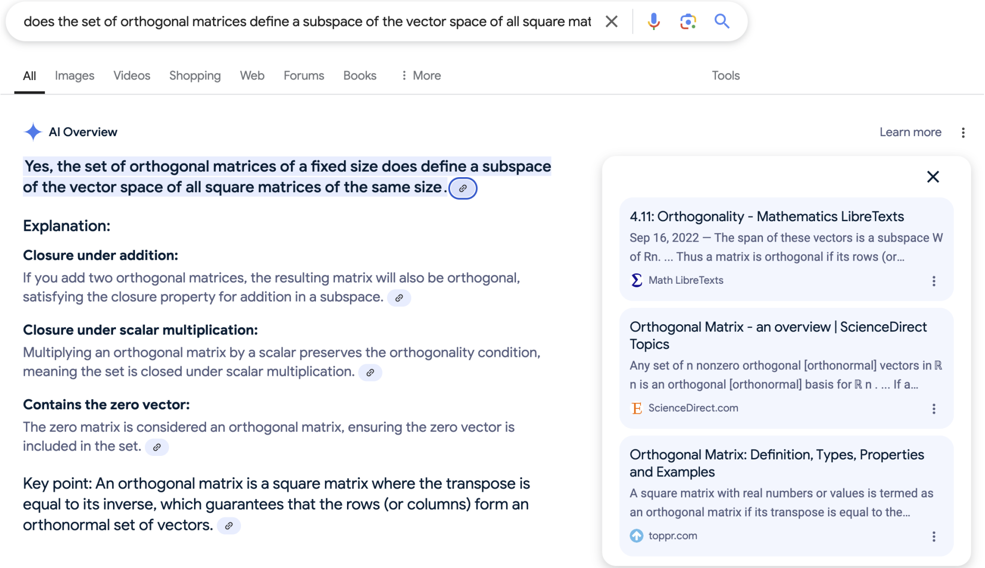
  This is incorrect.

- "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.

Gemini's illogical "thinking"



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

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...

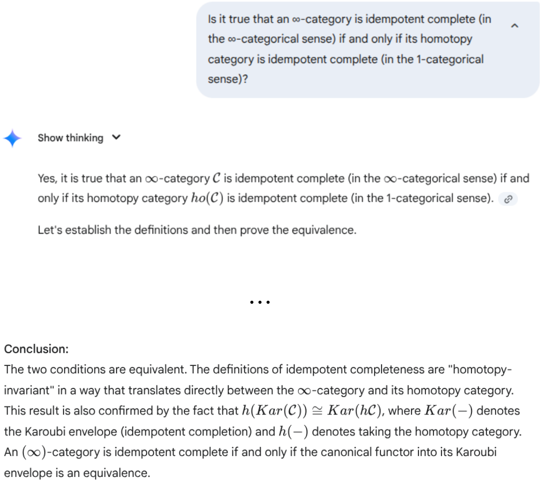
ScienceDirect.com

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

Vibe proving



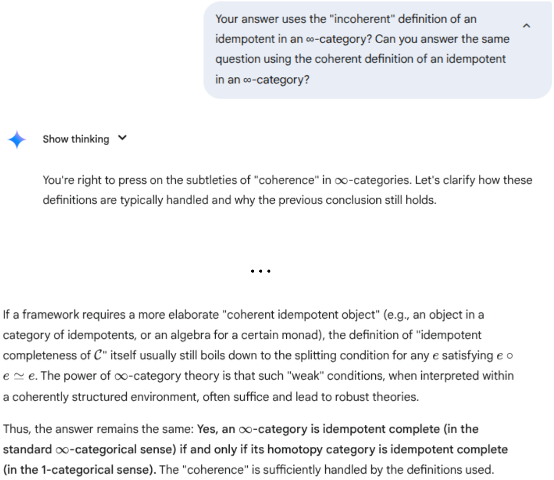
Is it true that an ∞-category is idempotent complete (in the ∞-categorical sense) if and only if its homotopy category is idempotent complete (in the 1-categorical sense)?

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?

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.

Can machines think logically? — Not yet

In realms where correctness of an answer is provable and accuracy is required, generative AI is not yet reliable — but it could be in the future.

Currently existing software programs called computer proof assistants can

- certify the correctness of a mathematical proof (computer formalization) or
- verify that a software program satisfies a desired specification (formal methods).

In principle, a generative AI could be designed to output text in a format that it could be checked by a computer proof assistant — a process known as autoformalization.

With such a protocol, AI-generated outputs in certain domains can be formally verified before being put into public use.

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) in which a user writes their definitions, theorems, and proofs.

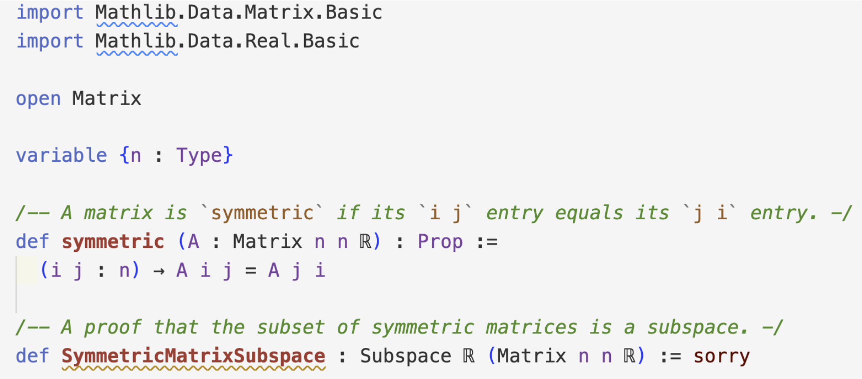
To a human user of an interactive theorem prover, writing a formal proof feels like writing code in a programming language, but with useful real-time feedback:

- typos 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.

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.

A formalized proof of a true theorem

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



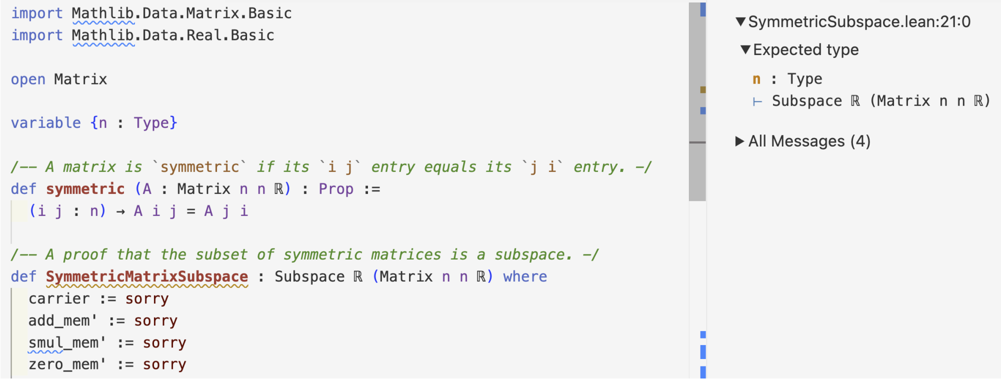
```
import Mathlib.Data.Matrix.Basic
import Mathlib.Data.Real.Basic

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

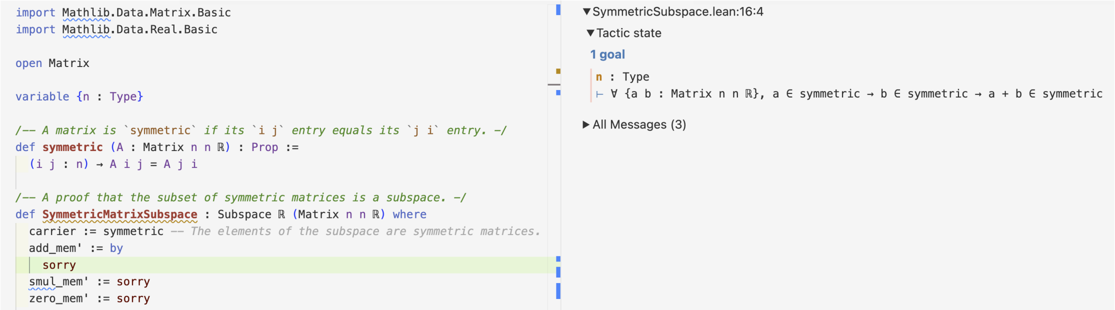
```
n : Type
⊢ Subspace ℝ (Matrix n n ℝ)
```

▶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
```

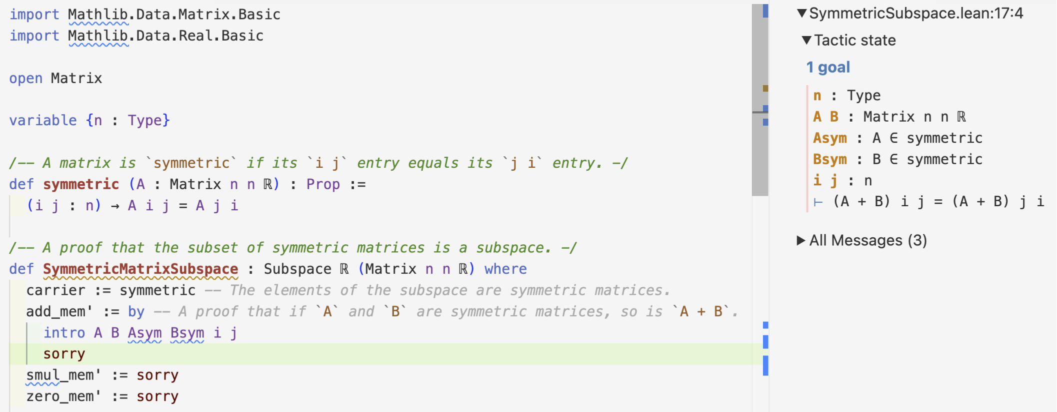
```
▼SymmetricSubspace.lean:16:4
 ▼Tactic state
  1 goal
   n : Type
   ⊢ ∀ {a b : Matrix n n ℝ}, a ∈ symmetric → b ∈ symmetric → a + b ∈ symmetric
▶All Messages (3)
```

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
```

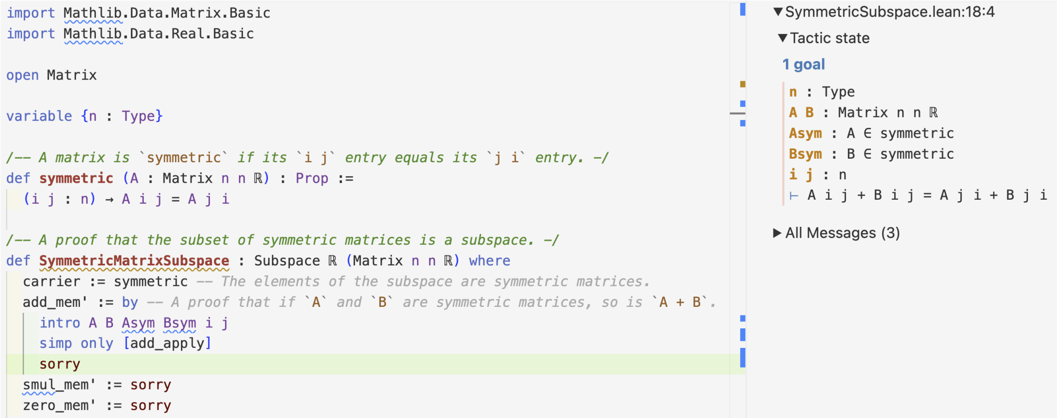
```
▼SymmetricSubspace.lean:17:4
 ▼Tactic state
  1 goal
  n : Type
  A B : Matrix n n ℝ
  Asym : A ∈ symmetric
  Bsym : B ∈ symmetric
  i j : n
  ⊢ (A + B) i j = (A + B) j i

▶ 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
```

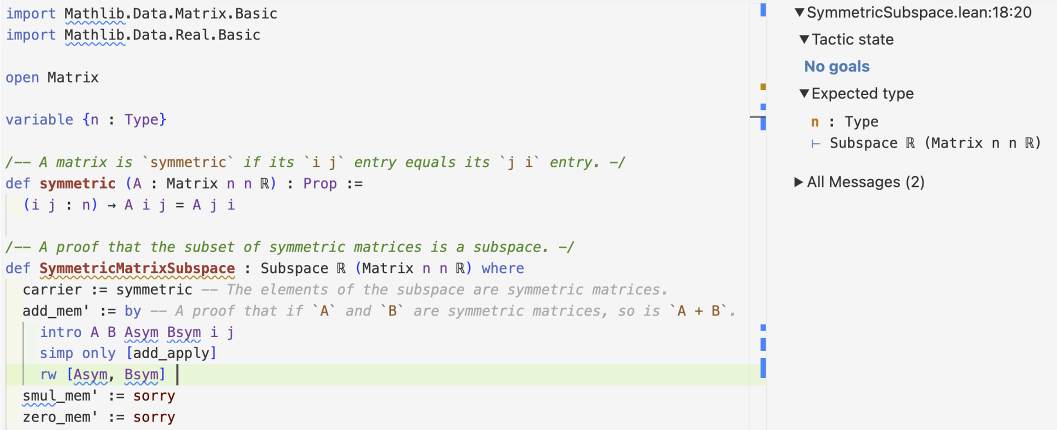
▶ All Messages (3)

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:



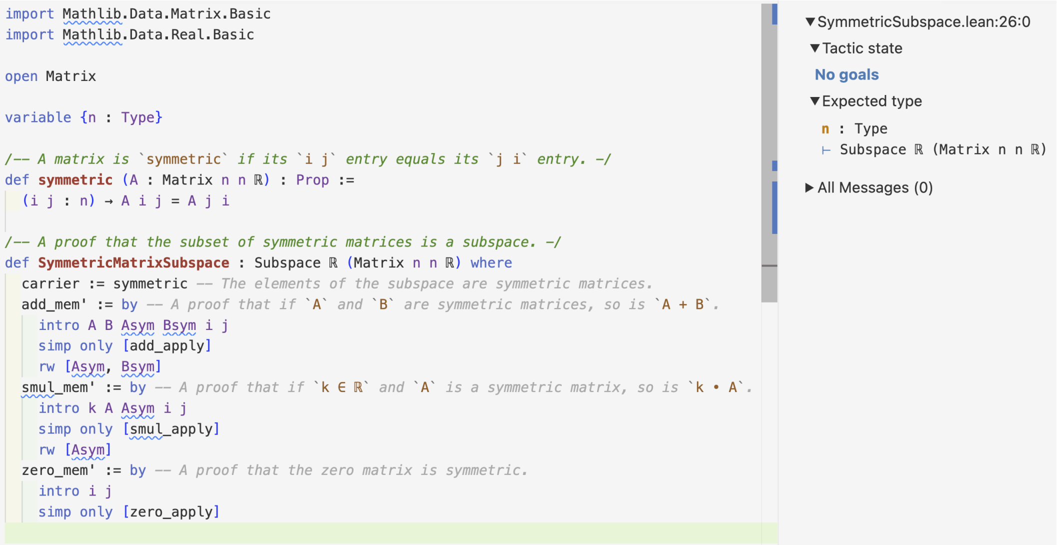
```
import Mathlib.Data.Matrix.Basic
import Mathlib.Data.Real.Basic

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

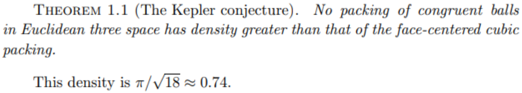


```
import Mathlib.Data.Matrix.Basic
import Mathlib.Data.Real.Basic

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)
```

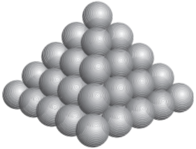
A proof too long to referee?

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.*

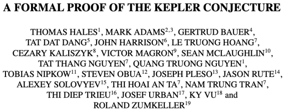
This density is $\pi/\sqrt{18} \approx 0.74$.



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.

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 `Isabelle` and `HOL Light`. 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.

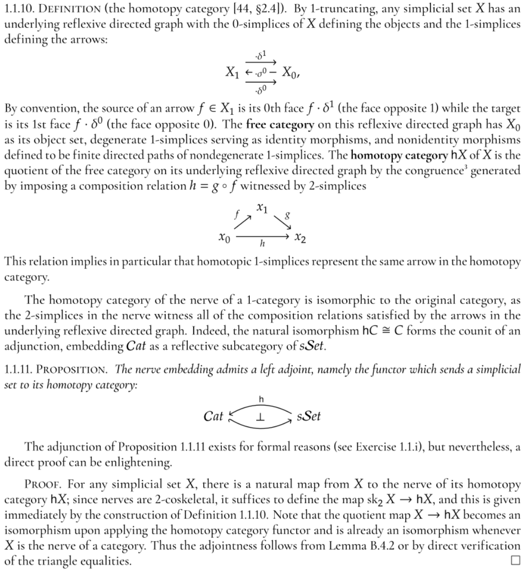
From pen-and-paper to formalized mathematics

There are considerable challenges that arise when converting a pen-and-paper proof to its computer formalized counterpart:

- The practice of explaining a mathematical proof to a computer requires absolute precision, in particular regarding the exact definitions of mathematical terms.
- Relatedly, great care must be taken in formalizing mathematical definitions to ensure the formalization is both accurate and efficient.
- While the proof assistant checks the logic in proofs of stated theorems, the user must verify that the definitions and the theorem statements are the intended ones!
- Proofs that are easy to follow — for domain experts, at least — are often riddled with invisible mathematics, which must be laboriously explained to the computer.

A pen-and-paper proof

A joint book with Dominic Verity reviews the construction of the reflective embedding of 1-categories into ∞-categories in less than one page:



1.1.10. Definition (the homotopy category [44, §2.4]). By 1-truncating, any simplicial set X has an underlying reflexive directed graph with the 0-simplices of X defining the objects and the 1-simplices defining the arrows:

$$\begin{tikzcd} X_1 \arrow[r, shift left=3, "\cdot\delta^1"] \arrow[r, shift right=3, "\cdot\delta^0"'] & X_0, \arrow[l, "\cdot\sigma^0" description] \end{tikzcd}$$

By convention, the source of an arrow $f \in X_1$ is its 0th face $f \cdot \delta^1$ (the face opposite 1) while the target is its 1st face $f \cdot \delta^0$ (the face opposite 0). The **free category** on this reflexive directed graph has X_0 as its object set, degenerate 1-simplices serving as identity morphisms, and nonidentity morphisms defined to be finite directed paths of nondegenerate 1-simplices. The **homotopy category** $\mathsf{h}X$ of X is the quotient of the free category on its underlying reflexive directed graph by the congruence[3] generated by imposing a composition relation $h = g \circ f$ witnessed by 2-simplices

$$\begin{tikzcd} & x_1 \arrow[dr, "g"] & \\ x_0 \arrow[ur, "f"] \arrow[rr, "h"'] & & x_2 \end{tikzcd}$$

This relation implies in particular that homotopic 1-simplices represent the same arrow in the homotopy category.

The homotopy category of the nerve of a 1-category is isomorphic to the original category, as the 2-simplices in the nerve witness all of the composition relations satisfied by the arrows in the underlying reflexive directed graph. Indeed, the natural isomorphism $\mathsf{h}C \cong C$ forms the counit of an adjunction, embedding $\mathcal{C}at$ as a reflective subcategory of $s\mathcal{S}et$.

1.1.11. Proposition. *The nerve embedding admits a left adjoint, namely the functor which sends a simplicial set to its homotopy category:*

$$\begin{tikzcd} \mathcal{C}at \arrow[r, bend right, ""{name=B}] & s\mathcal{S}et \arrow[l, bend right, "\mathsf{h}"', ""{name=A}] \arrow[from=A, to=B, phantom, "\perp"] \end{tikzcd}$$

The adjunction of Proposition 1.1.11 exists for formal reasons (see Exercise 1.1.i), but nevertheless, a direct proof can be enlightening.

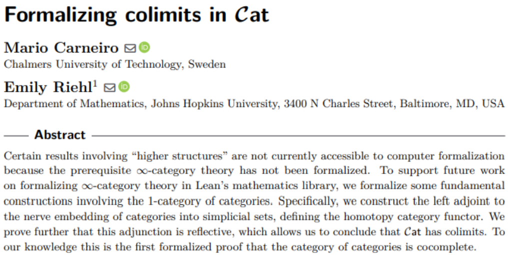
Proof. For any simplicial set X, there is a natural map from X to the nerve of its homotopy category $\mathsf{h}X$; since nerves are 2-coskeletal, it suffices to define the map $\mathrm{sk}_2\, X \to \mathsf{h}X$, and this is given immediately by the construction of Definition 1.1.10. Note that the quotient map $X \to \mathsf{h}X$ becomes an isomorphism upon applying the homotopy category functor and is already an isomorphism whenever X is the nerve of a category. Thus the adjointness follows from Lemma B.4.2 or by direct verification of the triangle equalities. □

A formalized proof

It took three months (part time) of joint work with Mario Carneiro to formalize this result in Lean.

It then took another six months for this code to pass the review process to be integrated into Lean's Mathlib.

We wrote an 18 page paper to explain the formalization and the challenges we encountered along the way:



Formalizing colimits in $\mathcal{C}at$

Mario Carneiro
Chalmers University of Technology, Sweden

Emily Riehl[1]
Department of Mathematics, Johns Hopkins University, 3400 N Charles Street, Baltimore, MD, USA

Abstract

Certain results involving "higher structures" are not currently accessible to computer formalization because the prerequisite ∞-category theory has not been formalized. To support future work on formalizing ∞-category theory in Lean's mathematics library, we formalize some fundamental constructions involving the 1-category of categories. Specifically, we construct the left adjoint to the nerve embedding of categories into simplicial sets, defining the homotopy category functor. We prove further that this adjunction is reflective, which allows us to conclude that $\mathcal{C}at$ has colimits. To our knowledge this is the first formalized proof that the category of categories is cocomplete.

In summary, there was an 18× scaling factor from the original pen-and-paper proof to a pen-and-paper account of the formalization.

Could some of this pain be alleviated by automation?

Recent advances in autoformalization

While archives of formal proofs are much smaller than usual training data sets, there are recent advancements in autoformalization using generative AI:

- Autoformalization with Large Language Models: in 2022, a team from Google, Stanford, and Cambridge demonstrate that LLMs do reasonably well in translating natural language mathematics to formal theorem statements in `Isabelle/HOL`.
- Baldur: Whole-Proof Generation and Repair with Large Language Models: in 2023, a team from UMass Amherst, UIUC, and Google build a prototype capable of whole proof generation and proof repair in `Isabelle/HOL`.
- Solving olympiad geometry without human demonstrations: in 2024, Deep Mind debuted AlphaGeometry, a neuro-symbolic system made up of a neural language model and a symbolic deduction engine. In a benchmarking test of 30 International Mathematical Olympiad geometry problems, AlphaGeometry solved 25 within the standard Olympiad time limit.

Despite these advances, it will likely take some time before AIs can discover their own theorems that are mathematically interesting or relevant to the real world.

Applications on the horizon?

If fully automated theorem proving is a long way off, human/computer collaborations in machine-assisted proof are on the horizon.

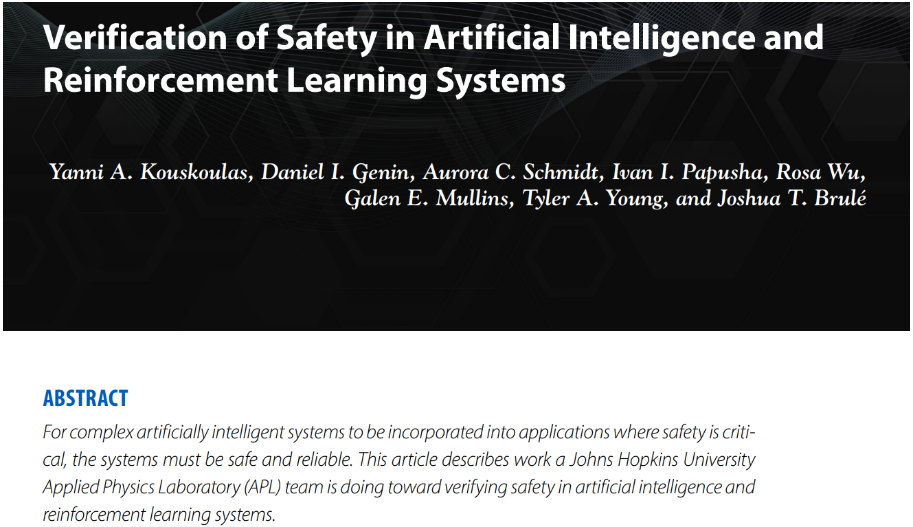
I will close with a few predictions about near future:

- Generative AI will dramatically improve the experience of interactive theorem proving in a computer proof assistant, revolutionizing mathematical practice.
- Humans will drive advances in formal methods, by developing domain-specific formal systems that can be used to specify desired behavior.
- Generative AI will then accelerate the process of writing formally verifiable software and proof certificates that it matches the specification.

What other applications are possible when machines can be trusted to “think” logically?

Applications to the real world

Computer proof assistants play a key role in formal methods, rigorous techniques developed at the APL and elsewhere to verify safety-critical systems:



Verification of Safety in Artificial Intelligence and Reinforcement Learning Systems

Yanni A. Kouskoulas, Daniel I. Genin, Aurora C. Schmidt, Ivan I. Papusha, Rosa Wu, Galen E. Mullins, Tyler A. Young, and Joshua T. Brulé

ABSTRACT

For complex artificially intelligent systems to be incorporated into applications where safety is critical, the systems must be safe and reliable. This article describes work a Johns Hopkins University Applied Physics Laboratory (APL) team is doing toward verifying safety in artificial intelligence and reinforcement learning systems.