To formalized mathematics and back with the Lean theorem prover

Kyle Miller
University of California, Santa Cruz

UCSC CSE Colloquium
24 January, 2024

I am interested in creating software tools for mathematicians, computer scientists, and students.

Idea:

We can leverage *formalization* in novel ways to design and develop useful tools.

- Formalization
- Informalization
- Knots and algorithms
- The future

Software correctness is essential

Computer systems are integrated into much of the modern world.

Software defects are frequent.

Defects can lead to loss of property or loss of life.

Aerospace

Ariane 5 test launch in 1996:

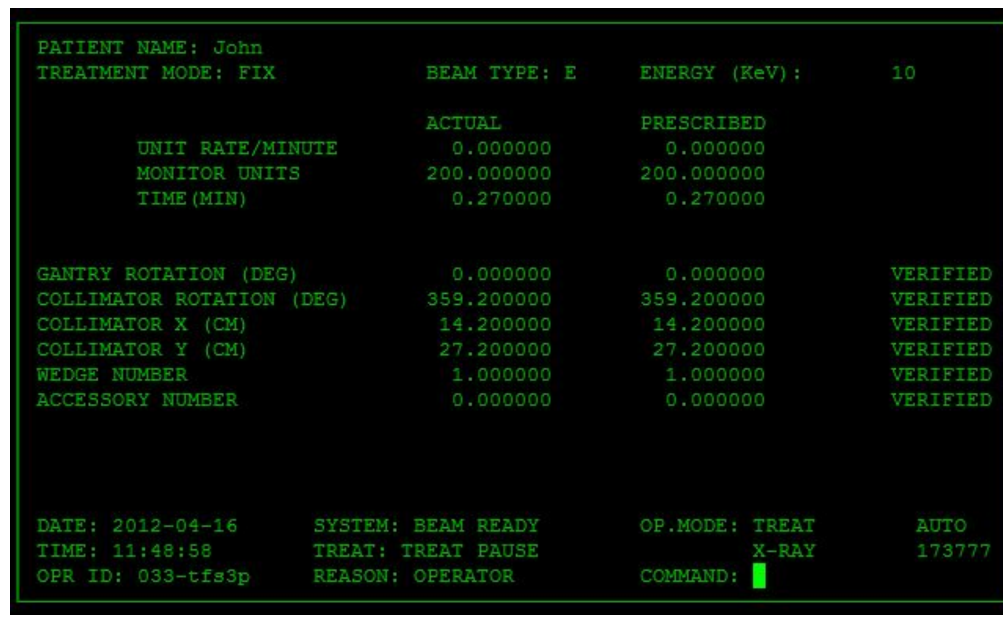
Invalid data conversion led to shutdown of inertial navigation system and destruction of launch vehicle.

$370 million

Medicine

1985-87: The *Therac-25* radiation therapy machine delivered at least six overdoses. Two attributed to race conditions.

Two deaths and at least four injured.



Cybersecurity

2012: OpenSSL had a latent buffer over-read vulnerability. Discovered and reported two years later as *Heartbleed*.

Unknown total impact, estimated at least $500 million.

Formal methods reduce software defects

Any testing or verification we can do reduces defects.

Gold standard: a full formal verification with a complete specification.

Writing proofs can be costly and takes expertise.

Reducing costs with Lean

The cost can be brought down with

- computer aid and
- libraries of formalized mathematics

Lean is a programming language and theorem prover, and it can be part of the solution.

Lean is a programming language

Lean 4 supports functional and imperative paradigms and compiles to C.

Has a sophisticated type system like Coq or Agda.

Supports invariants and proofs, using the same language.

```
def scale (xs : List Int) (c : Int) : List Int :=
  match xs with
  | [] => []
  | x :: xs' => (c * x) :: scale xs' c

def scale (a : Array Int) (c : Int) : Array Int := Id.run do
  let mut b : Array Int := #[]
  for h : i in [0:a.size] do
    b := b.push (c * a[i]'h.2)
  return b

structure NonemptyList (T : Type) where
  list : List T
  nonempty : list ≠ []
```

Mathematics itself is amenable to formal methods

A *formal logic system* is a language along with truth-preserving formal symbolic manipulations. *Examples: first-order logic, type theory.*

Formalization hypothesis:

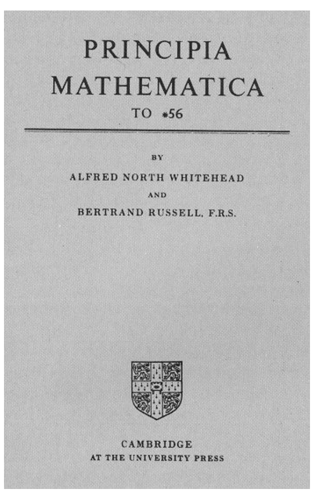
All of mathematics can be represented in a formal logic system.

Mathematics itself is amenable to formal methods

A *formal logic system* is a language along with truth-preserving formal symbolic manipulations. *Examples: first-order logic, type theory.*

Formalization hypothesis:

> All of mathematics can be represented in a formal logic system.



∗54·43. $\vdash :. \alpha, \beta \in 1 . \supset : \alpha \cap \beta = \Lambda . \equiv . \alpha \cup \beta \in 2$

Dem.

$\vdash . {*}54{\cdot}26 . \supset \vdash :. \alpha = \iota \text{‘} x . \beta = \iota \text{‘} y . \supset : \alpha \cup \beta \in 2 . \equiv . x \neq y .$

$[{*}51{\cdot}231] \qquad \equiv . \iota \text{‘} x \cap \iota \text{‘} y = \Lambda .$

$[{*}13{\cdot}12] \qquad \equiv . \alpha \cap \beta = \Lambda \qquad (1)$

$\vdash . (1) . {*}11{\cdot}11{\cdot}35 . \supset$

$\vdash :. (\exists x, y) . \alpha = \iota \text{‘} x . \beta = \iota \text{‘} y . \supset : \alpha \cup \beta \in 2 . \equiv . \alpha \cap \beta = \Lambda \qquad (2)$

$\vdash . (2) . {*}11{\cdot}54 . {*}52{\cdot}1 . \supset \vdash .$ Prop

From this proposition it will follow, when arithmetical addition has been defined, that $1 + 1 = 2$.

Mathematics itself is amenable to formal methods

A *formal logic system* is a language along with truth-preserving formal symbolic manipulations. *Examples: first-order logic, type theory.*

Formalization hypothesis:

All of mathematics can be represented in a formal logic system.

Computer formalization hypothesis:

We can develop tools to write formal proofs at a comfortably high level.

Lean is flexible!

Lean is an interactive theorem prover

Has a similar type theory to Coq or Agda.

Has **mathlib**, a large library of undergraduate-level mathematics and beyond.

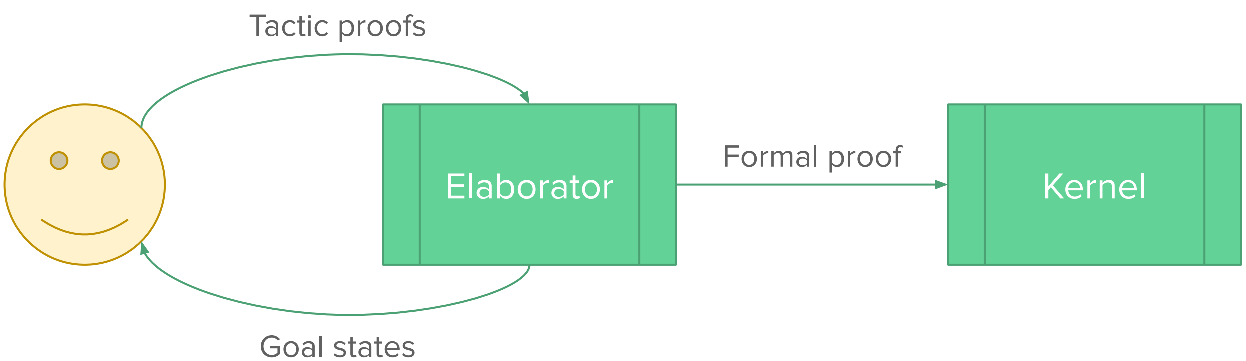
Proofs are “programs”, often constructed with a higher-level *tactics* language.

Lean is an interactive theorem prover

Has a similar type theory to Coq or Agda.

Has **mathlib**, a large library of undergraduate-level mathematics and beyond.

Proofs are “programs”, often constructed with a higher-level *tactics* language.

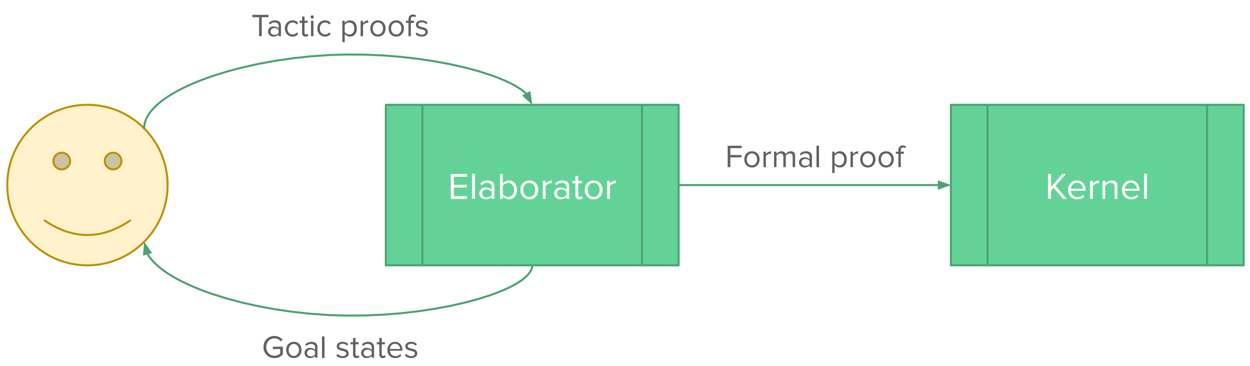


Lean is an interactive theorem prover

Has a similar type theory to Coq or Agda.

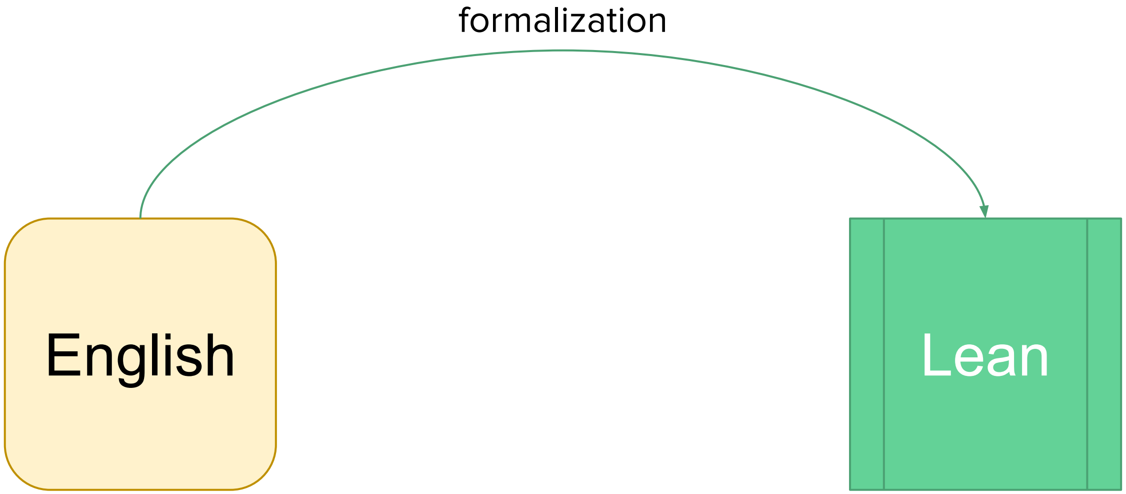
Has **mathlib**, a large library of undergraduate-level mathematics and beyond.

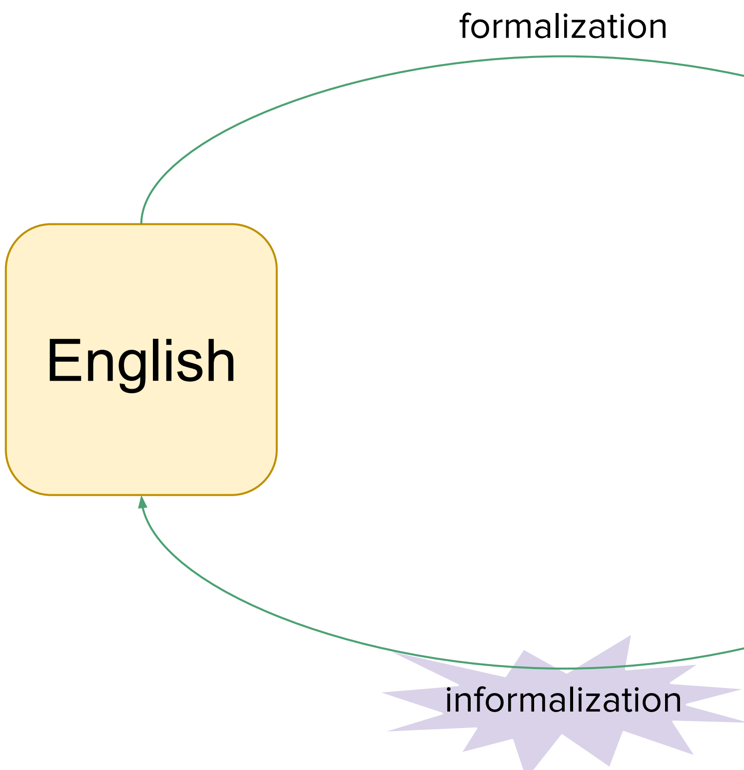
Proofs are “programs”, often constructed with a higher-level *tactics* language.




```
theorem scale_append (xs ys : List Int) (c : Int) :
    scale (xs.append ys) c = (scale xs c).append (scale ys c) := by
  induction xs with
  | nil => rfl
  | cons x xs ih => simp only [List.append, scale, ih]
```

- Formalization
- Informalization
- Knots and algorithms
- The future


formalization
English
Lean



We realized this arrow was missing!

Patrick Massot, Université Paris-Saclay

Textbooks lack context

From Rudin, *Principles of Mathematical Analysis*:

2.23 Theorem *A set E is open if and only if its complement is closed.*

Proof First, suppose E^c is closed. Choose $x \in E$. Then $x \notin E^c$, and x is not a limit point of E^c. Hence there exists a neighborhood N of x such that $E^c \cap N$ is empty, that is, $N \subset E$. Thus x is an interior point of E, and E is open.

Next, suppose E is open. Let x be a limit point of E^c. Then every neighborhood of x contains a point of E^c, so that x is not an interior point of E. Since E is open, this means that $x \in E^c$. It follows that E^c is closed.

Textbooks lack context

From Rudin, *Principles of Mathematical Analysis*:

2.23 Theorem *A set E is open if and only if its complement is closed.*

Proof First, suppose E^c is closed. Choose $x \in E$. Then $x \notin E^c$, and x is not a limit point of E^c. Hence there exists a neighborhood N of x such that $E^c \cap N$ is empty, that is, $N \subset E$. Thus x is an interior point of E, and E is open.

Next, suppose E is open. Let x be a limit point of E^c. Then every neighborhood of x contains a point of E^c, so that x is not an interior point of E. Since E is open, this means that $x \in E^c$. It follows that E^c is closed.

Why are we choosing an x?
What is the current goal?

Textbooks lack context

From Rudin, *Principles of Mathematical Analysis*:

2.23 Theorem *A set E is open if and only if its complement is closed.*

Proof First, suppose E^c is closed. Choose $x \in E$. Then $x \notin E^c$, and x is not a limit point of E^c. Hence there exists a neighborhood N of x such that $E^c \cap N$ is empty, that is, $N \subset E$. Thus x is an interior point of E, and E is open.

Next, suppose E is open. Let x be a limit point of E^c. Then every neighborhood of x contains a point of E^c, so that x is not an interior point of E. Since E is open, this means that $x \in E^c$. It follows that E^c is closed.

Current proof state:

X is a topological space

E is a subset of X

E^c is closed

x is an element of E

Goal: $x \in E^\circ$

What if the document could show this context?

Textbooks show only one level of detail

From Rudin, *Principles of Mathematical Analysis*:

2.23 Theorem *A set E is open if and only if its complement is closed.*

Proof First, suppose E^c is closed. Choose $x \in E$. Then $x \notin E^c$, and x is not a limit point of E^c. Hence there exists a neighborhood N of x such that $E^c \cap N$ is empty, that is, $N \subset E$. Thus x is an interior point of E, and E is open.

Next, suppose E is open. Let x be a limit point of E^c. Then every neighborhood of x contains a point of E^c, so that x is not an interior point of E. Since E is open, this means that $x \in E^c$. It follows that E^c is closed.

Textbooks show only one level of detail

From Rudin, *Principles of Mathematical Analysis*:

2.23 Theorem *A set E is open if and only if its complement is closed.*

Proof First, suppose E^c is closed. Choose $x \in E$. Then $x \notin E^c$, and x is not a limit point of E^c. Hence there exists a neighborhood N of x such that $E^c \cap N$ is empty, that is, $N \subset E$. Thus x is an interior point of E, and E is open.

Next, suppose E is open. Let x be a limit point of E^c. Then every neighborhood of x contains a point of E^c, so that x is not an interior point of E. Since E is open, this means that $x \in E^c$. It follows that E^c is closed.

Why can we deduce this fact?

Textbooks show only one level of detail

From Rudin, *Principles of Mathematical Analysis*:

2.23 Theorem *A set E is open if and only if its complement is closed.*

Proof First, suppose E^c is closed. Choose $x \in E$. Then $x \notin E^c$, and x is not a limit point of E^c. Hence there exists a neighborhood N of x such that $E^c \cap N$ is empty, that is, $N \subset E$. Thus x is an interior point of E, and E is open.

Next, suppose E is open. Let x be a limit point of E^c. Then every neighborhood of x contains a point of E^c, so that x is not an interior point of E. Since E is open, this means that $x \in E^c$. It follows that E^c is closed.

prove $E \subseteq E^\circ$. Let x be an element of E. ⊕One can see that $x \notin E^c$.

⊖Claim: x is not a limit point of E^c.

> Assume that x is a limit point of E^c. ⊕We see $x \in E^c$. Using this and our assumption that $x \notin E^c$ we are done.

⊕One can obtain an open neighborhood N of x such that $N \cap E^c = \varnothing$.

Imagine clicking a ⊕ and seeing further proof.

To go arbitrarily deep,
we need a formalized proof!

A *structured proof* is a proof written to make the logical structure evident.

Def The greatest common divisor d of a and b is the greatest element of $\{d \in \mathbb{Z} \mid d \mid a \text{ and } d \mid b\}$, if it exists. Written as (a,b) or $\gcd(a,b)$.

Thm Suppose $a, b \in \mathbb{Z}$ and $a \neq 0$ or $b \neq 0$. Then (a,b) exists and is positive.

Pf Without loss of generality, $a \neq 0$ (by $a \leftrightarrow b$ if needed).
Let $S = \{d \in \mathbb{Z} \mid d \mid a \text{ and } d \mid b\}$.
Claim: $1 \in S$.
$1 \in S \Leftrightarrow 1 \mid a$ and $1 \mid b$. $1a = a$ and $1b = b$, so holds by def.
Hence S is nonempty.
Claim: for all $e \in S$, $|e| \leq |a|$. ✓
Suppose $e \in S$. Then $e \mid a$, so there is $k \in \mathbb{Z}$ such that $ke = a$.
Let $k' = |k| \in \mathbb{N}$. Then $k'|e| = |a|$.
Case I) $k' = 0$. Then $k = 0$, and $a = ke = 0$. Contradiction!!
Case II) There is $k'' \in \mathbb{N}$ such that $k' = k'' + 1$. Then $|a| = k'|e| = k''|e| + |e|$, thus $|e| \leq |a|$ by def.
Have $S \cap \{1, \ldots, |a|\}$ nonempty. Adapting lemma for well-ordering, this has a greatest element d. By second claim, $d \in S$ is greatest. ∎

from my Math 110 lectures

Lemma 2.2.8

The element $a \in F$ is a root of the nonzero polynomial $f(x)$ in F iff $x - a$ is a factor of $f(x)$. As a consequence, the number of roots of a polynomial is no more than its degree.

Proof 2.2.9

1. If $a \in F$ is a root of f, then $x - a$ is a factor of f.
    1. **Suppose** $a \in F$ is a root of f.
    2. **Let** q, r satisfy $f(x) = q(x)(x-a) + r(x)$ per the division algorithm.
    3. $0 = f(a) = q(a)(a-a) + r(a) = r(a)$
    4. Since $\deg r \leq 1, r(x) = 0$
    5. Hence $f(x) = q(x)(x-a)$.
2. If $a \in F$ and $x - a$ is a factor of F, then a is a root of F.
    1. **Let** q be such that $f(x) = q(x)(x-a)$
    2. $f(a) = q(a)(a-a) = 0$
3. The number of roots of f is no more than the degree of f.
    1. **Case I.** $\deg f = 0$
        1. Since f is non-zero, it has no roots.

       Case II. $\deg f > 0$
        1. **Case I.** f has a root $a \in F$
            1. Then f has $(x-a)$ as a factor.
            2. **Let** q be such that $f(x) = q(x)(x-a)$.
            3. $\deg q < \deg f$. By induction, q has at most $\deg q$ roots.
            4. Thus f has at most $1 + \deg q = \deg f$ roots.

           Case II. f does not have a root in F
            1. f has $0 < \deg f$ roots.

from my personal notes

A formalized proof is the exemplary structured proof.

```
theorem rudin {X : Type*} [TopologicalSpace X] (E : Set X) :
    IsOpen E ↔ IsClosed Eᶜ := by
  constructor
  · intro hop
    apply isClosed_iff_clusterPt'.2
    intro x hx
    have hx' : x ∉ interior E := by
      intro hi
      rcases mem_interior.1 hi with ⟨U, hU, hop, hm⟩
      rcases hx U (IsOpen.mem_nhds hop hm) hop with ⟨y, hy⟩
      exact absurd (hU (Set.mem_of_mem_inter_left hy))
              ((Set.mem_compl_iff _ _).1 <| Set.mem_of_mem_inter_right hy)
    have hintc := Set.compl_subset_compl_of_subset (subset_interior_iff_isOpen.2 hop)
    exact hintc hx'
  · intro hc
    apply subset_interior_iff_isOpen.1
    intro x hx
    have hx' : x ∉ Eᶜ := Set.not_mem_compl_iff.mpr hx
    have hnc : ¬ClusterPt' x Eᶜ := by
      intro h
      exact absurd (isClosed_iff_clusterPt'.1 hc _ h) hx'
    rcases not_clusterPt'_principal_iff.1 hnc with ⟨N, hN, hop, he⟩
    apply mem_interior.2
    use N, Set.diff_eq_empty.mp he, hop
    exact mem_of_mem_nhds hN
```

However, it is not human readable on its own.

Informalization for interactive structured proofs

Theorem. Let X be a topological space. Let E be a subset of X. Then E is open if and only if E^c is closed.

Proof. By definition it suffices (1) to prove that if E is open then E^c is closed and (2) to prove that if E^c is closed then E is open.

1. Claim: if E is open then E^c is closed.

   Suppose E is open. Using isClosed_iff_clusterPt' it suffices to prove for all cluster points a of E^c, $a \in E^c$. Let x be a cluster point of E^c.

   Claim: $x \notin E^\circ$.

   Assume that $x \in E^\circ$. Using our assumption that $x \in E^\circ$, mem_interior proves one can obtain an open subset U of E such that $x \in U$. We see for all open neighborhoods U' of x, $U' \cap E^c$ is nonempty.

   Claim: U is a neighborhood of x.

   Using our assumption that U is open and our assumption that $x \in U$ with IsOpen.mem_nhds we are done.

   Using the above claim and our assumption that U is open, a preceding claim proves one can obtain an element y of $U \cap E^c$.

   Claim: $y \in E$.

   We see $y \in U$. Using the above claim with our assumption we are done.

   Claim: $y \notin E$.

   We see $y \in E^c$. Using the above claim with Set.mem_compl_iff we are done.

   Using the above claims we are done.

   Claim: $(E^\circ)^c \subseteq E^c$.

   Claim: $E \subseteq E^\circ$.

   Using our assumption that E is open with subset_interior_iff_isOpen we are done.

   Using the above claim with Set.compl_subset_compl_of_subset we are done.

   Claim: $x \in (E^\circ)^c$.

   It suffices to prove that $x \notin E^\circ$, and by assumption we are done.

   Using the above claim with our assumption we are done.

2. Claim: if E^c is closed then E is open.

   Assume that E^c is closed. Using subset_interior_iff_isOpen it suffices to prove $E \subseteq E^\circ$. Let x be an element of E.

   Claim: $x \notin E^c$.

   Using our assumption that $x \in E$ with Set.not_mem_compl_iff we are done.

   Claim: x is not a cluster point of E^c.

   Assume that x is a cluster point of E^c.

   Claim: $x \in E^c$.

   Claim: for all cluster points a of E^c, $a \in E^c$.

   Using our assumption that E^c is closed with isClosed_iff_clusterPt' we are done.

   Using our assumption that x is a cluster point of E^c with a preceding claim we are done.

   Using the above claim and our assumption that $x \notin E^c$ we are done.

   Using our assumption that x is not a cluster point of E^c, not_clusterPt'_principal_iff proves one can obtain an open neighborhood N of x such that $N \cap E^c = \varnothing$. Using mem_interior it suffices to prove there exists an open subset t of E such that $x \in t$.

   Claim: $N \subseteq E$.

   We see $N \setminus E = \varnothing$. Using the above claim with Set.diff_eq_empty we obtain $N \subseteq E$.

   Using the above claim, N and our assumption that N is open it suffices to prove $x \in N$. Using our assumption that N is a neighborhood of x with mem_of_mem_nhds we are done.

□

Such a document could answer any question you have about the proof.

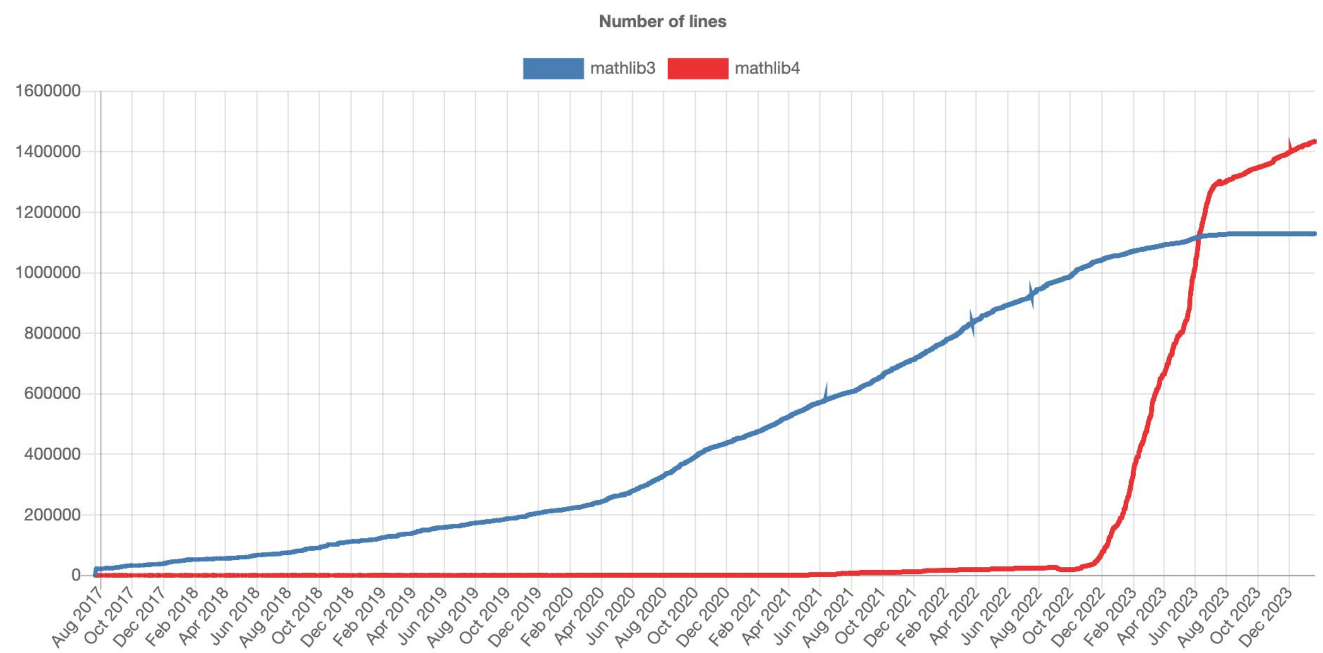
But authoring these by hand is a chore.

Mathlib is a large library of formalized mathematics

73k definitions
135k theorems

Contains undergraduate, graduate, and even some research level mathematics.

Can all of mathematics be formalized in a unified, cohesive library?



Hall’s marriage theorem in mathlib [Gusakov–Mehta–Miller 2021]

```
theorem Finset.all_card_le_biUnion_card_iff_exists_injective
      {ι : Type u} {α : Type v} [DecidableEq α] (t : ι → Finset α) :
  (∀ (s : Finset ι), s.card ≤ (Finset.biUnion s t).card) ↔
   ∃ (f : ι → α), Function.Injective f ∧ ∀ (x : ι), f x ∈ t x
```

Informalization for accessible formal proof libraries

- ```
  theorem Finset.all_card_le_biUnion_card_iff_exists_injective
        {ι : Type u} {α : Type v} [DecidableEq α] (t : ι → Finset α) :
      (∀ (s : Finset ι), s.card ≤ (Finset.biUnion s t).card) ↔
      ∃ (f : ι → α), Function.Injective f ∧ ∀ (x : ι), f x ∈ t x
  ```

- Let ι be a type and let α be a type with decidable equality. Let t be an ι-indexed family of finite subsets of α. Then the following are equivalent:
  - For all finite subsets s of ι, $|s| \leq \left|\bigcup_{x \in s} t_x\right|$.
  - There exists an injective function $f : \iota \to \alpha$ such that for all x in ι, $f(x) \in t_x$.

This takes less specialized training to read!
  ```

Autoformalization

An *autoformalization* system automatically translates informal documents into a machine-checkable formalization.

Large Language Models (LLMs) are being applied to this problem.

Example: Math competition problem statements to formal specifications
[Wu–Jiang–Li–Rabe–Staats–Jamnik–Szegedy 2022]

Training data is scarce

Observation: formalization does not lead to training pairs.

Papers are too informal.

They have to be transformed and expanded.

The formalization might have no obvious connection to structure of paper.

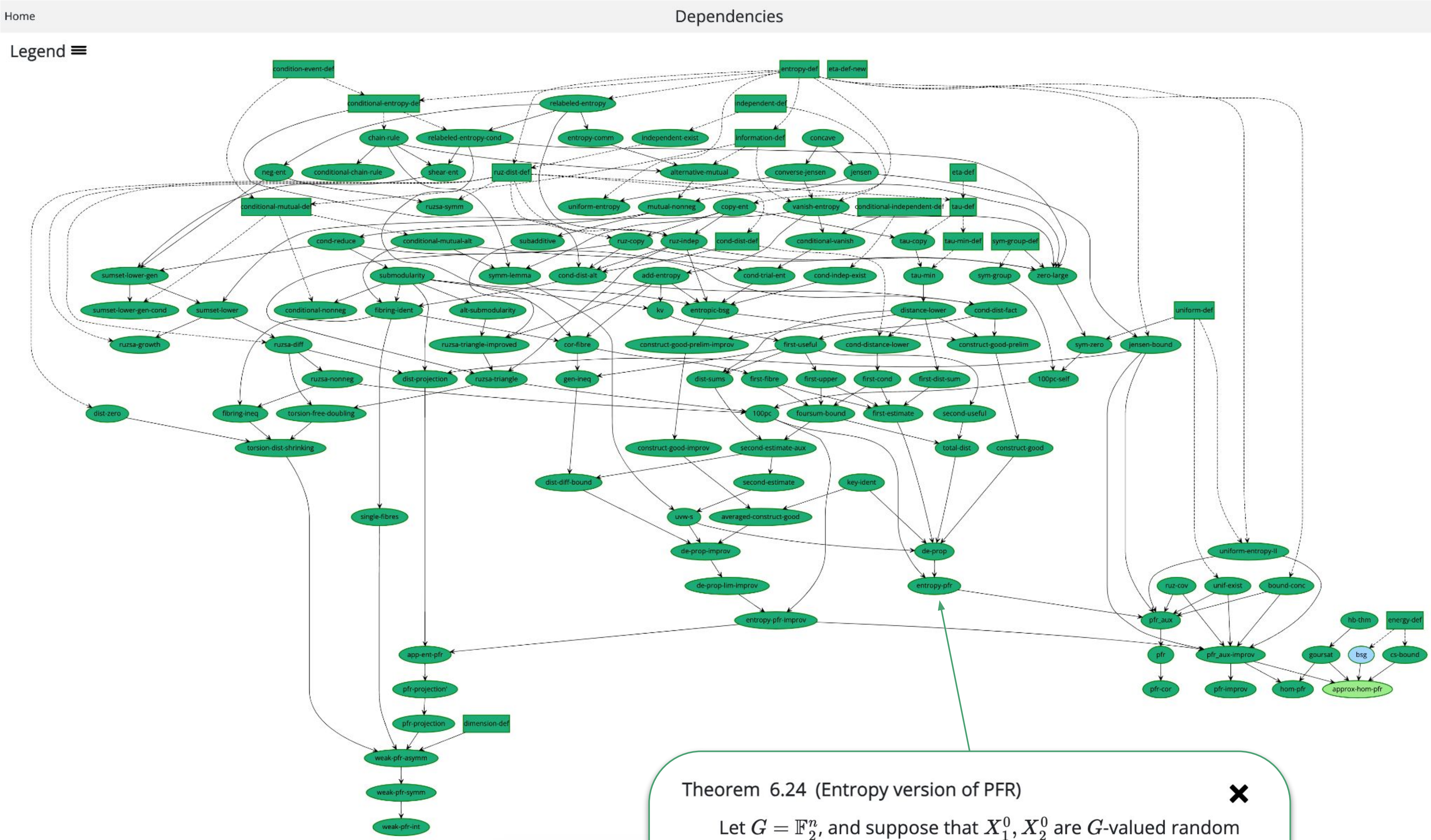
Formal blueprints

A *formal blueprint* is an design document for a formalization project.

It contains a plan (in natural language) of each theorem and definition that will be in the formalization.

Creating a formal blueprint takes significant effort and domain knowledge.

Formalizing from the blueprint takes only general knowledge.



Theorem 6.24 (Entropy version of PFR) ✖

Let $G = \mathbb{F}_2^n$, and suppose that X_1^0, X_2^0 are G-valued random variables. Then there is some subgroup $H \leq G$ such that

$$d[X_1^0; U_H] + d[X_2^0; U_H] \leq 11 d[X_1^0; X_2^0],$$

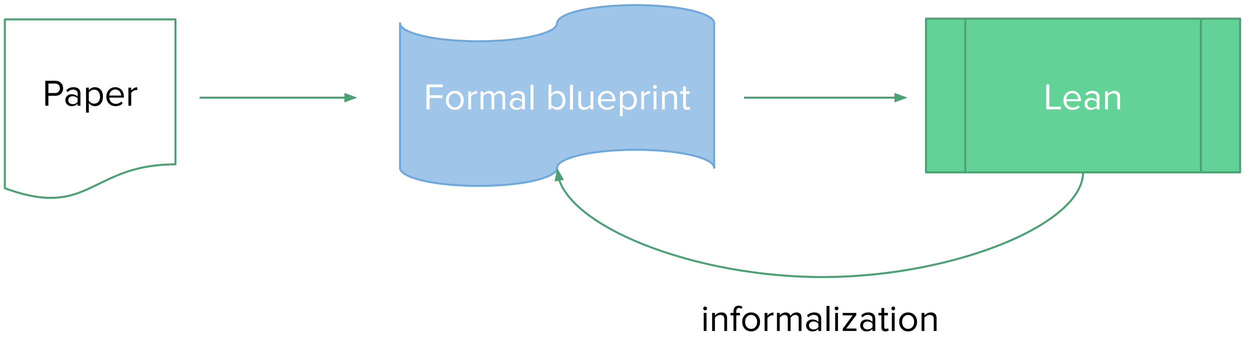
where U_H is uniformly distributed on H. Furthermore, both $d[X_1^0; U_H]$ and $d[X_2^0; U_H]$ are at most $6d[X_1^0; X_2^0]$.

LaTeX *Lean*

From the **formal blueprint** for Polynomial Freiman-Ruzsa (PFR) Conjecture project, led by Terence Tao

Informalization for training data for autoformalization

Informalization naturally yields a formal blueprint.



We could use it to generate two sets of pairs from arbitrary Lean code.

- Formal blueprint & Lean
- Paper & formal blueprint

Autoformalizers could train for each translation task independently.

How to informalize

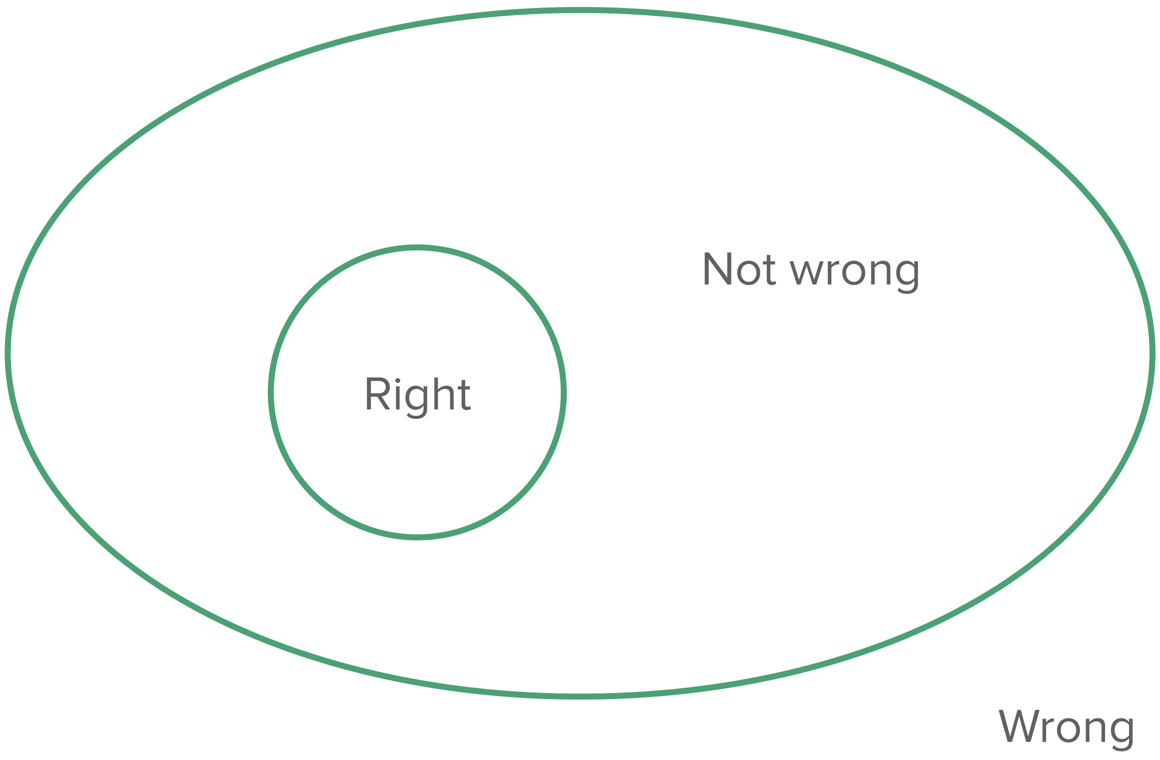
I developed a prototype auto-informalizer.

Input: a Lean 4 module

Output: an interactive document (HTML)

Users write extensions to translate concepts to English.
Currently extensions focus on topology.

Principle: output that is **not wrong**



Principle: output that is **not wrong**

Ideally, we output correct informal proofs.

Ambiguity, omissions, lack of clarity: these are bugs, but not wrong.

Informalizer uses symbolic AI principles, known as “good old-fashioned AI”.

We can have precise understanding of the informalization, and fix it as needed.

LLMs: output is not generally not wrong

We could apply a large language model (LLM) to the informalization problem.

However:

- While LLMs easily produce informal-looking text, it can be wrong.
- More training and better prompting, decrease probabilities, not eliminate.
- LLMs struggle with logical reasoning.

You

The following theorem statement is formalized in Lean. Please give a precise and accurate translation of it into mathematical English.

```
theorem Finset.all_card_le_biUnion_card_iff_exists_injective
  {ι : Type u} {α : Type v} [DecidableEq α] (t : ι → Finset α) :
  (∀ s : Finset ι, s.card ≤ (s.biUnion t).card) ↔
   ∃ f : ι → α, Function.Injective f ∧ ∀ x, f x ∈ t x
```

ChatGPT

The given Lean theorem formalizes a statement about finite sets. Let's translate it into mathematical English.

"For any types ι and α, where α is equipped with a decidable equality, and for any family of finite sets t indexed by elements of ι, the following statement holds: The cardinality of any finite subset s of ι is less than or equal to the cardinality of the union of the sets indexed by the elements of s if and only if there exists an injective function f from ι to α such that, for every element x in ι, the image of x under f belongs to the set indexed by x."

ChatGPT

The given Lean theorem formalizes a statement about finite sets. Let's translate it into mathematical English.

"For any types ι and α, where α is equipped with a decidable equality, and for any family of finite sets t indexed by elements of ι, the following statement holds: The cardinality of any finite subset s of ι is less than or equal to the cardinality of the union of the sets indexed by the elements of s if and only if there exists an injective function f from ι to α such that, for every element x in ι, the image of x under f belongs to the set indexed by x."

ChatGPT

The theorem states that for any indexed family of finite sets $t : \{\text{indexed by } \iota\}$ with elements in a type α, the following equivalence holds:

For all finite index sets $s : \text{Finset } \iota$, the cardinality of s is less than or equal to the cardinality of the union of the sets indexed by s (denoted as s biUnion t), if and only if there exists an injective function $f : \iota \to \alpha$ such that for all x in ι, the element $f(x)$ belongs to the set $t(x)$.

ChatGPT

The theorem states that for any indexed family of finite sets $t : \{\text{indexed by } \iota\}$ with elements in a type α, the following equivalence holds:

For all finite index sets $s : \text{Finset } \iota$, the cardinality of s is less than or equal to the cardinality of the union of the sets indexed by s (denoted as s biUnion t), if and only if there exists an injective function $f : \iota \to \alpha$ such that for all x in ι, the element $f(x)$ belongs to the set $t(x)$.

ChatGPT

The given Lean theorem can be translated into mathematical English as follows:

"For any types ι and α, where α is equipped with a decidable equality, and for any collection of finite sets indexed by ι denoted by t, the following statement holds: The cardinality of any finite subset s of ι is less than or equal to the cardinality of the union of the sets in t indexed by the elements of s if and only if there exists an injective function f from ι to α such that for every element x in ι, the image of x under f belongs to the set t(x)."

ChatGPT

The given Lean theorem can be translated into mathematical English as follows:

"For any types ι and α, where α is equipped with a decidable equality, and for any collection of sets indexed by ι denoted by t : ι → Finset α, the following statement holds: The cardinality of any finite subset s of ι is less than or equal to the cardinality of the union of the sets corresponding to the elements of s, if and only if there exists an injective function f : ι → α such that for every element x in ι, the image of x under f belongs to the set t x."

ChatGPT

The theorem states that for any indexed collection of finite sets $t : \backslash\iota \to \text{Finset } \backslash\alpha$, the following equivalence holds:

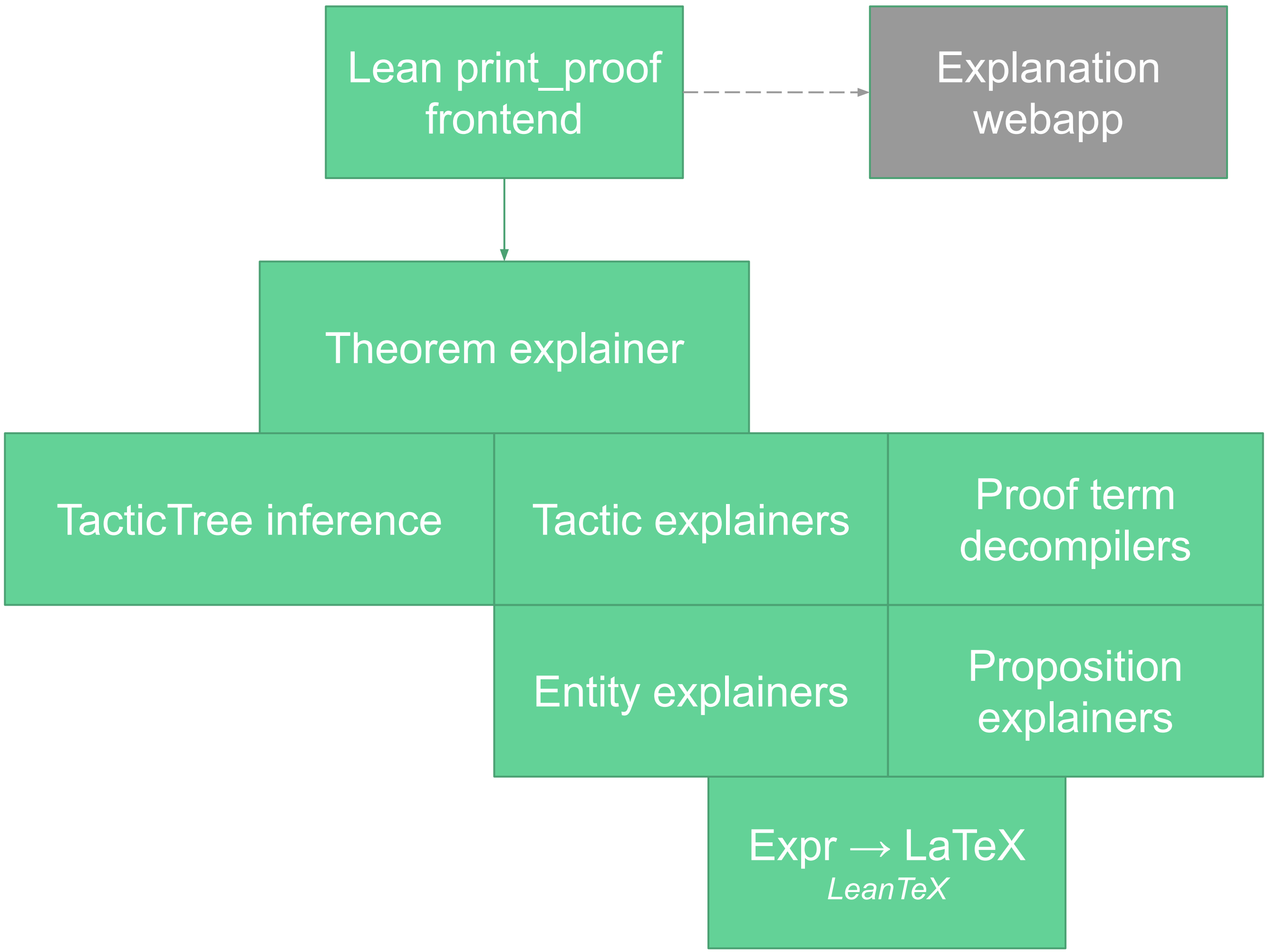
$$\forall s : \text{Finset } \backslash\iota,\ s.card \leq (s \,\text{biUnion}\, t).card \iff \exists f : \backslash\iota \to \backslash\alpha,\ \text{Functi}$$

In simpler terms, the inequality $s.card \leq (s \,\text{biUnion}\, t).card$ for any finite set s of indices is equivalent to the existence of an injective function f from the indices to the elements, such that each element fx belongs to the corresponding set tx.

Consecutive responses with the same prompt, all different

Let’s look at a demo

General architecture



Theorem *A set E is open if and only if its complement is closed.*

```lean
theorem rudin [TopologicalSpace X] (E : Set X) :
    IsOpen E ↔ IsClosed Eᶜ
```

Theorem (`rudin`). Let X be a topological space. Let E be a subset of X. Then E is open if and only if E^c is closed.

From Bourbaki, *Topologie Générale:*

Theorem. *Let X be a topological space, A a dense subset of X, $f : X \to Y$ a mapping of X into a regular space Y. If, for each $x \in X$, $f(y)$ tends to $f(x)$ when y tends to x while remaining in A then f is continuous.*

```
theorem continuous_of_dense [TopologicalSpace X] [TopologicalSpace Y] [RegularSpace' Y]
    {A : Set X} (hA : Dense A) (f : X → Y) (hf : ∀ x, ContinuousWithinAt' f A x) : Continuous' f
```

Theorem (`continuous_of_dense`). Let X be a topological space and let Y be a regular topological space. Let A be a dense subset of X. Let $f : X \to Y$ be a function. Assume that for all elements x of X, f is continuous at x within A. Then f is continuous.

Ontologies

In AI, an *ontology* is a formal system that models knowledge about a domain:

concepts, properties, and relations

Ontologies are the foundation for reasoning and inference.

Engineering an ontology takes detecting un-represented knowledge and modifying the model to represent it.

Lean and English

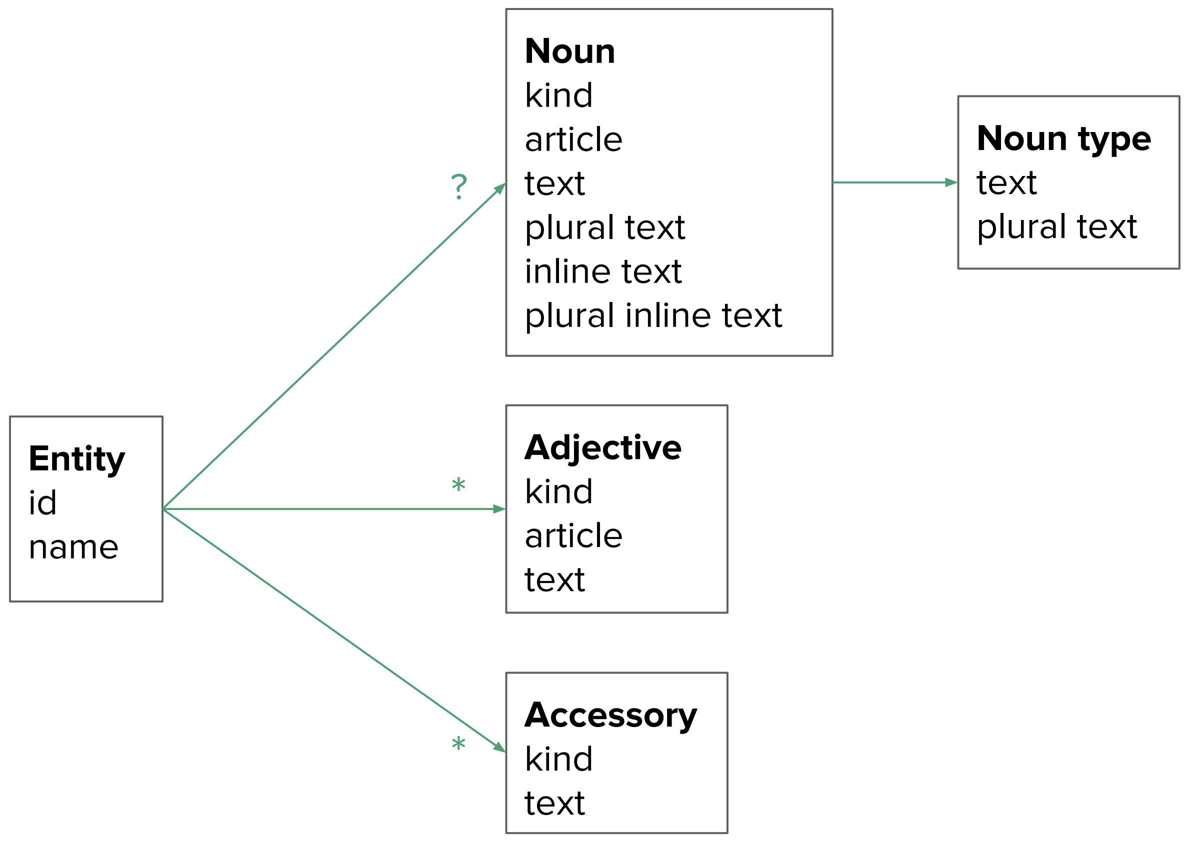
Lean 4 has expressions, declarations, metavariables, local contexts, tactic states, tactics, and so on.

To translate to English, we need

- an ontology compatible with (a subset of) common practice mathematical language and
- a mapping from the Lean 4 ontology to the English ontology.

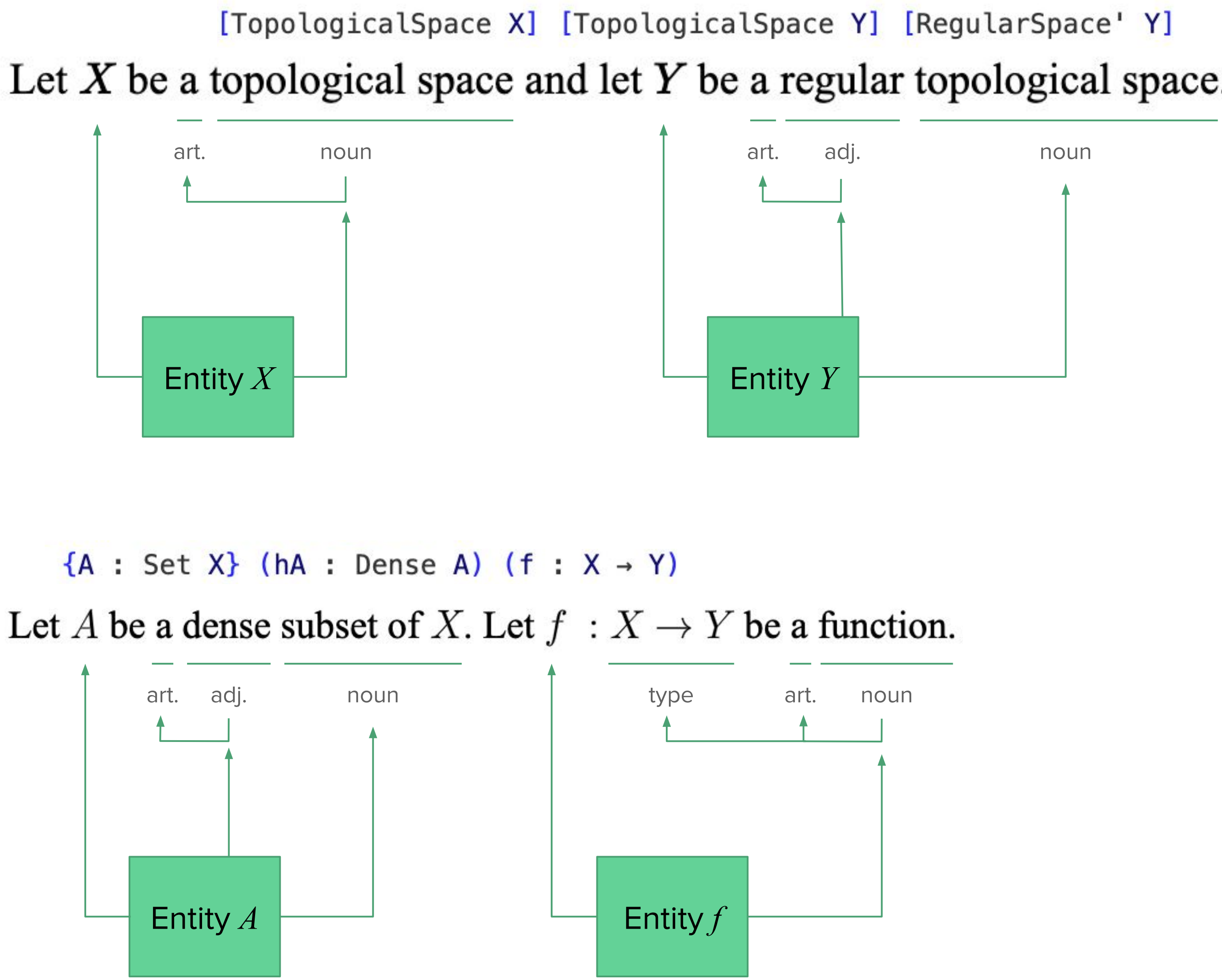
The better the ontology, the more natural the output we can produce.

An ontology for theorem-style paragraphs



```
structure Entity where
  fvarid : FVarId
  entityName : String
  noun : Option Noun
  provides : Array FVarId := #[fvarid]
  adjectives : Array Adjective := #[]
  accessories : Array Accessory := #[]
```

```
structure Adjective where
  kind : Name
  expr : Expr
  article : Article
  text : String

structure Accessory where
  kind : Name
  expr : Expr
  text : String

structure NounTypePayload where
  type : String
  (text pluralText : String)

structure Noun where
  kind : Name
  article : Article
  text : String
  pluralText : String
  typePayload : Option NounTypePayload := none
  (inlineText inlinePluralText : String)
```



[TopologicalSpace X] [TopologicalSpace Y] [RegularSpace' Y]
Let X be a topological space and let Y be a regular topological space.
art.
noun
art.
adj.
noun
Entity X
Entity Y
{A : Set X} (hA : Dense A) (f : X → Y)
Let A be a dense subset of X. Let f : X → Y be a function.
art.
adj.
noun
type
art.
noun
Entity A
Entity f

Entity construction

For each local variable,

Find and run a handler that applies to the variable.

Handlers generally

Decide what the local variable is about.

Examples: `(T : Type)` is about T `(h : U ∈ Nhd x)` is about U

Looks for or creates an entity entry, taking into account dependencies.

Adds/alters the Noun or attaches an Adjective or Accessory, as appropriate.

```lean
@[english_param const.TopologicalSpace] def param_TopologicalSpace : EnglishParam
| fvarid, deps, type@(.app _ (.fvar fvaridE)), _ => do
  trace[English] "Using the english_param handler for TopologicalSpace"
  let e ← getEntityFor fvaridE deps
  if e.kind == `Type then
    let ns : NounSpec :=
      { kind := `TopologicalSpace
        article := .a
        text := nt!"topological space{.s}"
        inlineText := nt!"topological space{.s} {.latex e.entityName}" }
    addEntity <| e.pushNoun fvarid (← ns.toNoun #[type])
  else
    addEntity <| e.pushAccessory fvarid
      { kind := `TopologicalSpace,
        expr := type,
        text := "a topology" }
| _, _, _, _ => failure

@[english_param const.RegularSpace'] def param_RegularSpace' : EnglishParam
| fvarid, deps, type@(.app (.app _ (.fvar fvaridE)) _), false => do
  trace[English] "Using the english_param handler for RegularSpace'"
  let e ← getEntityFor fvaridE deps
  addEntity <| e.pushAdjective fvarid
    { kind := `RegularSpace',
      expr := type,
      article := .a,
      text := "regular" }
| _, _, _, _ => failure

@[english_param const.Dense] def param_Dense : EnglishParam
| fvarid, deps, type@(.app _ (.fvar fvaridE)), false => do
  trace[English] "Using the english_param handler for Dense"
  let e ← getEntityFor fvaridE deps
  addEntity <| e.pushAdjective fvarid
    { kind := `Dense,
      expr := type,
      article := .a,
      text := "dense" }
| _, _, _, _ => failure

@[english_param const.IsOpen] def param_IsOpen : EnglishParam
| fvarid, deps, type@(.app _ (.fvar fvaridE)), false => do
  trace[English] "Using the english_param handler for IsOpen"
  let e ← getEntityFor fvaridE deps
  addEntity <| e.pushAdjective fvarid
    { kind := `IsOpen,
      expr := type,
      article := .an,
      text := "open" }
| _, _, _, _ => failure

@[english_param const.IsClosed] def param_IsClosed : EnglishParam
| fvarid, deps, type@(.app _ (.fvar fvaridE)), false => do
  let e ← getEntityFor fvaridE deps
  addEntity <| e.pushAdjective fvarid
    { kind := `IsClosed,
      expr := type,
      article := .a,
      text := "closed" }
| _, _, _, _ => failure
```

The parameter handlers are currently crafted by hand

Basic grammatical construction

“**Let** *entityName* [**:** *noun.type*] **be** *article adjectives noun.text* **with** *accessories***.**”

Let n be a natural number.

Let $f : X \to Y$ be an injective function.

“**For all** *adjectives noun.inlineText **with** accessories***,** …”

For all finite types T with decidable equality, …

A simple calculation: merging

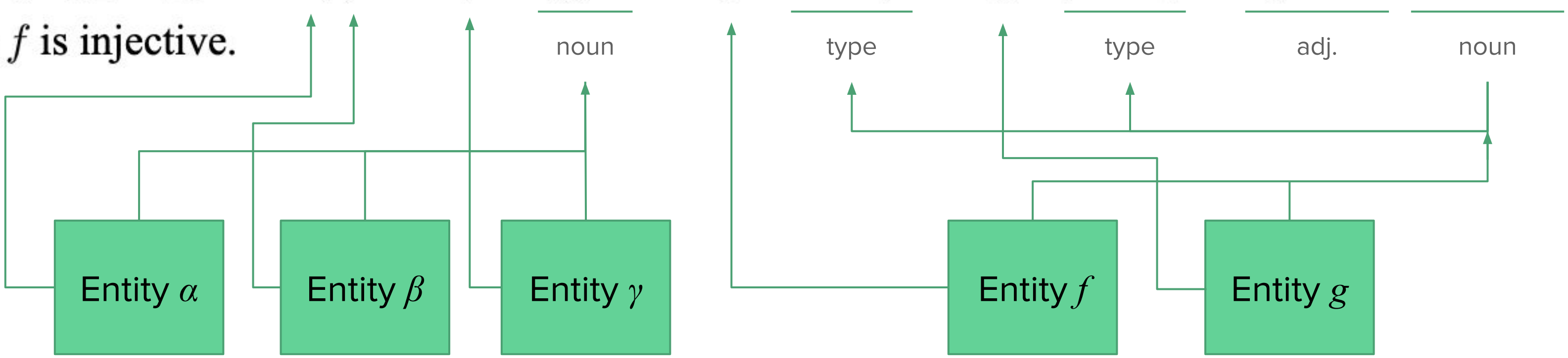
If consecutive entities have compatible data, we can merge their introductions into a single sentence.

```
theorem inj_comp {α β γ} {f : α → β} {g : β → γ} (hf : injective f) (hg : injective g) :
  injective (g ∘ f) :=
```

Theorem (`inj_comp`). Let α, β and γ be types. Let $f\ :\alpha\to\beta$ and $g\ :\beta\to\gamma$ be injective functions. Then $g\ \circ f$ is injective.

```
theorem inj_comp {α β γ} {f : α → β} {g : β → γ} (hf : injective f) (hg : injective g) :
  injective (g ∘ f) :=
```

Theorem (`inj_comp`). Let α, β and γ be types. Let $f : \alpha \to \beta$ and $g : \beta \to \gamma$ be injective functions. Then $g \circ f$ is injective.



Propositions to English

```
#english_prop ∀ {α β γ : Type _} {f : α → β} {g : β → γ},
  injective f → injective g → injective (g ∘ f)
```

for all types α, types β, types γ, injective functions $f : \alpha \to \beta$ and injective functions $g : \beta \to \gamma$, $g \circ f$ is injective

Grammatical agreement

With English, there are two main grammatical features that need to be observed:

- Plurality
  - Verbs: is/are
  - Nouns: function/functions
- Articles
  - *A* function
  - *An* injective function

We avoid tenses, but we do make use of the subjunctive for "to be":

- Let n *be* a natural number.
- Suppose n *is* a natural number.

Describing proofs

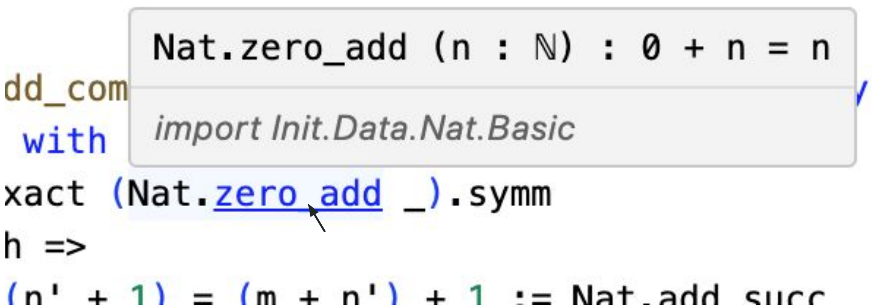
The next big part: representing proofs

Main elements:

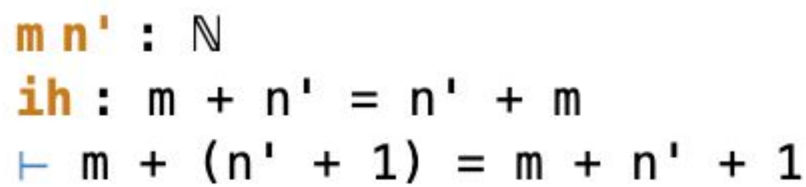
- Deducing tactic proof structure
- Tactic describers
- Proof term decompiler

InfoTrees

Original purpose: providing all the information one sees in the VS Code IDE, including mouseover text, jump-to-definition, and the Infoview.



Infoview (goal states)



Excerpt of an InfoTree

```
have key : f x = f y := by
  exact hg _ _ h
```


```
<Node elaborator="Lean.Parser.Tactic._aux_Init_Tactics___macroRules_Lean_Parser_Tactic_tacticHave__1" type="tactic">
  <Node elaborator="Lean.Parser.Tactic._aux_Init_Tactics___macroRules_Lean_Parser_Tactic_tacticRefine_lift__1" type="tactic
    <Node elaborator="Lean.Elab.Tactic.evalFocus" type="tactic">
      <Node elaborator="Lean.Elab.Tactic.evalFocus" type="tactic"></Node>
      <Node elaborator="Lean.Elab.Tactic.evalTacticSeq" type="tactic">
        <Node elaborator="Lean.Elab.Tactic.evalTacticSeq1Indented" type="tactic">
          <Node elaborator="Lean.Elab.Tactic.evalParen" type="tactic">
            <Node elaborator="Lean.Elab.Tactic.evalTacticSeq" type="tactic">
              <Node elaborator="Lean.Elab.Tactic.evalTacticSeq1Indented" type="tactic">
                <Node elaborator="Lean.Elab.Tactic.evalRefine" type="tactic">
                  <Node binder="false" elaborator="Lean.Elab.Term.elabNoImplicitLambda" type="term">
                    <type>x = y</type>
                    <Node binder="false" elaborator="Lean.Elab.Term.expandHave" type="term">
                      <type>x = y</type>
                      <Node type="macro_expansion">
                        <from>have key : f x = f y := by exact hg _ _ h;
                        ?_</from>
                        <to>let_fun key : f x = f y := by exact hg _ _ h;
                        ?_</to>
                        <Node binder="false" elaborator="Lean.Elab.Term.elabLetFunDecl" type="term">
                          <type>x = y</type>
                          <Node binder="false" elaborator="«_aux_Init_Notation___macroRules_term_=__2»" type="term">
                            <type>Prop</type>
                            <Node type="macro_expansion">
                              <from>f x = f y</from>
                              <to>binrel% Eq† (f x)(f y)</to>
                              <Node binder="false" elaborator="Lean.Elab.Term.Op.elabBinRel" type="term">
                                <type>Prop</type>
                                <Node binder="false" type="term">
                                  <type>Prop</type>
```


…

Tactic describers

These consume these trees and create hierarchical explanations.

```
/-- A tactic describer is a function that takes a `TacticTree` and returns a `ProofStep`.

If it does not want to be responsible for the tree, then it can use `throwInapplicableDescriber`. -/
def TacticDescriber := TacticTree → DescriberM ProofStep
```

Tactics are semi-hierarchical

We want to recover the true proof tree.

Many tactics produce *side goals* that are solved for later.

```
constructor
· trivial
· rfl
```

```
obtain ⟨V, V_in, V_op, hV⟩ : ∃ V ∈ Nhd x, IsOpen V ∧ f '' (V ∩ A) ⊆ V'
· rcases (hf x).2 V' V'_in with ⟨U, U_in, hU⟩
  rcases exists_IsOpen_Nhd U_in with ⟨V, V_in, V_op, hVU⟩
  use V, V_in, V_op
  exact (image_subset f $ inter_subset_inter_left A hVU).trans hU
```

Tactic describers may elect to collect side goals.

```
/-- For every sibling tree that's a sidegoal for tree, use `f` to filter them,
and return the chosen ones. The list of sibling trees is updated to be unchosen trees. -/
def getSideGoalsFor (tree : TacticTree) (f : TacticTree → MVarId → DescriberM Bool) :
    DescriberM (Array TacticTree)
```

Explanations

The output of a tactic describer is a piece of a structured document. These support:

- Block indentation
- Paragraph breaks
- Text with a (+)/(-) to replace some text with other text
- Clickable words that show additional text
- Tooltips
- Goal states
- Multiline equations

There is a JavaScript webapp that renders `Explanation`s.

This webapp could be used for other purposes.

Proof term explanations

For a number of tactics, we decompile the proof term it generates into an equivalent sequence of simpler tactics and then compile *that* into English.

> **Reason 1.** We want to avoid recapitulating tactic implementations.
>
> **Reason 2.** The "Laziness Principle": *a Lean author writes primarily to be understood by the computer*.
>
> **Reason 3.** Not everything that a computer wants to see is similarly desired by a human reader.

This is used by `exact`, `apply`, `refine`, and many others.

```
exact hf x y key
```

Using our assumption that f is injective and our assumption that $f(x) = f(y)$ proves $x = y$.

Proof term explanations, a decompiler

Local context + expected type + proof term

⬇

Synthesized tactic trees, with synthesized intermediate goal states

⬇

Rendered Explanations

Example: if introducing a variable requires unfolding a goal, there is an explainer inserting an intermediate step: “by definition of injectivity, …”

LeanTeX

```
#latex 1 + (2 + 3)
```

$$1 + (2 + 3)$$

```
#latex (1 + 2) + 3
```

$$1 + 2 + 3$$

```
#latex Nat × Nat × Nat × Fin 37
```

$$\mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}_{<37}$$

```
#latex s.sum (λ x => x + 1)
```

$$\sum_{x \in s} (x + 1)$$

```
#latex 1+1/(1+1/(1+1/(1+1/(1+1/(1+1)))))
```

$$1 + \frac{1}{1 + \frac{1}{1+1/(1+1/(1+1/(1+1)))}}$$

```
#latex 2^2^2
```

$$2^{2^2}$$

```
#latex (2^2)^2
```

$$(2^2)^2$$

```
#latex s.sum (λ x => 2 * x)
```

$$\sum_{x \in s} 2 \cdot x$$

```
#latex s.sum (λ x => x + 1) * a
```

$$\left(\sum_{x \in s} (x + 1)\right) \cdot \mathrm{a}$$

```
#latex a * s.sum (λ x => x + 1)
```

$$\mathrm{a} \cdot \sum_{x \in s} (x + 1)$$

```
#latex s.sum (λ x => x + 1) + a
```

$$\left(\sum_{x \in s} (x + 1)\right) + \mathrm{a}$$

```
#latex a + s.sum (λ x => x + 1)
```

$$\mathrm{a} + \sum_{x \in s} (x + 1)$$

```
#latex s.sum (λ x => x + 1) * s.sum (λ x => 2 * x)
```

$$\left(\sum_{x \in s} (x + 1)\right) \cdot \sum_{x \in s} 2 \cdot x$$

Ongoing work and investigations

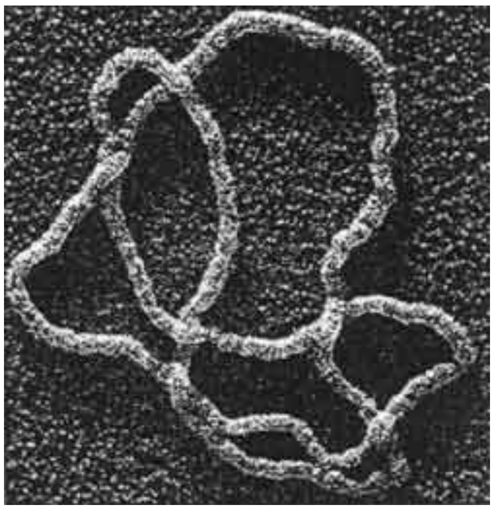
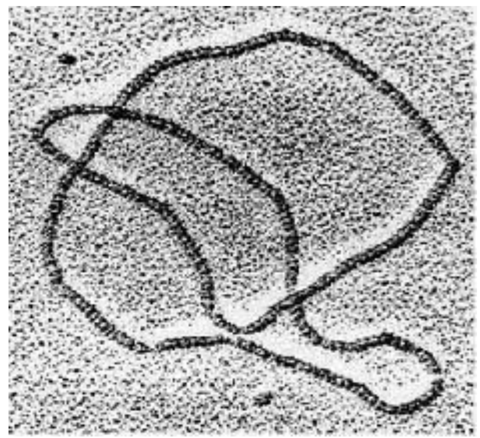
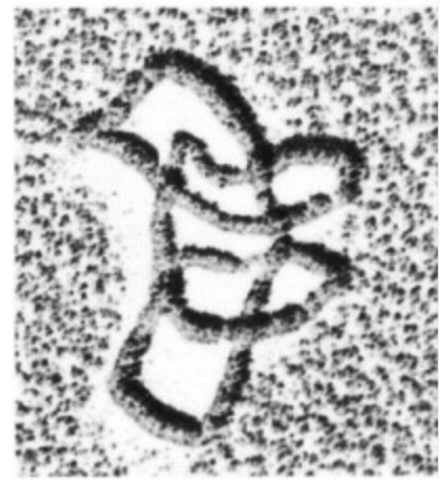
As we saw, we have a working informalizer!

There is plenty more to do, and plenty more ontology engineering to consider.

- Formalization
- Informalization
- Knots and algorithms
- The future

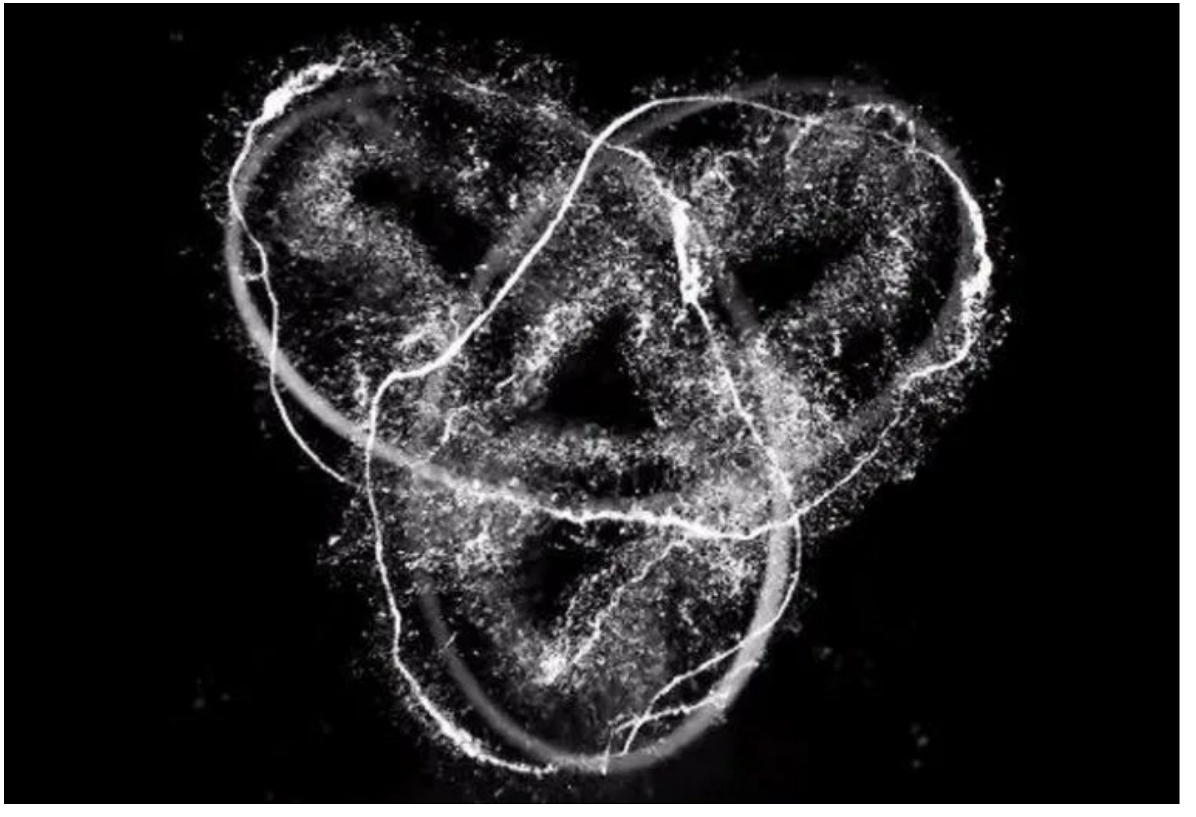
Knots

In DNA:

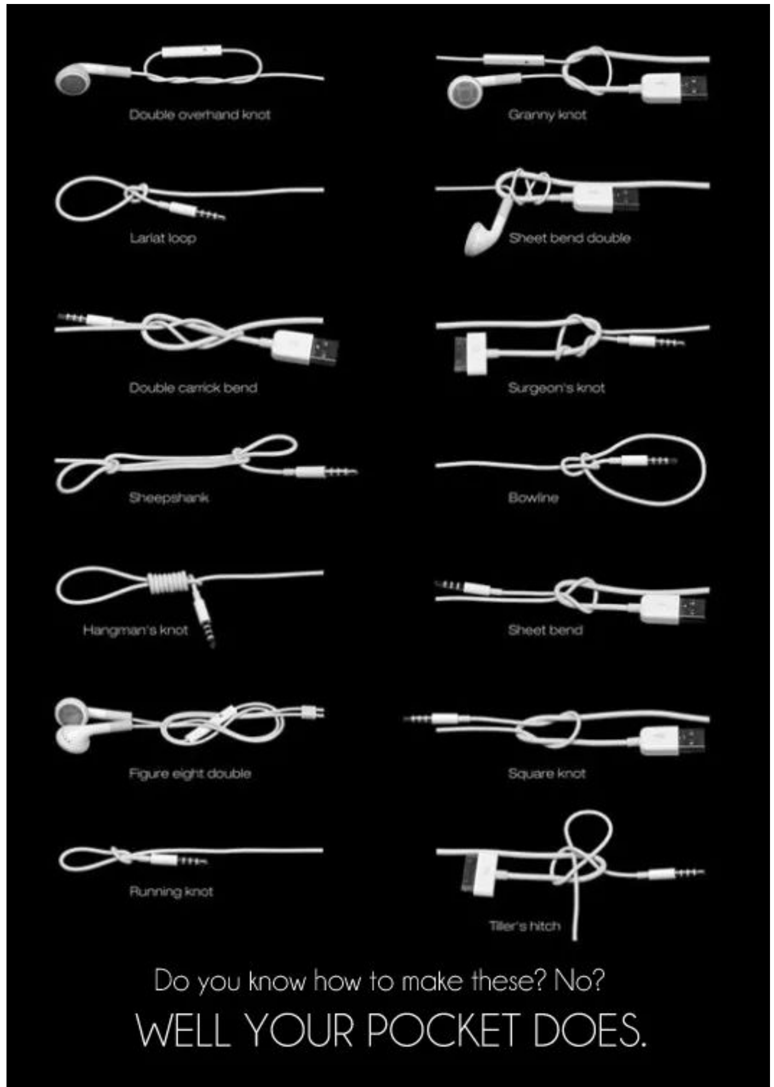
Knots

In physics:



Knots

In life:



Knots

In mathematics:

Definition. A *knot* is a closed loop in 3D space.

Classifying knots

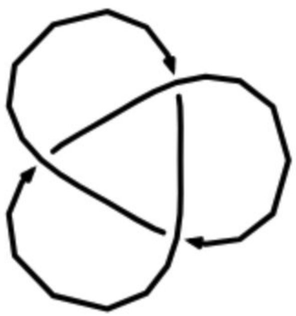
What are all the knots?

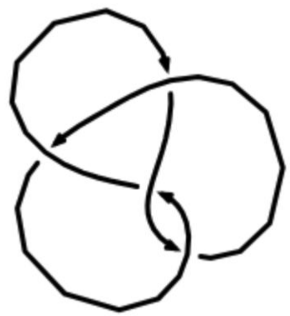
How can we distinguish different knots?

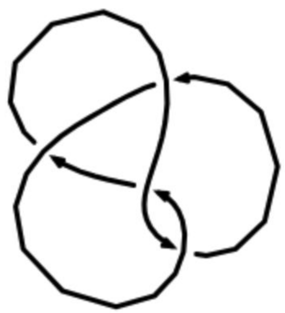
Classifying knots

What are all the knots?

How can we distinguish different knots?







Classifying knots

What are all the knots?

How can we distinguish different knots?

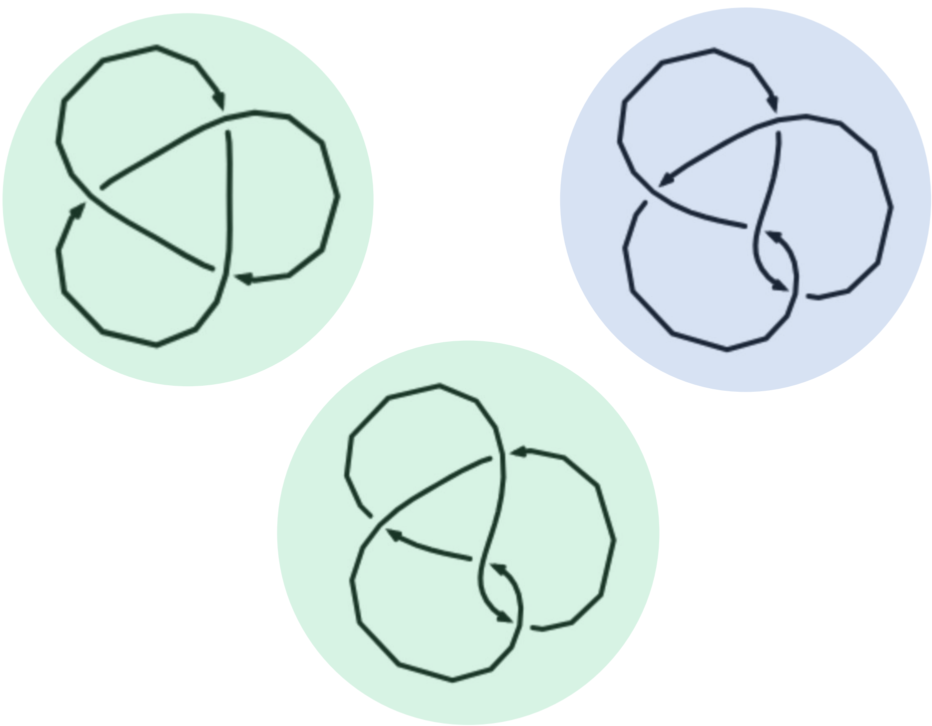
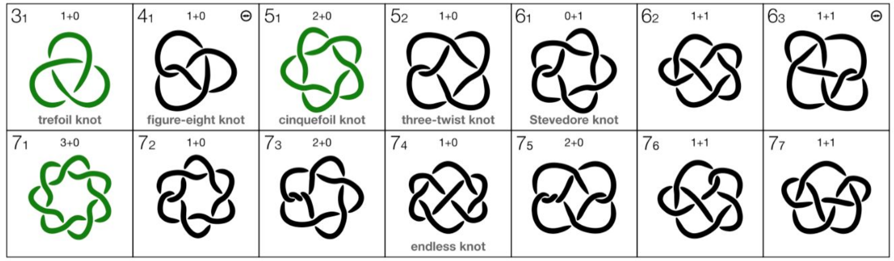
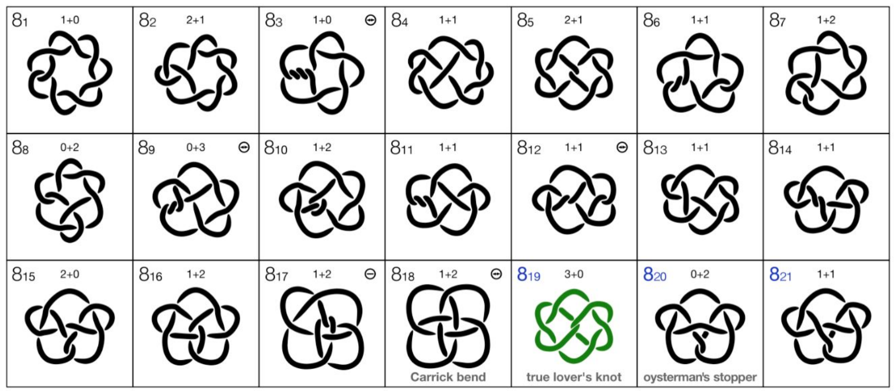


TABLE OF PRIME KNOTS

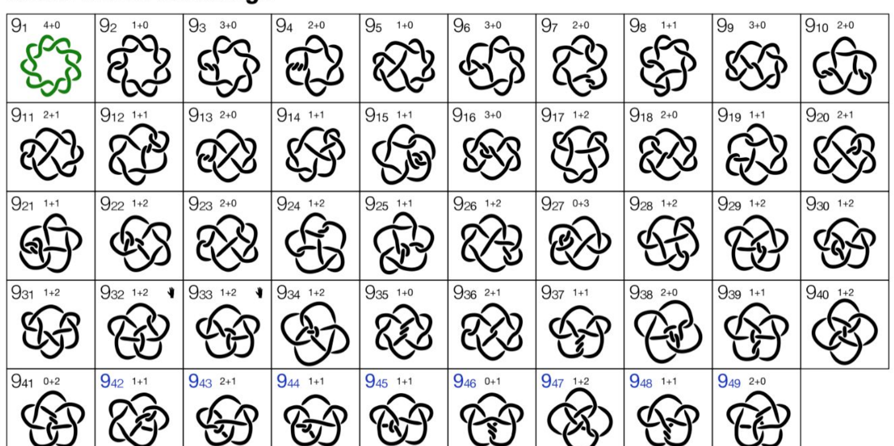
Knots with up to 7 crossings

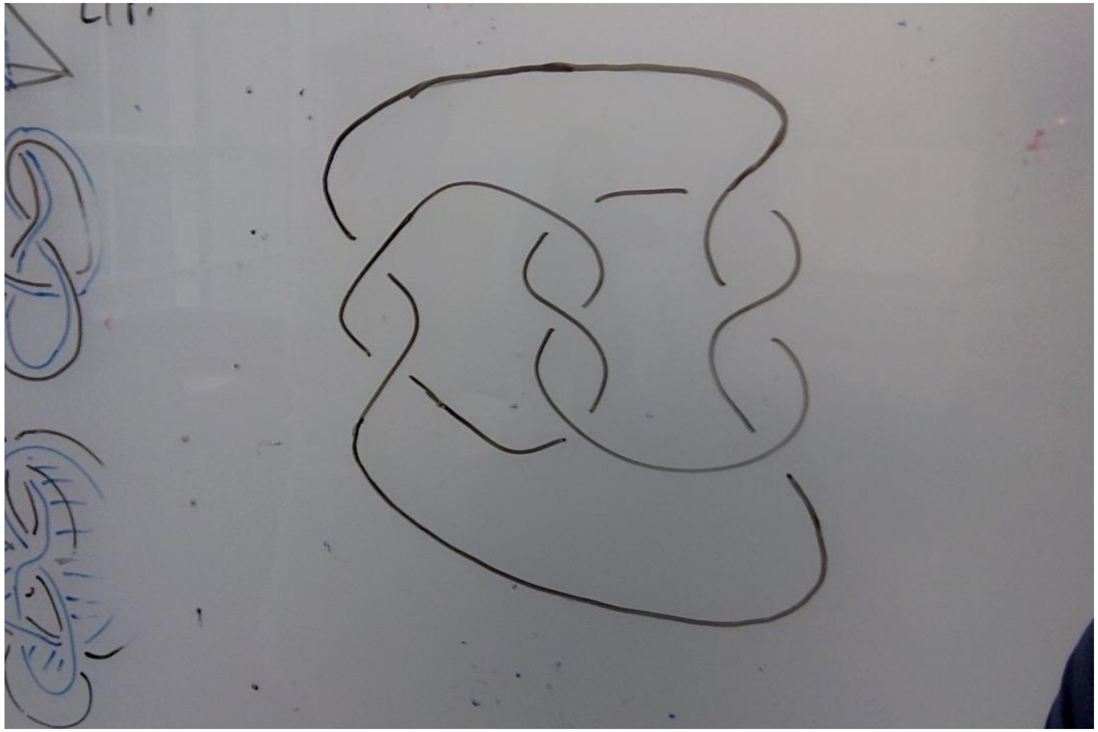


Knots with 8 crossings



Knots with 9 crossings





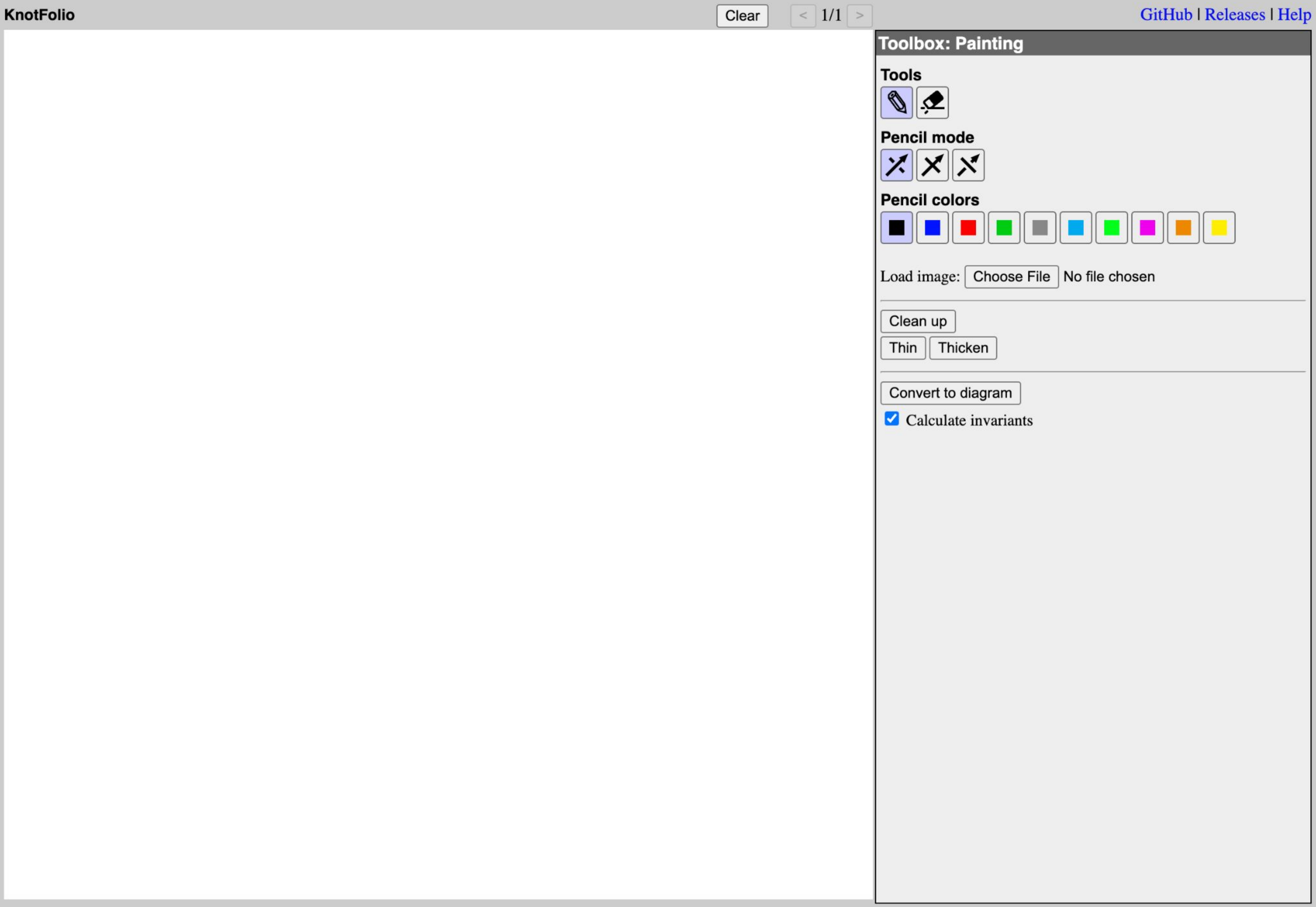
"Papers contain knot diagrams, and we manipulate knots on chalkboards. Is there software to read knot diagrams from photographs?"

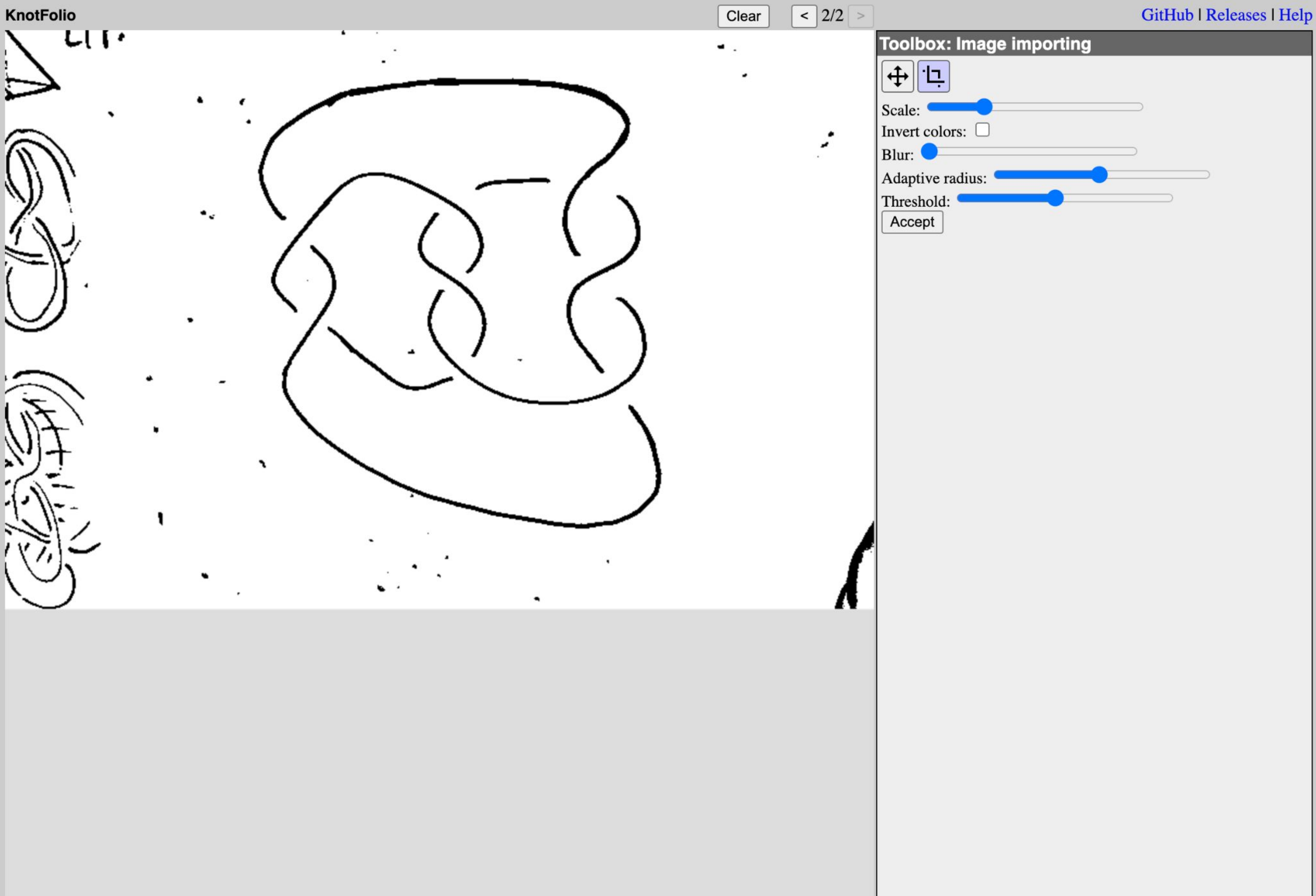
– Question in *Perspectives on Dehn Surgery* at ICERM 2019

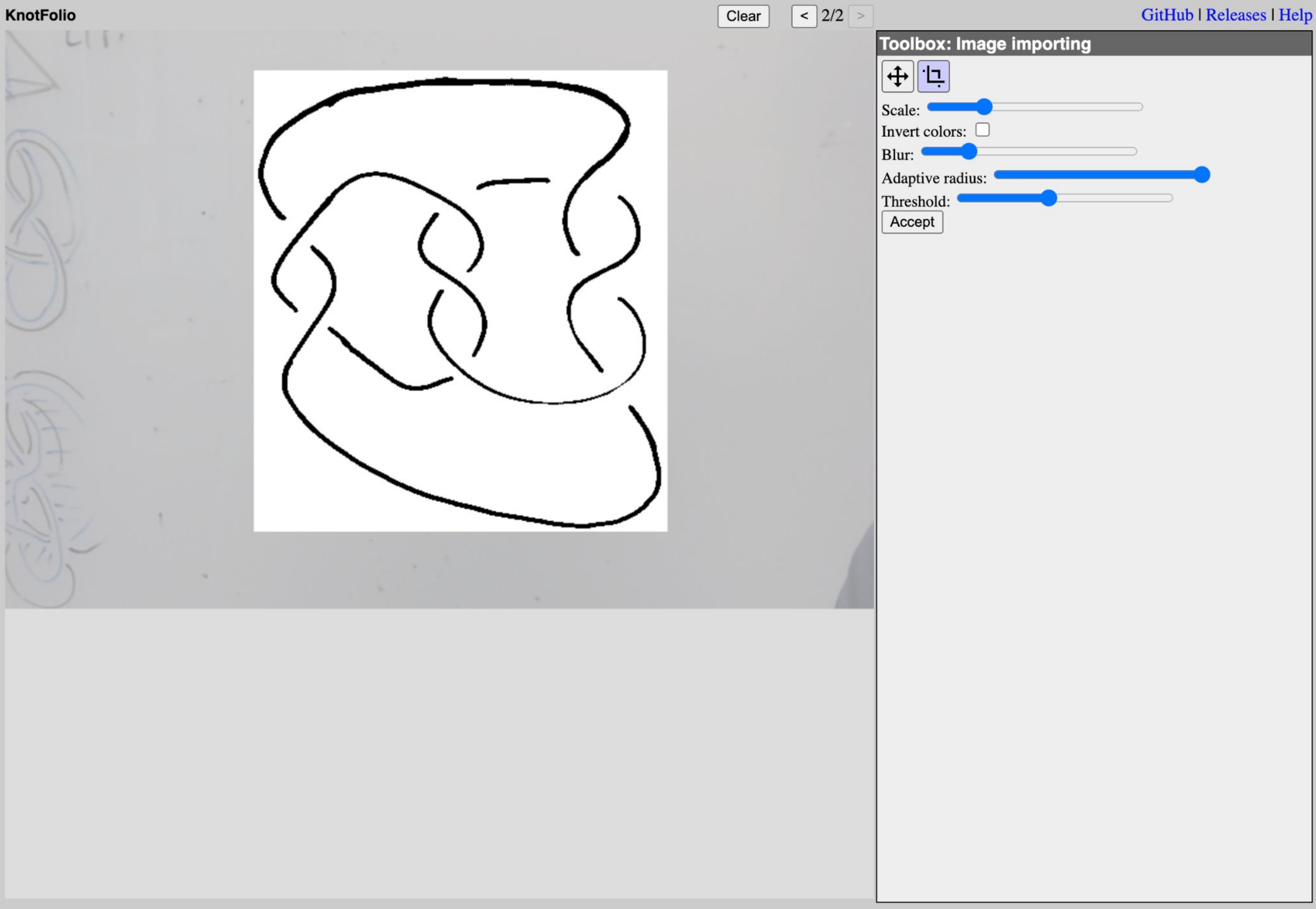
There was no such software,
but I had an idea for an algorithm.

There was no such software,
but I had an idea for an algorithm.

This led to KnotFolio
https://knotfol.io/


KnotFolio
Clear
1/1
GitHub | Releases | Help
Toolbox: Painting
Tools
Pencil mode
Pencil colors
Load image: Choose File No file chosen
Clean up
Thin
Thicken
Convert to diagram
Calculate invariants

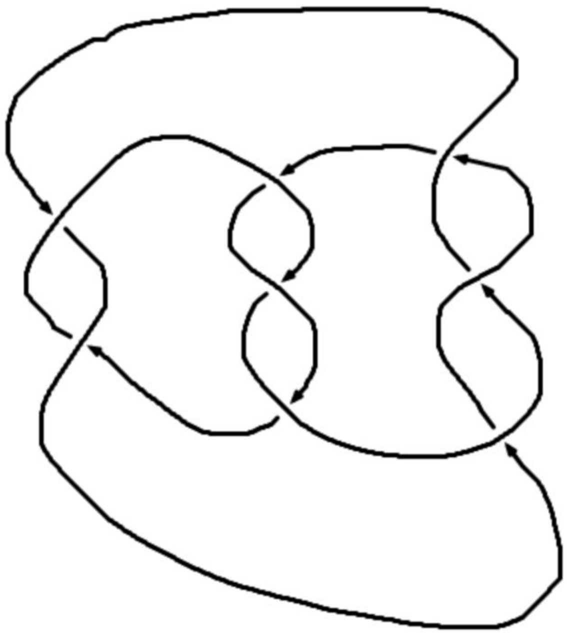

KnotFolio
Clear
<
2/2
>
GitHub | Releases | Help
Toolbox: Image importing
Scale:
Invert colors:
Blur:
Adaptive radius:
Threshold:
Accept


KnotFolio
Clear
<
2/2
>
GitHub | Releases | Help
Toolbox: Image importing
Scale:
Invert colors:
Blur:
Adaptive radius:
Threshold:
Accept

KnotFolio

Clear < 4/4 >

GitHub | Releases | Help



Toolbox: Diagrams

Modification tools

Mirror | Invert | Make alternating | Auto-color

Convert to drawing

Isotopy tools

Beautify | Reset view

Diagram information Pretty Mathematica

Crossings:	8
Components:	1
Writhe:	-2
Bridges:	6
Can. genus:	3
Turaev genus:	1
Properties:	plus-adequate

▼ Linking matrix

-2

▶ Seifert form

PD: KnotTheory Oriented KnotTheory SnapPy

```
PD[X[2,7,3,8], X[6,1,7,2], X[3,12,4,13], X[11,4,12,5],
X[13,9,14,8], X[16,5,1,6], X[9,15,10,14], X[15,11,16,10]]
```

DT: [6, -12, 16, 2, -14, -4, -8, -10]

Kauffman bracket:
$-A^{-10} + 2A^{-6} - A^{-2} + 2A^{2} - A^{6} + A^{10} - A^{14}$

Identification

Candidates: 8_20

Invariants

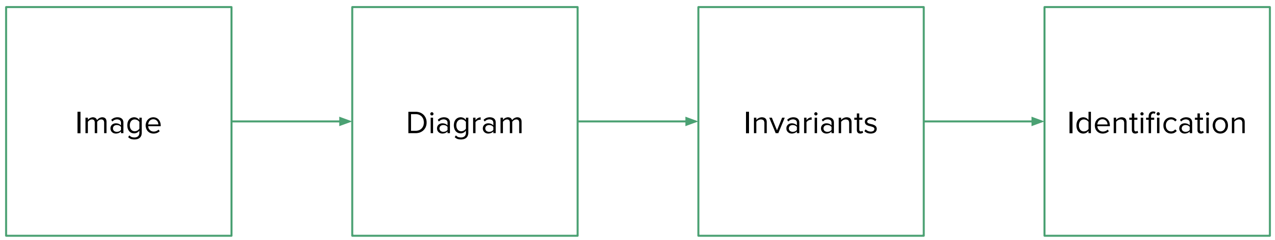
Determinant: 9

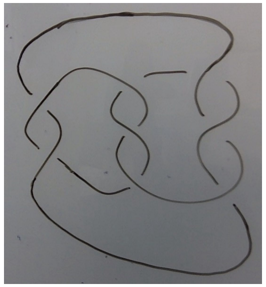
(Cabled) Jones polynomials: Next

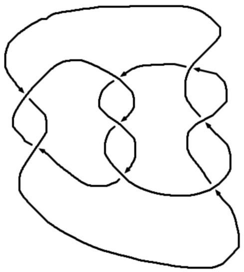
$V_1 = -t + 2 - t^{-1} + 2t^{-2} - t^{-3} + t^{-4} - t^{-5}$

Conway potential:
$1 + 2z^2 + z^4$

Alexander polynomials:

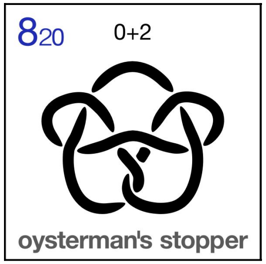






$$V_1 = -t + 2 - t^{-1} + 2\,t^{-2} - t^{-3} + t^{-4} - t^{-5}$$

$$\Delta^0(t) = 1 - 2\,t + 3\,t^2 - 2\,t^3 + t^4$$




```
PD[Xm[2,7,3,8], Xm[6,1,7,2],
   Xm[3,12,4,13], Xm[11,4,12,5],
   Xp[13,9,14,8], Xm[16,5,1,6],
   Xp[9,15,10,14], Xp[15,11,16,10]]
```

Algorithms in KnotFolio

I wrote all the algorithms myself in JavaScript.

- Jones polynomial
- Alexander polynomial and Conway potential
- Canonical Seifert genus
- Seifert form
- Turaev genus
- Crossing number, writhe, bridge number counting, linking matrix

“Can KnotFolio help us identify *virtual* knots?”

“Can KnotFolio help us identify *virtual* knots?”

Yes, but the Jones and Alexander polynomials aren’t enough.

There are many virtual knots, and coincidences.

Crossings	Count
0	1
1	0
2	1
3	7
4	108
5	2448

Table 4. Virtual knots up to four crossings with non-unique 1- and 2-cabled Jones polynomials. Each cell consists of virtual knots with the same such polynomials.

0.1, 4.55, 4.56, 4.76, 4.77	2.1, 4.4, 4.5, 4.54, 4.74	3.3, 4.63	4.1, 4.3, 4.7, 4.53, 4.73
4.2, 4.6, 4.8, 4.12, 4.75	4.13, 4.59, 4.107	4.19, 4.42	4.26, 4.97
4.28, 4.83	4.95, 4.101		

Table 5. Virtual knots up to five crossings with non-unique 1-, 2-, and 3-cabled Jones polynomials. Each cell consists of virtual knots with the same such polynomials.

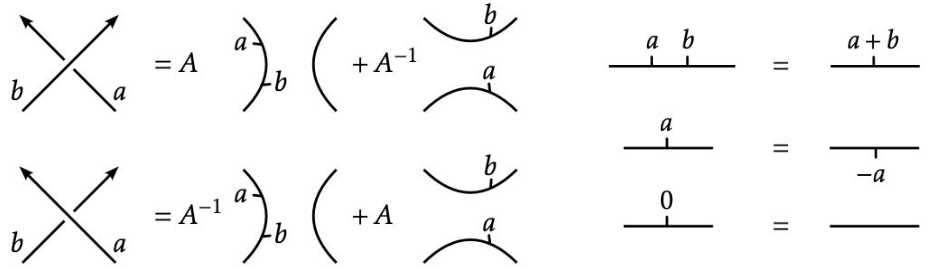
5.15, 5.116	5.16, 5.117	5.23, 5.71	5.24, 5.73	5.25, 5.72
5.26, 5.74	5.58, 5.94	5.59, 5.95	5.60, 5.96	5.61, 5.97
5.196, 5.1662	5.197, 5.1657	5.204, 5.1670	5.205, 5.1665	5.287, 5.1168
5.294, 5.1175	5.295, 5.1176	5.302, 5.1183	5.661, 5.662	5.754, 5.763
5.757, 5.760	5.807, 5.1672	5.808, 5.1674	5.809, 5.1673	5.810, 5.1675
5.811, 5.814, 5.1676, 5.1679		5.812, 5.813, 5.1677, 5.1678		5.1113, 5.1124
5.1116, 5.1121	5.1184, 5.1187	5.1186, 5.1189	5.1190, 5.1192, 5.1193, 5.1195	
5.1191, 5.1194	5.2322, 5.2411			

Furthermore, the n-cabled Jones polynomial of a virtual knot with c crossings is by definition a polynomial with 2^{nc^2} terms.

For example, the 3-cabled Jones polynomial of a 5-crossing virtual knot has 35,184,372,088,832 terms.

Even with tricks, it is too slow to run in an interactive application.

Arrow polynomials



I discovered a way to compute arrow polynomials of virtual knots, which are a refinement of Jones polynomials. [Miller 2023]

I implemented the algorithms once in JavaScript, and once again in Lean.

I did not formally verify the Lean code, however I did include invariants.

```
structure Poly where
  terms : Array Monomial
  incr : (terms.map Monomial.exponents).strictIncreasing

structure Monomial where
  exponents : Array Int
  coeff : Int
  coeff_nonzero : coeff ≠ 0 := by simp [*]
  exp_norm : exponents.back? ≠ some 0 := by simp [*]
```

```
structure ATLP where
  (fst snd : Nat)
  whiskers : Int
  normalized : fst ≤ snd

structure ATLD where
  coeff : Poly
  paths : Array ATLP
  coeff_nonzero : coeff ≠ 0 := by simp [*]
  ordered : paths |>.map ATLP.toPair |>.strictIncreasing
  nodup : (paths.toList.bind ATLP.indices).nodup
```

Evaluation

The 1-, 2-, and 3-cabled arrow polynomials of all virtual knots up to 5 crossings took 2.8 days of computer time with the Lean implementation.

TABLE 2. Virtual knots up to five crossings with non-unique 1- and 2-cabled arrow polynomials. Each cell consists of virtual knots with the same such polynomials.

5.196, 5.1662	5.197, 5.1657	5.204, 5.1670	5.205, 5.1665
5.287, 5.1168	5.294, 5.1175	5.295, 5.1176	5.302, 5.1183
5.757, 5.760	5.1113, 5.1124	5.2322, 5.2411	

TABLE 3. Virtual knots up to five crossings with non-unique 1-, 2-, and 3-cabled arrow polynomials. Each cell consists of virtual knots with the same such polynomials. Pairs indicated by † are distinguishable by their Alexander polynomials.

5.196, 5.1662	5.197, 5.1657	5.204, 5.1670
5.205, 5.1665	5.287, 5.1168 †	5.294, 5.1175 †
5.295, 5.1176 †	5.302, 5.1183 †	5.2322, 5.2411

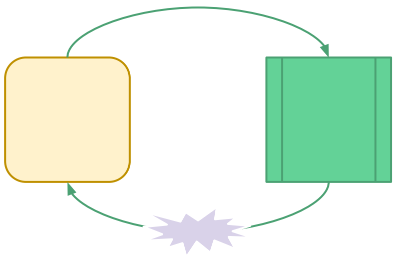
Improvements

Now KnotFolio can uniquely identify 2543 out of 2565 small virtual knots.

While having two implementations yielding the same results is nice, ideally we would have a formally verified algorithm!

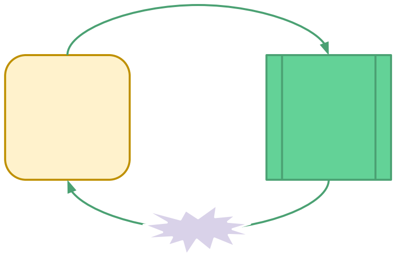
- Formalization
- Informalization
- Knots and algorithms
- The future

Informalization



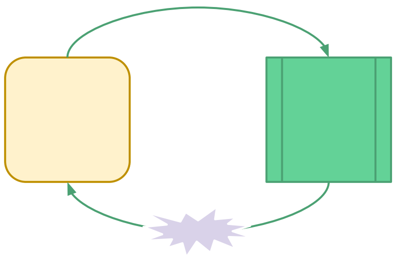
- Write interactive textbooks and evaluate their use pedagogically
  - Target: introduction to proofs and discrete mathematics courses at UCSC
  - Test out UI concepts for navigation and surfacing salient details
  - Plan: obtain grants to support curriculum development

Informalization



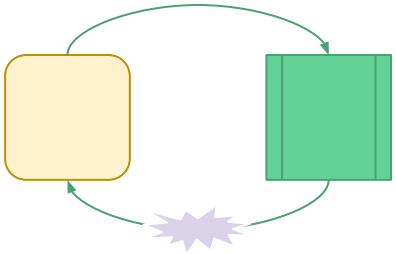
- Write interactive textbooks and evaluate their use pedagogically
  - Target: introduction to proofs and discrete mathematics courses at UCSC
  - Test out UI concepts for navigation and surfacing salient details
  - Plan: obtain grants to support curriculum development
- Integrate into the mathlib documentation
  - Solve scaling issues (need explainers for the 73955 definitions and 135484 theorems)

Informalization



- Write interactive textbooks and evaluate their use pedagogically
  - Target: introduction to proofs and discrete mathematics courses at UCSC
  - Test out UI concepts for navigation and surfacing salient details
  - Plan: obtain grants to support curriculum development
- Integrate into the mathlib documentation
  - Solve scaling issues (need explainers for the 73955 definitions and 135484 theorems)
- Collaborate with AI researchers
  - Autoformalization researchers send frequent inquiries about the prototype
  - Explore ML augmentations to informalization (style, level of detail, etc.)

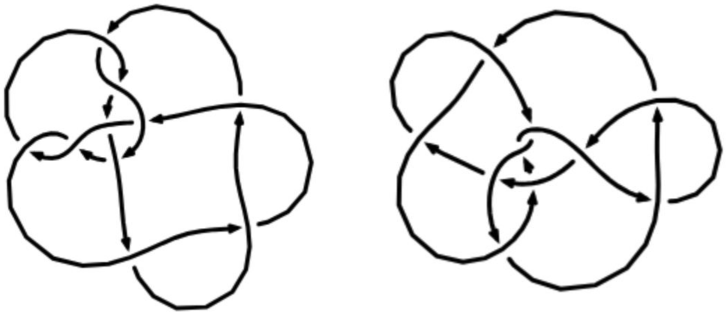
Informalization



- Write interactive textbooks and evaluate their use pedagogically
  - Target: introduction to proofs and discrete mathematics courses at UCSC
  - Test out UI concepts for navigation and surfacing salient details
  - Plan: obtain grants to support curriculum development
- Integrate into the mathlib documentation
  - Solve scaling issues (need explainers for the 73955 definitions and 135484 theorems)
- Collaborate with AI researchers
  - Autoformalization researchers send frequent inquiries about the prototype
  - Explore ML augmentations to informalization (style, level of detail, etc.)
- Create a *correct* informalizer
  - Use controlled natural language such as Formal Theory Language (ForTheL)
  - Develop a full ontology for structured proofs
  - Prove that informalizer output is semantically equivalent to original Lean
  - Apply this framework to other domains — element of expert systems

    answer "why" with reliable arbitrarily detailed explanations

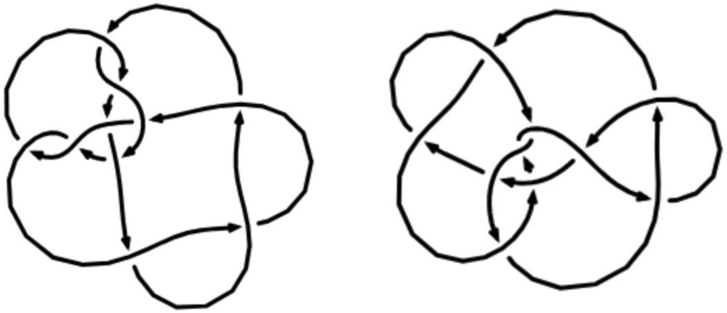
Formal knot theory



People use my KnotFolio program.

The algorithms are carefully-written JavaScript — they should be formally verified.

Formal knot theory



People use my KnotFolio program.

The algorithms are carefully-written JavaScript — they should be formally verified.

Long term project:

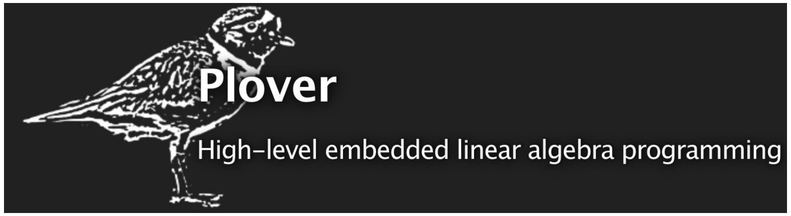
- Formalization of piecewise linear manifolds, 3-manifold topology, 2D and 3D meshes, normal surface theory, knot theory, skein theory, planar algebras, quantum topology, etc.
- Verified algorithms for low-dimensional topology, including knot theory.
- Add support to popular software such as SnapPy and Regina to output proof certificates.

Carrying this out requires one foot in theoretical mathematics and another in CS.

Verified compilation projects

with Scott Kovach (Stanford)

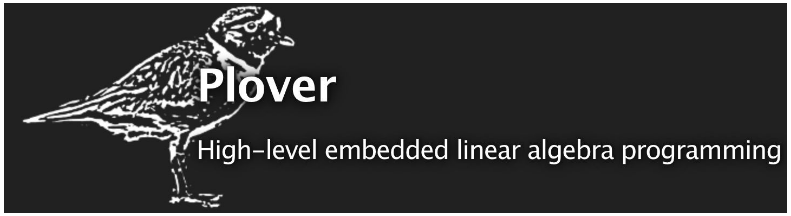
At Swift Navigation in 2015, we developed a compiler for programming with linear algebra and tensors, to target memory-constrained systems.



Verified compilation projects

with Scott Kovach (Stanford)

At Swift Navigation in 2015, we developed a compiler for programming with linear algebra and tensors, to target memory-constrained systems.



We are now working on languages for high-level programming with tensors, probability distributions, associative arrays, and so on, using Lean.

Current project: Design compilation strategies to lower these computations to efficient kernels, and prove (in Lean) that the transformation is correct.

Last thoughts...

Lean is a versatile (and fun!) language for programming and formalization.

The fruits of formalization go beyond verifying truth.

Let's enrich education, mathematics, and engineering with these tools! 🐙

Kyle Miller, PhD
🌐 https://kmill.github.io/ ✉️ kymiller@ucsc.edu

KnotFolio	https://knotfol.io/
arrow_poly	https://github.com/kmill/arrow_poly
Lean FRO	https://lean-fro.org/

More slides

What is Lean?

The Lean theorem prover

Lean is one of the newest interactive theorem provers.

- 1973 Mizar
- 1986 Isabelle
- 1989 Coq
- 1999 Agda
- 2013 Lean 1
- 2021 Lean 4

It was initially developed at Microsoft Research and Carnegie Mellon University by Leonardo de Moura, Jeremy Avigad, and others.

The Lean theorem prover

- Lean is a functional programming language
- Like Coq, its type theory is based on the Calculus of Inductive Constructions
- *Dependent types*: argument types may depend on previous arguments
- Every mathematical proposition can be encoded as a type;
  Proofs are "programs"! (the Curry–Howard correspondence)
- Unlike Coq or Agda, has *proof irrelevance* and *quotient types*
- Its theory is equiconsistent with ZFC plus some inaccessible large cardinals
  [Carneiro 2019]

These features together work well for mathematics,
and (partly by accident) mathematicians form a large fraction of Lean users.

Lean 4

- Is a full programming language that is also an interactive theorem prover
  - Much of Lean 4 is written in Lean 4
- Compiles to C
- Can prove theorems in Lean 4 about programs written in Lean 4
- Can write programs in Lean 4 that write programs written in Lean 4
  - For example, tactics are proof-writing programs
- Can extend Lean 4 from within Lean 4
  - Syntax, macros, elaboration rules, tactics, etc.
  - One language for everything!

Lean Focused Research Organization

As of July 2023, the Lean FRO is a nonprofit dedicated to advancing formal mathematics, tackling scalability, usability, and proof automation.

The Lean FRO has assembled a team to develop Lean.

Its mission includes supporting broad applications including software and hardware verification and AI research for mathematics and code synthesis.

It is on a five-year mission to become self-sustainable.

Examples

```
/-- Linked lists -/
inductive List (α : Type u) where
  | nil : List α
  | cons (head : α) (tail : List α) : List α

infixr:67 " :: " => List.cons

/-- Notation `[x,y,z]` for `x :: y :: z :: List.nil` -/
syntax (name := List.brackets) "[" term,* "]" : term

macro_rules (kind := List.brackets)
  | `([]) => `(List.nil)
  | `([$x]) => `($x :: List.nil)
  | `([$x, $xs,*]) => `($x :: [$xs,*])
```

Examples

```
def List.length : List α → ℕ
  | [] => 0
  | _ :: xs => xs.length + 1

def List.head (xs : List α) (h : length xs ≠ 0) : α :=
  match xs with
  | [] => absurd rfl h /- `h` is specialized to `length [] ≠ 0`
                          but `length [] = 0` by definition. -/
  | x :: _ => x

-- `match` can see the absurdity of `h` in the `nil` case automatically.
def List.head' (xs : List α) (h : length xs ≠ 0) : α :=
  match xs, h with
  | x :: _, _ => x

def List.append (xs ys : List α) : List α :=
  match xs with
  | [] => ys
  | x :: xs' => x :: xs'.append ys
```

Examples

```
theorem List.length_append (xs ys : List α) :
    (xs.append ys).length = xs.length + ys.length := by
  induction xs with
  | nil =>
    /- goal is `([].append ys).length) = [].length + ys.length` -/
    simp [append, length]
  | cons x xs' ih =>
    /- ih is `(xs'.append ys).length = xs'.length + ys.length`
       goal is `((x :: xs').append ys).length = (x :: xs').length + ys.length` -/
    simp [append, length]
    /- goal is `(xs'.append ys).length + 1 = xs'.length + 1 + ys.length` -/
    rw [ih]
    /- goal is `xs'.length + ys.length + 1 = xs'.length + 1 + ys.length` -/
    ring

/- Tersely: -/
theorem List.length_append' (xs ys : List α) :
    (xs.append ys).length = xs.length + ys.length := by
  induction xs <;> simp! [*]; ring
```

Examples

```
#print List.length_append
```

```
theorem Mine.List.length_append.{u_1} : ∀ {α : Type u_1} (xs ys : List α),
  List.length (List.append xs ys) = List.length xs + List.length ys :=
fun {α} xs ys ↦
  List.rec (of_eq_true ((congrArg (Eq (List.length ys)) (zero_add (List.length ys))).trans (eq_self (List.length ys))))
    (fun x xs' ih ↦
      id
        (Eq.mpr (id (ih ▸ Eq.refl (List.length (List.append xs' ys) + 1 = List.length xs' + 1 + List.length ys)))
          (Mathlib.Tactic.Ring.of_eq
            (Mathlib.Tactic.Ring.add_congr
              (Mathlib.Tactic.Ring.add_congr (Mathlib.Tactic.Ring.atom_pf (List.length xs'))
                (Mathlib.Tactic.Ring.atom_pf (List.length ys))
                (Mathlib.Tactic.Ring.add_pf_add_lt (List.length xs' ^ Nat.rawCast 1 * Nat.rawCast 1)
                  (Mathlib.Tactic.Ring.add_pf_zero_add (List.length ys ^ Nat.rawCast 1 * Nat.rawCast 1 + 0))))
              (Mathlib.Tactic.Ring.cast_pos (Mathlib.Meta.NormNum.isNat_ofNat ℕ (Eq.refl 1)))
              (Mathlib.Tactic.Ring.add_pf_add_gt (Nat.rawCast 1)
                (Mathlib.Tactic.Ring.add_pf_add_zero
                  (List.length xs' ^ Nat.rawCast 1 * Nat.rawCast 1 +
                    (List.length ys ^ Nat.rawCast 1 * Nat.rawCast 1 + 0)))))
            (Mathlib.Tactic.Ring.add_congr
              (Mathlib.Tactic.Ring.add_congr (Mathlib.Tactic.Ring.atom_pf (List.length xs'))
                (Mathlib.Tactic.Ring.cast_pos (Mathlib.Meta.NormNum.isNat_ofNat ℕ (Eq.refl 1)))
                (Mathlib.Tactic.Ring.add_pf_add_gt (Nat.rawCast 1)
                  (Mathlib.Tactic.Ring.add_pf_add_zero (List.length xs' ^ Nat.rawCast 1 * Nat.rawCast 1 + 0))))
              (Mathlib.Tactic.Ring.atom_pf (List.length ys))
              (Mathlib.Tactic.Ring.add_pf_add_lt (Nat.rawCast 1)
                (Mathlib.Tactic.Ring.add_pf_add_lt (List.length xs' ^ Nat.rawCast 1 * Nat.rawCast 1)
                  (Mathlib.Tactic.Ring.add_pf_zero_add (List.length ys ^ Nat.rawCast 1 * Nat.rawCast 1 + 0))))))))
    xs
```

What is mathlib?

Partial mathlib overview

General algebra

Category theory category, small category, functor, natural transformation, Yoneda embedding, adjunction, monad, comma category, limits, presheafed space, sheafed space, monoidal category, cartesian closed, abelian category. See also our documentation page about category theory.

Numbers natural number, integer, rational number, continued fraction, real number, extended real number, complex number, p-adic number, p-adic integer, hyper-real number. See also our documentation page about natural numbers.

Group theory group, group morphism, group action, class formula, Burnside lemma, subgroup, subgroup generated by a subset, quotient group, first isomorphism theorem, second isomorphism theorem, third isomorphism theorem, abelianization, free group, presented group, Schreier's lemma, cyclic group, nilpotent group, permutation group of a type, structure of fintely generated abelian groups.

Rings ring, ring morphism, the category of rings, subring, localization, local ring, noetherian ring, ordered ring.

Ideals and quotients ideal of a commutative ring, quotient ring, prime ideal, maximal ideal, Chinese remainder theorem, fractional ideal, first isomorphism theorem for commutative rings.

Divisibility in integral domains irreducible element, coprime element, unique factorisation domain, greatest common divisor, least common multiple, principal ideal domain, Euclidean domain, Euclid's' algorithm, Euler's totient function (φ), Lucas-Lehmer primality test.

Polynomials and power series polynomial in one indeterminate, roots of a polynomial, multiplicity, separable polynomial, $K[X]$ is Euclidean, Hilbert basis theorem, $A[X]$ has gcd and lcm if A does, $A[X_i]$ is a UFD when A is a UFD, irreducible polynomial, Eisenstein's criterion, polynomial in several indeterminates, power series.

Algebras over a ring associative algebra over a commutative ring, the category of algebras over a ring, free algebra of a commutative ring, tensor product of algebras, tensor algebra of a commutative ring, Lie algebra, exterior algebra, Clifford algebra.

Field theory field, characteristic of a ring, characteristic zero, characteristic p, Frobenius morphism, algebraically closed field, existence of algebraic closure of a field, $\mathbb{C}$ is algebraically closed, field of fractions of an integral domain, algebraic extension, rupture field, splitting field, perfect closure, Galois correspondence, Abel-Ruffini theorem (one direction).

Homological algebra chain complex, functorial homology.

Number theory sum of two squares, sum of four squares, quadratic reciprocity, solutions to Pell's equation, Matiyasevič's theorem, arithmetic functions, Bernoulli numbers, Chevalley-Warning theorem, Hensel's lemma (for $\mathbb{Z}_p$), ring of Witt vectors, perfection of a ring.

Transcendental numbers Liouville's theorem on existence of transcendental numbers.

Representation theory representation, category of finite dimensional representations, character, orthogonality of characters.

Linear algebra

Fundamentals module, linear map, the category of modules over a ring, vector space, quotient space, tensor product, noetherian module, basis, multilinear map, alternating map, general linear group.

Duality dual vector space, dual basis.

Finite-dimensional vector spaces finite-dimensionality, isomorphism with K^n, isomorphism with bidual.

Finitely generated modules over a PID structure theorem.

Matrices ring-valued matrix, matrix representation of a linear map, determinant, invertibility.

Endomorphism polynomials minimal polynomial, characteristic polynomial, Cayley-Hamilton theorem.

Structure theory of endomorphisms eigenvalue, eigenvector, existence of an eigenvalue.

Bilinear and quadratic forms bilinear form, alternating bilinear form, symmetric bilinear form, matrix representation, quadratic form, polar form of a quadratic.

Finite-dimensional inner product spaces (see also Hilbert spaces, below) existence of orthonormal basis, diagonalization of self-adjoint endomorphisms.

See also our documentation page about linear algebra.

Topology

General topology filter, limit of a map with respect to filters, topological space, continuous function, the category of topological spaces, induced topology, open map, closed map, closure, cluster point, Hausdorff space, sequential space, extension by continuity, compactness in terms of filters, compactness in terms of open covers (Borel-Lebesgue), connectedness, compact open topology, Stone-Cech compactification, topological fiber bundle, topological vector bundle, Urysohn's lemma, Stone-Weierstrass theorem.

Uniform notions uniform space, uniformly continuous function, uniform convergence, Cauchy filter, Cauchy sequence, completeness, completion, Heine-Cantor theorem.

Topological algebra order topology, intermediate value theorem, extreme value theorem, limit infimum and supremum, topological group, completion of an abelian topological group, infinite sum, topological ring, completion of a topological ring, topological module, continuous linear map, Haar measure on a locally compact Hausdorff group.

Metric spaces metric space, ball, sequential compactness is equivalent to compactness (Bolzano-Weierstrass), Heine-Borel theorem (proper metric space version), Lipschitz continuity, Hölder continuity, contraction mapping theorem, Baire theorem, Arzela-Ascoli theorem, Hausdorff distance, Gromov-Hausdorff space.

See also our documentation page about topology.

Analysis

Topological vector spaces local convexity, Bornology, weak-* topology for dualities.

Normed vector spaces/Banach spaces normed vector space over a normed field, topology on a normed vector space, equivalence of norms in finite dimension, finite dimensional normed spaces over complete normed fields are complete, Heine-Borel theorem (finite dimensional normed spaces are proper), norm of a continuous linear map, Banach-Steinhaus theorem, Banach open mapping theorem, absolutely convergent series in Banach spaces, Hahn-Banach theorem, dual of a normed space, isometric inclusion in double dual, completeness of spaces of bounded continuous functions.

Hilbert spaces Inner product space, over R or C, Cauchy-Schwarz inequality, self-adjoint operator, orthogonal projection, reflection, orthogonal complement, existence of Hilbert basis, eigenvalues from Rayleigh quotient, Fréchet-Riesz representation of the dual of a Hilbert space, Lax-Milgram theorem.

Differentiability differentiable function between normed vector spaces, derivative of a composition of functions, derivative of the inverse of a function, Rolle's theorem, mean value theorem, Taylor's theorem, C^k function, Leibniz formula, local extrema, inverse function theorem, implicit function theorem, analytic function.

Convexity convex function, characterization of convexity, Jensen's inequality (finite sum version), Jensen's inequality (integral version), convexity inequalities, Carathéodory's theorem.

Special functions logarithm, exponential, trigonometric functions, inverse trigonometric functions, hyperbolic trigonometric functions, inverse hyperbolic trigonometric functions.

Measures and integral calculus sigma-algebra, measurable function, the category of measurable spaces, Borel sigma-algebra, positive measure, Stieltjes measure, Lebesgue measure, Hausdorff measure, Hausdorff dimension, Giry monad, integral of positive measurable functions, monotone convergence theorem, Fatou's lemma, vector-valued integrable function (Bochner integral), uniform integrability, L^p space, Bochner integral, dominated convergence theorem, fundamental theorem of calculus, part 1, fundamental theorem of calculus, part 2, Fubini's theorem, product of finitely many measures, convolution, approximation by convolution, regularization by convolution, change of variables formula, divergence theorem.

Complex analysis Cauchy integral formula, Liouville theorem, maximum modulus principle, principle of isolated zeros, principle of analytic continuation, analyticity of holomorphic functions, Schwarz lemma, removable singularity, Phragmen-Lindelöf principle, fundamental theorem of algebra.

Distribution theory Schwartz space.

Probability Theory

Definitions in probability theory probability measure, independent events, independent sigma-algebras, conditional probability, conditional expectation.

Why Lean? Why not $THEOREM_PROVER?

Lean happens to have a community of research mathematicians surrounding it; it feels comfortably set-theory-ish to them.

The community formalizes things that are interesting to mathematicians:

- Perfectoid spaces [Buzzard, Commelin, and Massot 2020]
- Scholze's main theorem of liquid modules [Commelin, et al. 2022]
- The existence of a sphere eversion [Massot, Nash, and Van Doorn 2023]
- The Polynomial Freiman-Ruzsa conjecture [Terence Tao, et al. 2023]

(To be clear, projects by other communities are not uninteresting!)

Why formalize mathematics?

Mathematicians seem not to be so interested in computer verification of proofs – while mistakes do slip into most papers, they're often believed to be "minor" since "the *idea* is generally right."

So why go through all this effort?

1. Crystallization of knowledge
2. Compatibility of definitions between papers (like Hales' Formal Abstracts)
3. To relieve referees from having to verify correctness
4. Preservation of mathematics
5. Formalization can reveal missing mathematics
6. Tautologically, it's becoming "interesting" to the wider mathematics community

I hear mathematicians outside ITP communities say they think journals will eventually require formalized proofs. (Will it be the new LaTeX?)

ForTheL and Naproche

There are numerous efforts to make precise human-readable modeling languages.

For example, ForTheL (Formal Theory Language) [Paskevich 2007] gives a flexible language for first-order logic.

ForTheL: `every natural number m greater than 0 divides m!`

Meaning: $\forall m\,((m\,\varepsilon\,\mathrm{Number} \wedge m\,\varepsilon\,\mathrm{Natural} \wedge$
$\wedge\,\mathrm{isGreaterThan}(m, \mathrm{Zero})) \supset \mathrm{Divides}(m, \mathrm{Factorial}(m)))$

Naproche is a proof assistant using this. [Koepke, et al. 2009]

Naproche implements elaboration of ForTheL.

Example from early 2022

Consider the following theorem statement (written in Lean) and give an accurate translation of it into English.

```
theorem finset.all_card_le_bUnion_card_iff_exists_injective
  {ι : Type u} {α : Type v} [decidable_eq α] (t : ι → finset α) :
  (∀ (s : finset ι), s.card ≤ (s.bUnion t).card) ↔
    (∃ (f : ι → α), function.injective f ∧ ∀ x, f x ∈ t x)
```

This theorem states that for any two sets of elements, namely, a collection of sets indexed by a type `ι` and a single set `α`, with `α` having decidable equality, the following two statements are equivalent:

- For any subset `s` of `ι`, the cardinality of `s` is less than or equal to the cardinality of the union of all the sets indexed by `s`.
- There exists a function `f` from `ι` to `α` such that `f` is injective (i.e., distinct elements in `ι` map to distinct elements in `α`) and for all `x` in `ι`, `f(x)` belongs to the set `t(x)` indexed by `x`.

ChatGPT doesn't introduce the family of sets *t*. What recourse do we have?

More informalization

Our driving example

From Bourbaki, *Topologie Générale*:

Theorem. *Let X be a topological space, A a dense subset of X, $f : X \to Y$ a mapping of X into a regular space Y. If, for each $x \in X$, $f(y)$ tends to $f(x)$ when y tends to x while remaining in A then f is continuous.*

Proof. We have to show that f is continuous at each point $x \in X$. Let then V' be a closed neighbourhood of $f(x)$ in Y; then by hypothesis there is an open neighbourhood V of x in X such that $f(V \cap A) \subset V'$. Since V is a neighbourhood of each of its points, we have

$$f(z) = \lim_{y \to z, y \in V \cap A} f(y)$$

for each z in V, and from this it follows that $f(z) \in \overline{f(V \cap A)} \subset V'$, since V' is closed. The result now follows from the fact that the closed neighbourhoods of $f(x)$ form a fundamental system of neighbourhoods of $f(x)$ in Y. $\square$

```lean
theorem continuous_of_dense [TopologicalSpace X] [TopologicalSpace Y] [RegularSpace Y]
    {A : Set X} (hA : Dense A) (f : X → Y) (hf : ∀ x, ContinuousWithinAt f A x) : Continuous' f := by
  intro x
  suffices key : ∀ V' ∈ Nhd (f x), IsClosed V' → ∃ U ∈ Nhd x, f '' U ⊆ V'
  · intro V' V'_in
    rcases RegularSpace.closed_nhd_basis (f x) V' V'_in with ⟨W, W_in, W_closed, hW⟩
    rcases key W W_in W_closed with ⟨U, U_in, hU⟩
    exact ⟨U, U_in, hU.trans hW⟩
  intro V' V'_in V'_closed
  obtain ⟨V, V_in, V_op, hV⟩ : ∃ V ∈ Nhd x, IsOpen V ∧ f '' (V ∩ A) ⊆ V'
  · rcases (hf x).2 V' V'_in with ⟨U, U_in, hU⟩
    rcases exists_IsOpen_Nhd U_in with ⟨V, V_in, V_op, hVU⟩
    use V, V_in, V_op
    exact (image_subset f $ inter_subset_inter_left A hVU).trans hU
  use V, V_in
  rintro _ ⟨z, z_in, rfl⟩
  have limV : TendsToWithin f (V ∩ A) z (f z)
  · constructor
    · apply V_op.inter_closure (t := A)
      exact ⟨z_in, hA z⟩
    · intro W W_in
      rcases (hf z).2 W W_in with ⟨U, U_in, hU⟩
      use U, U_in
      exact (image_subset f $ inter_subset_inter_right U (inter_subset_right V A)).trans hU
  calc
    f z ∈ closure (f '' (V ∩ A)) := limV.mem_closure_image
      _ ⊆ closure V'             := closure_mono hV
      _ = V'                     := V'_closed.closure_eq
```

Example: the proof

This is expanded out to a comparable level of detail to the Lean code.

Lean proofs can take advantage of the elaborator being able to fill in details obvious to a computer, so it's not surprising if an English version might be wordier!

Proof. Let x be an element of X. One can see it suffices to prove that for all closed neighborhoods V' of $f(x)$, there exists a neighborhood U of x such that $f[U] \subseteq V'$, which we see by the following argument:

> Let V' be a neighborhood of $f(x)$. One can obtain a closed neighborhood W of $f(x)$ such that $W \subseteq V'$. One can obtain a neighborhood U of x such that $f[U] \subseteq W$. We will show that U is suitable by proving that U is a neighborhood of x and $f[U] \subseteq V'$. By assumption, U is a neighborhood of x. Using our assumption that $f[U] \subseteq W$ and our assumption that $W \subseteq V'$ proves $f[U] \subseteq V'$.

Let V' be a closed neighborhood of $f(x)$.

Claim: there exists an open neighborhood V of x such that $f[V \cap A] \subseteq V'$.

> One can obtain a neighborhood U of x such that $f[U \cap A] \subseteq V'$. One can obtain an open neighborhood V of x such that $V \subseteq U$. We will show that V is suitable by proving that V is a neighborhood of x, V is open and $f[V \cap A] \subseteq V'$. By assumption, V is a neighborhood of x. By assumption, V is open. Using our assumption that $V \subseteq U$ and our assumption that $f[U \cap A] \subseteq V'$ proves $f[V \cap A] \subseteq V'$.

Using this claim we obtain an open neighborhood V of x such that $f[V \cap A] \subseteq V'$. We will show that V is suitable by proving that V is a neighborhood of x and $f[V] \subseteq V'$. By assumption, V is a neighborhood of x. Let z be an element of V.

Claim: $f(z_1)$ tends to $f(z)$ as z_1 tends to z within $V \cap A$.

> By definition it suffices (1) to prove that $z \in \overline{V \cap A}$ and (2) to prove that for all neighborhoods V'' of $f(z)$, there exists a neighborhood U of z such that $f[U \cap (V \cap A)] \subseteq V''$.
>
> 1. One can see that $z \in \overline{V \cap A}$.
> 2. One can see that for all neighborhoods V'' of $f(z)$, there exists a neighborhood U of z such that $f[U \cap (V \cap A)] \subseteq V''$.

Consider the following:

$f(z) \in \overline{f[V \cap A]}$	Using our assumption that $f(z_1)$ tends to $f(z)$ as z_1 tends to z within $V \cap A$ proves this step.
$\subseteq \overline{V'}$	Using our assumption that $f[V \cap A] \subseteq V'$ proves this step.
$= V'$	Using our assumption that V' is closed proves this step.

This completes the proof. □

Trivial logic

Mathematicians do not manually curry/uncurry implications/conjunctions

```
theorem obv1 (P Q : Prop) (hp : P) (hq : Q) : P ∧ Q

theorem obv2 (P Q : Prop) (hpq : P ∧ Q) : P ∧ Q
```

Theorem (obv1). Let P and Q be propositions. If P and Q then P and Q.

Theorem (obv2). Let P and Q be propositions. Assume that P and Q. Then P and Q.

(The wording difference is due to a limitation in the current ontology.)

Expressions to LaTeX: LeanTeX

We have a `Lean.Expr` ➔ LaTeX pretty printer.

Basic “impedance mismatch”:

> Lean expressions are trees, but traditional notation is a 2D layout.
>
> Precedence levels are not sufficient to model 2D layout properly.

$${\frac{x+y}{2}}^{2}_{3} \quad \to \quad \frac{(x+y)^2}{2_3} \quad \text{or} \quad \left(\frac{x+y}{2}\right)^{2}_{3} \quad ?$$

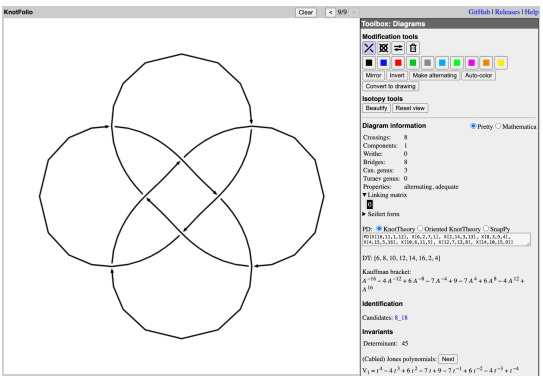
So far this has been enough:

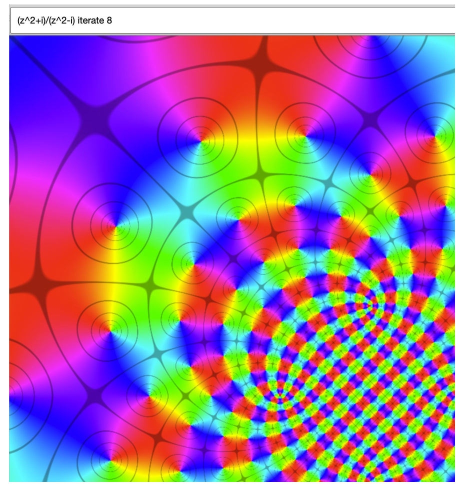
```
inductive LatexData where
  /-- anything that can support superscripts and subscripts without needing parentheses -/
  | Atom (latex : String) (bigness : Nat := 0) (sup? : Option LatexData) (sub? : Option LatexData)
  /-- anything that is a row of operators and operands -/
  | Row (latex : String) (lbp : BP) (rbp : BP) (bigness : Nat)
```

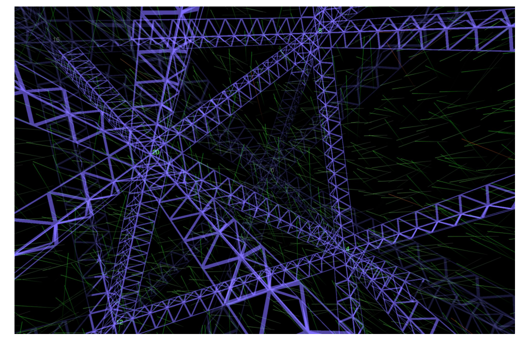
Some LaTeX pretty printers

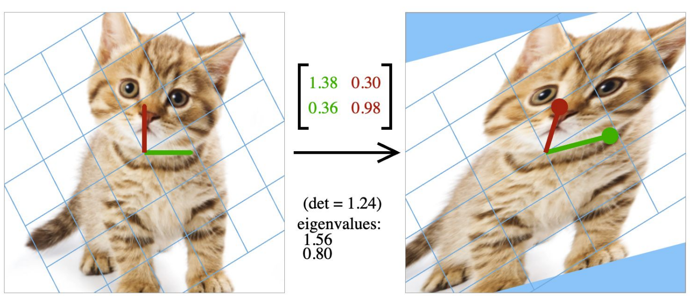
```
latex_pp_app_rules (const := Eq)
  | _, #[_, a, b] => do
    let a ← latexPP a
    let b ← latexPP b
    return a.protectRight 50 ++ LatexData.nonAssocOp " = " 50 ++ b.protectLeft 50

/-- Fancy division: use frac or tfrac if things aren't too small. -/
latex_pp_app_rules (const := HDiv.hDiv)
  | _, #[_, _, _, _, a, b] => do
    let frac ←
      match (← read).smallness with
        | 0 => pure LatexData.frac
        | 1 => pure LatexData.tfrac
        | _ => failure
    let a ← withExtraSmallness 1 <| latexPP a
    let b ← withExtraSmallness 1 <| latexPP b
    return frac a b

/-- Powers. -/
latex_pp_app_rules (const := HPow.hPow)
  | _, #[_, _, _, _, a, b] => do
    let a ← latexPP a
    let b ← withExtraSmallness 2 <| latexPP b
    return a.sup b
```

I am interested in creating software tools for mathematics, computer science, and education









pyquiz

This is a domain-specific language for authoring Canvas quizzes. It is designed for with an eye toward

1. making it easy to have a ready library of questions to create new quizzes and
2. making it easy to generate variants of questions (making use of question groups).

The way in which quiz questions are authored is somewhat inspired by WeBWorK.

Pyquiz has been used for a variety of courses, including

- the 2021-2023 summer sessions of Math 54 at UC Berkeley,
- the 2022 summer session of PHILOS W12A at UC Berkeley, and
- the 2022 Winter and Spring offerings of Math 110 and Math 116 at UC Santa Cruz.