From Lean to Natural Language and Back

Interfaces for Formal Proofs

Kyle Miller
University of California, Santa Cruz

2025 ICERM Autoformalization for the Working Mathematician

Overview

- Musings on Mathematics and Proof
- Using natural language as an interface
  - InformalLean
  - Verbose Lean
- The Future

What is mathematics anyway?

We ought to understand what we're talking about if we want to talk about interfaces between things.

What is mathematics anyway?

A pure-math-biased answer:

Mathematics is what mathematicians do,
and mathematicians study structures in mathematics.

Germ theory of mathematics:

- Mathematical ideas live in mathematician 'hosts'.
- Mathematics spreads through communities via contact (meetings, talks, etc.), provided the ideas are **interesting** and **skepticism-resistant**.
- Papers and textbooks contain the dormant 'spores' of ideas.
- *Is an idea alive if there's no one still aware of it?*

What is mathematics anyway?

A pedagogical/Greek answer:

Mathematics is what one learns in mathematics courses.

(*μαθήματα* (*mathēmata*) means “lessons”)

- *All* structured learning once was under the umbrella of mathematics.
- In any case, Euclid’s *Elements* was very influential.
  “Let’s do more of the way he does geometry!”

What is mathematics anyway?

A Dutch answer:

Mathematics is what one can know for sure.

(*wiskunde*, coined by Simon Stevins, 16th century)

- With this framing, all that's left are the things that leave no room for doubt.
- We can see the role of **proof** in this. Let's say

  A document *P* is a *proof* for mathematician *M* of theorem *T*
  if after *M* reads *P* then *M* is sure that *T* is true.

Plus, the Dutch invented Automath (De Bruijn, 1967), the primordial proof assistant.
(Introduced the Curry–Howard–de Bruijn correspondence, dependent types, ... (!))

Uses of proofs

Proofs serve different purposes to different audiences:

- **Mathematicians**
  - Propagation of new ideas and structural observations.
  - Support for claims of solving open problems.
- **Students**
  - Models from which to learn the language and practices of mathematics, for cultural integration.
- **Users** (might be other mathematicians)
  - To be able to “black box” a result, given that there’s a proof that’s (presumably) been checked.
- **Engineers**
  - A theorem is a specification, a proof is good so long as it exists and is maintainable.

Formal proofs

It’s possible to write proofs where one can know for sure that a reasonable person who reads it will know for sure that the theorem is true. (Thanks logicians!)

Strict definition of *reasonable*: one who accepts a given set of principles of logical reasoning.

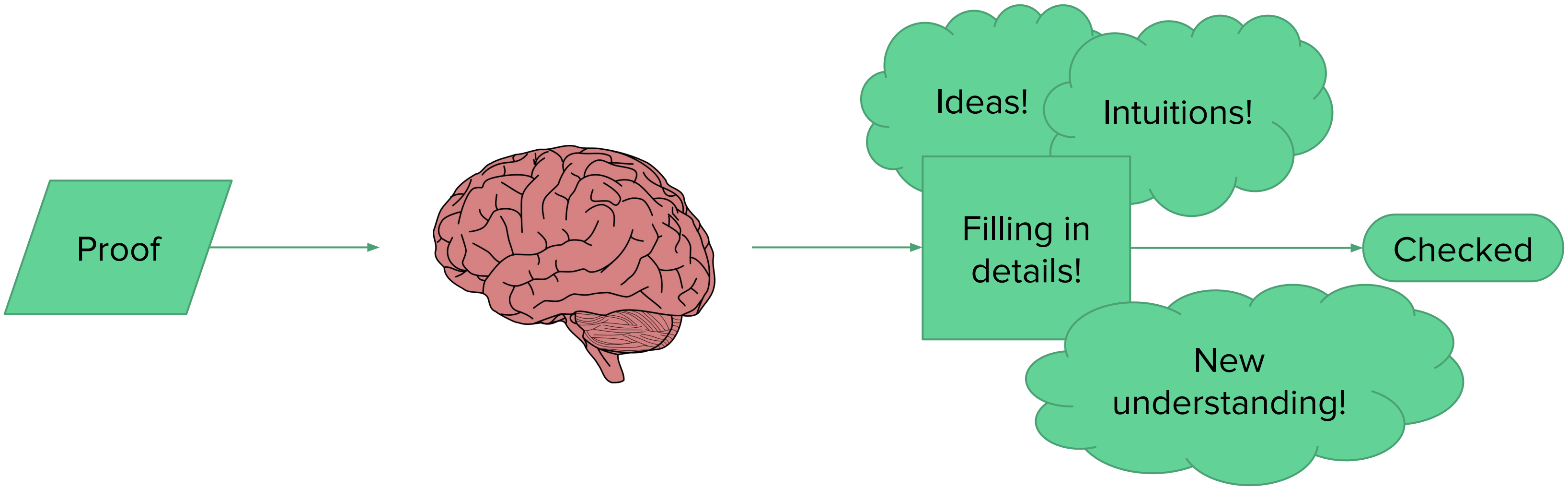
These proofs are checkable by their formal structure alone.

Large-scale experiments give strong evidence of a “**formalization hypothesis**”, that all the objects of mathematical knowledge can be rendered formally.

Tension:

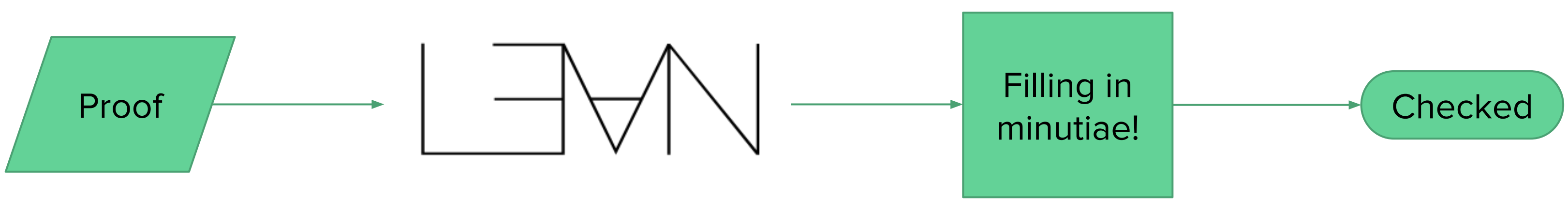
Pre-mechanization of proofs

Classic “compilation” path for checking



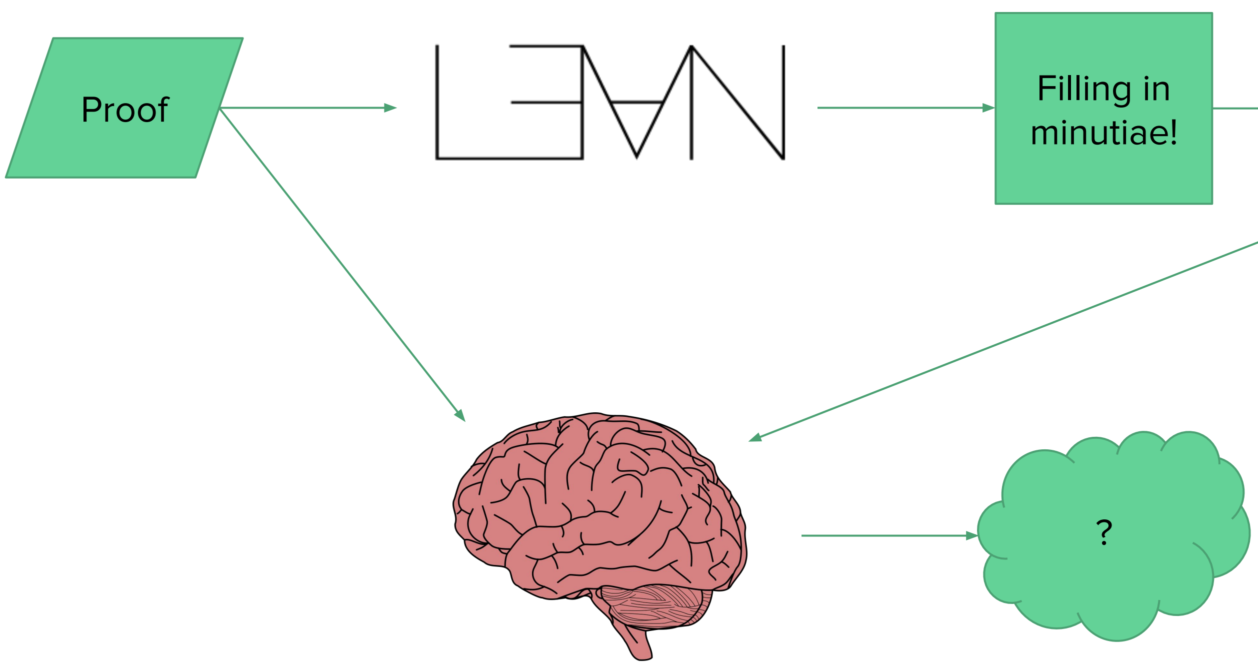
Post-mechanization of proofs

Modern “compilation” path for checking



Post-mechanization of proofs

Modern “compilation” path for checking



Humans & Machines: some questions

- Can we expect to write proofs that satisfy both humans and machines?
  - Machines can tirelessly check minute details, but don't currently understand the big picture.
  - Humans can see the big picture and can be convinced that there must be a way to fill in the details, but too many details can be overwhelming.
- What about satisfying different human audiences?
  - We already explain things differently for a subfield, the wider community, and the classroom.
  - Is there a future where one source text can be used to generate audience-appropriate treatments of a proof? Or of a whole theory? Correctly?
- To what extent can machine-checked proofs be used in teaching?
  - Mathematicians don't run formal proof checkers in their heads, so teaching how to prove things in Lean likely isn't teaching mathematical thought itself.
  - However, it surely can play a role in Tao's pre-rigorous / rigorous / post-rigorous progression.

Exploration: natural language as an interface

Mathematics is traditionally communicated in a (dialect of a) natural language.

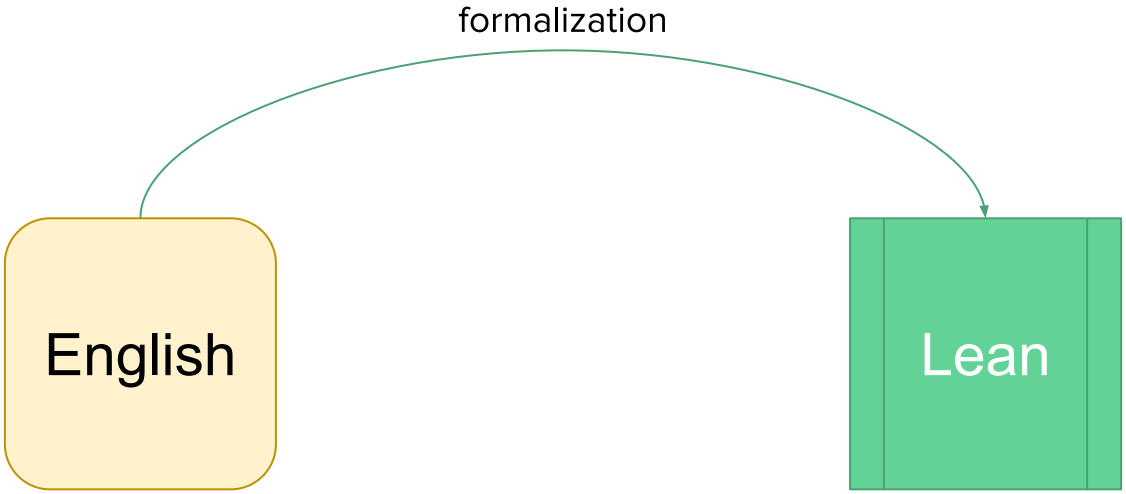
An old idea: natural language programming (e.g. COBOL).
Using controlled natural language to meet the user where they are at.
This is not what I am talking about today exactly.

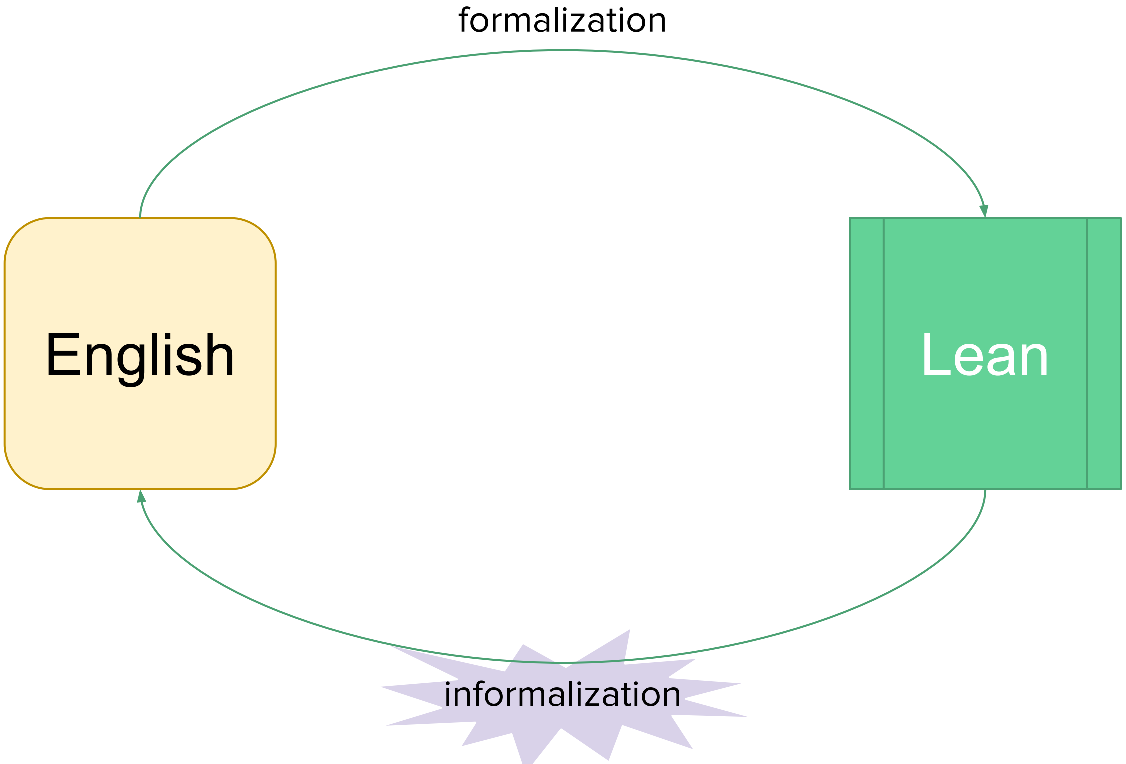
Closer example: Inform 7, for making parser-based interactive fiction.
Graham Nelson explored “what if writing the games was like playing them?”

For math:
What are ways to engage with formal proofs using the common language?

Natural Language for Reading Proofs

Patrick Massot
Université Paris-Saclay


formalization
English
Lean


formalization
English
Lean
informalization

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.

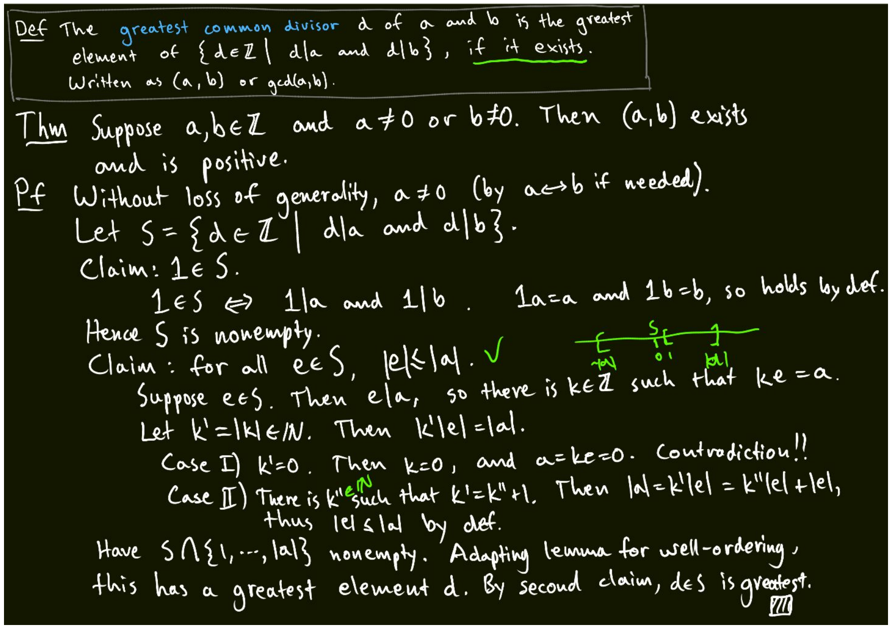
Assume that x is a limit point of E^c. ⊕We see $x \in E^c$. Using this and our assumption that $x \notin E^c$ we are done.

⊕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 $\{d \in \mathbb{Z} \mid d|a \text{ and } d|b\}$, 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 $S = \{d \in \mathbb{Z} \mid d|a \text{ and } d|b\}$.
Claim: $1 \in S$.
$1 \in S \iff 1|a$ and $1|b$. $1a = a$ 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 $e|a$, 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. ▨

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.

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, they are usually not human readable on their 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.

□

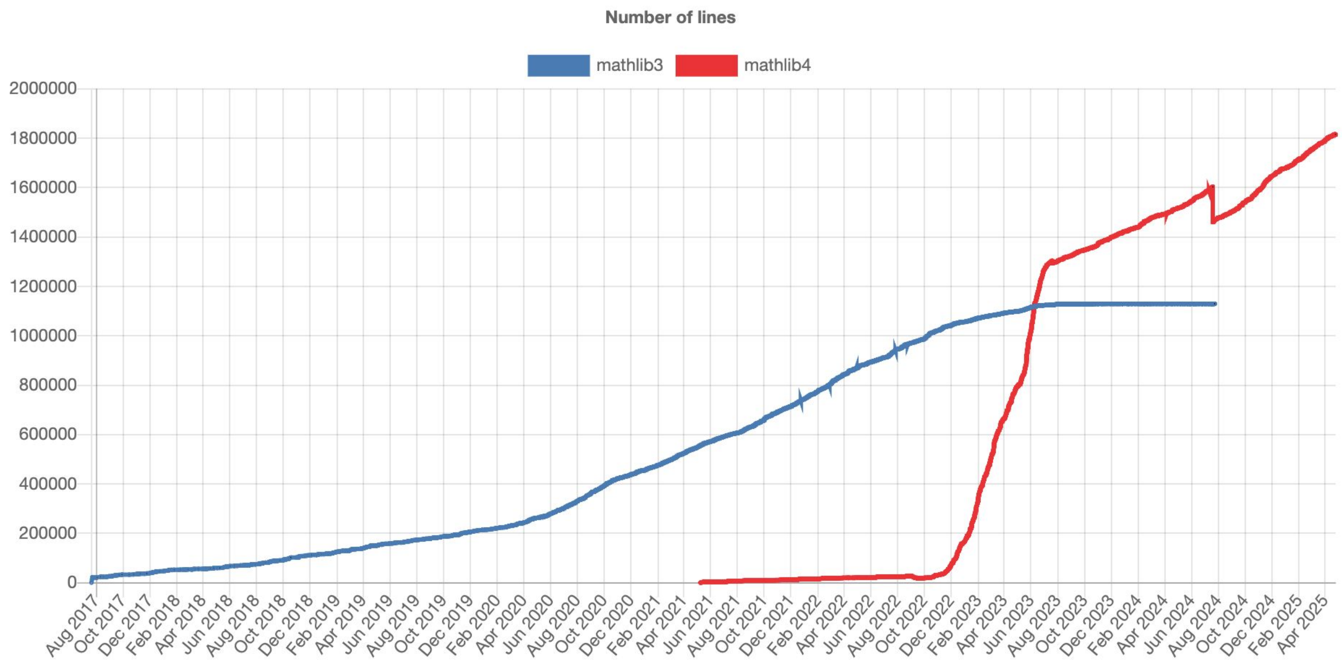
Such a document could answer any question you have about the proof.

Authoring these by hand is at least as hard as writing a formal proof!

Mathlib is a large library of formalized mathematics

103k definitions
207k theorems

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



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.

As we know, Large Language Models (LLMs) are being applied to this problem.

Training data is scarce

Observation: formalization does not lead to training pairs.

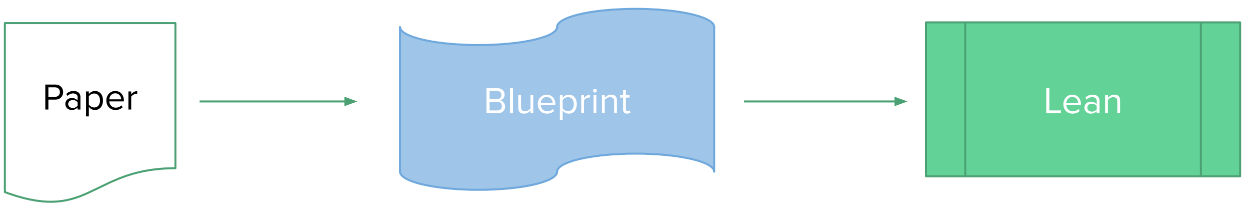
Papers are too informal.

They have to be transformed and expanded.

The formalization might have no obvious connection to structure of the source text.

Blueprints

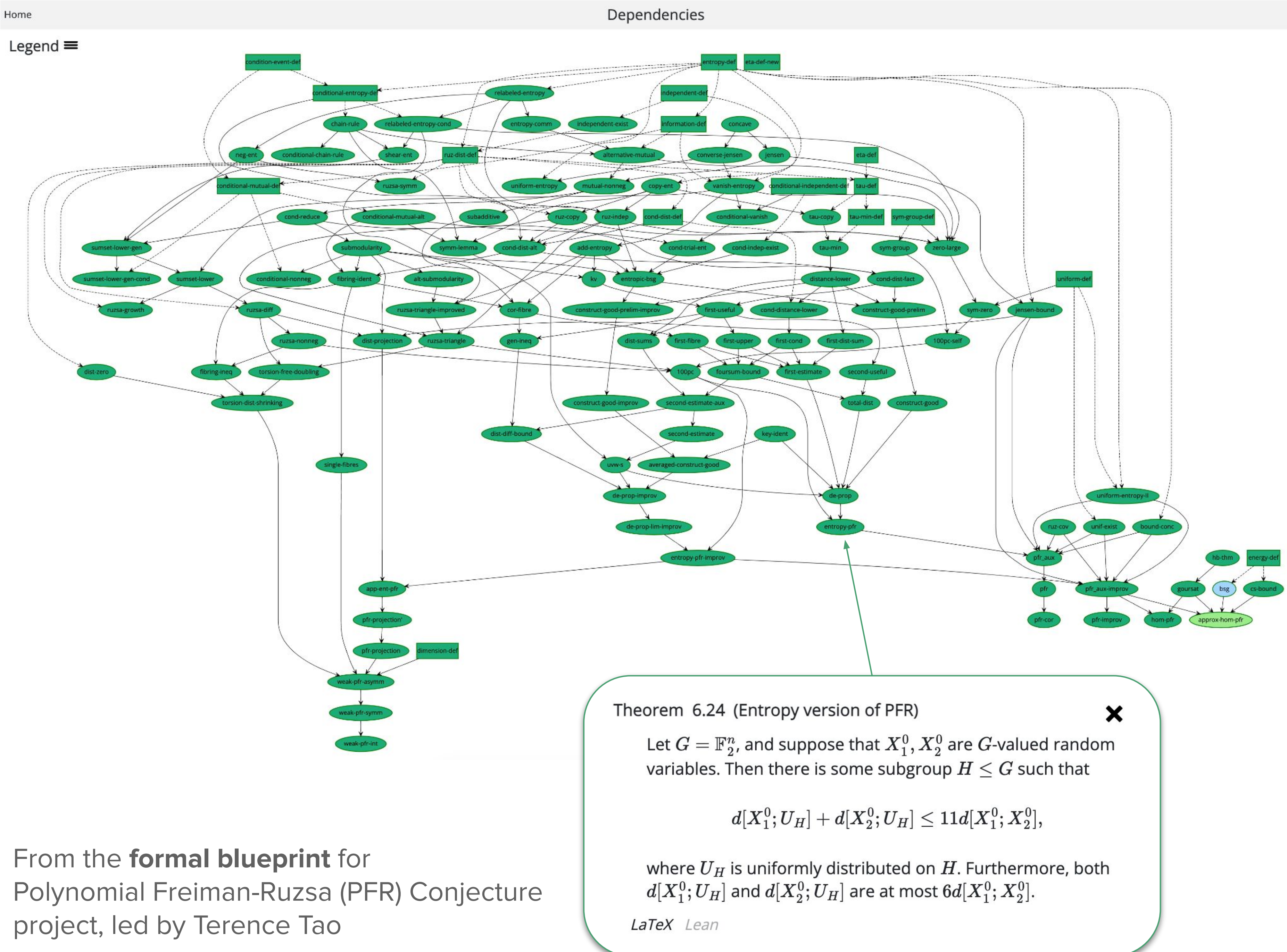
A *blueprint* is an design document for a formalization project.



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.

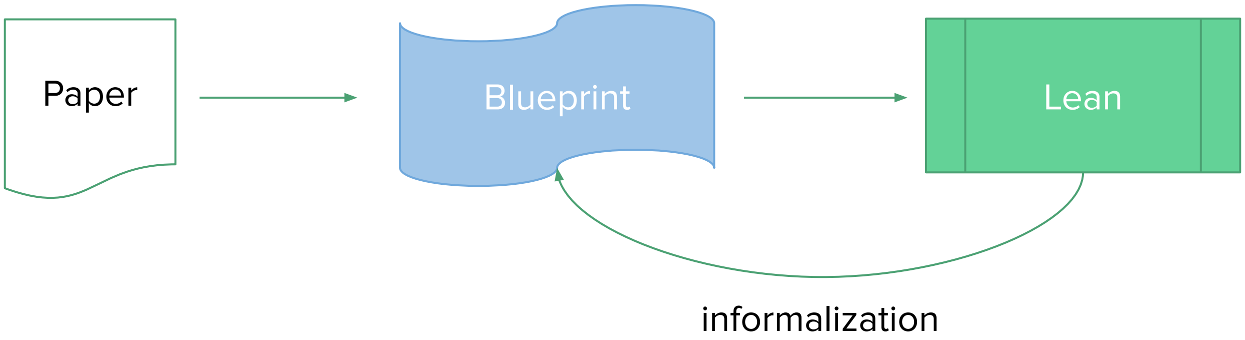
Formalizing from the blueprint takes only general knowledge.



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 blueprint.



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

- Blueprint & Lean
- Paper & blueprint

Autoformalizers could train for each translation task independently.

How to informalize

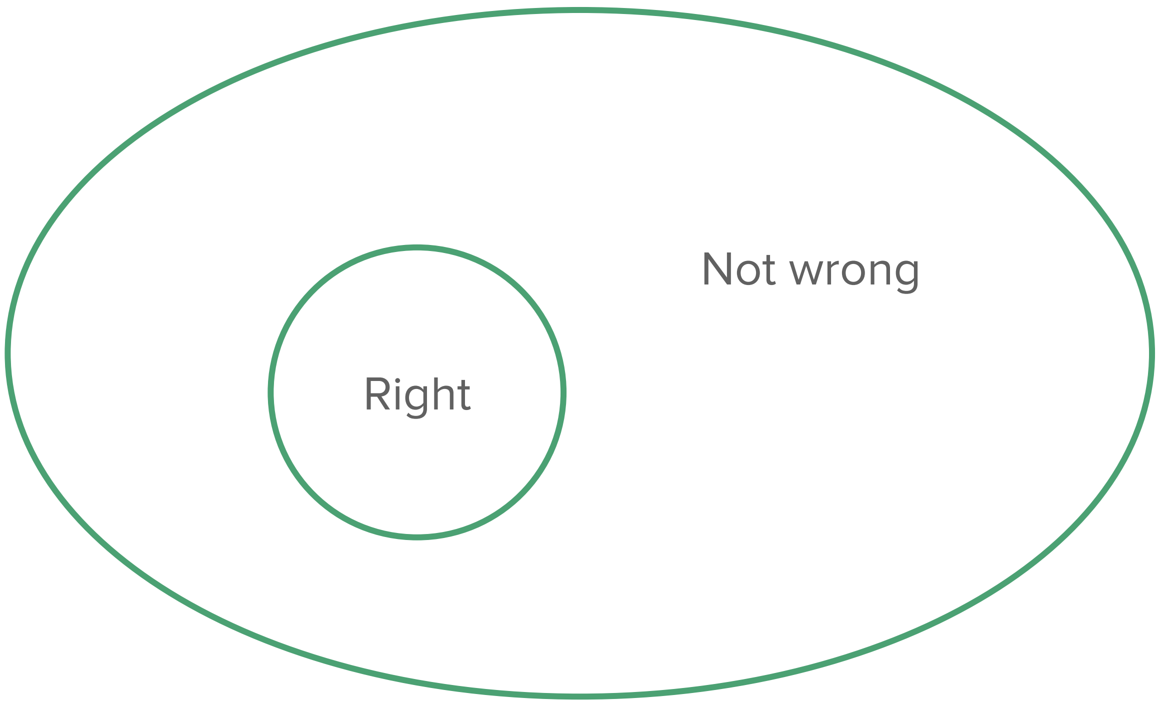
Patrick Massot and I developed a prototype auto-informalizer.

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, decreases probabilities, doesn’t eliminate.
- LLMs struggle with logical reasoning. (It’s gotten significantly better!)
    - “Sorry, you are correct that 8 is not odd. However, since 8 is odd, …”

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 4o

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 ≤ #(s.biUnion t)) ↔
   ∃ f : ι → α, Function.Injective f ∧ ∀ x, f x ∈ t x
```

Here's a **precise and accurate translation** of that Lean theorem into mathematical English:

Theorem.
Let $\{t(x) \subseteq \alpha \mid x \in \iota\}$ be a family of finite subsets of a type α, indexed by a type ι.
Then the following are equivalent:

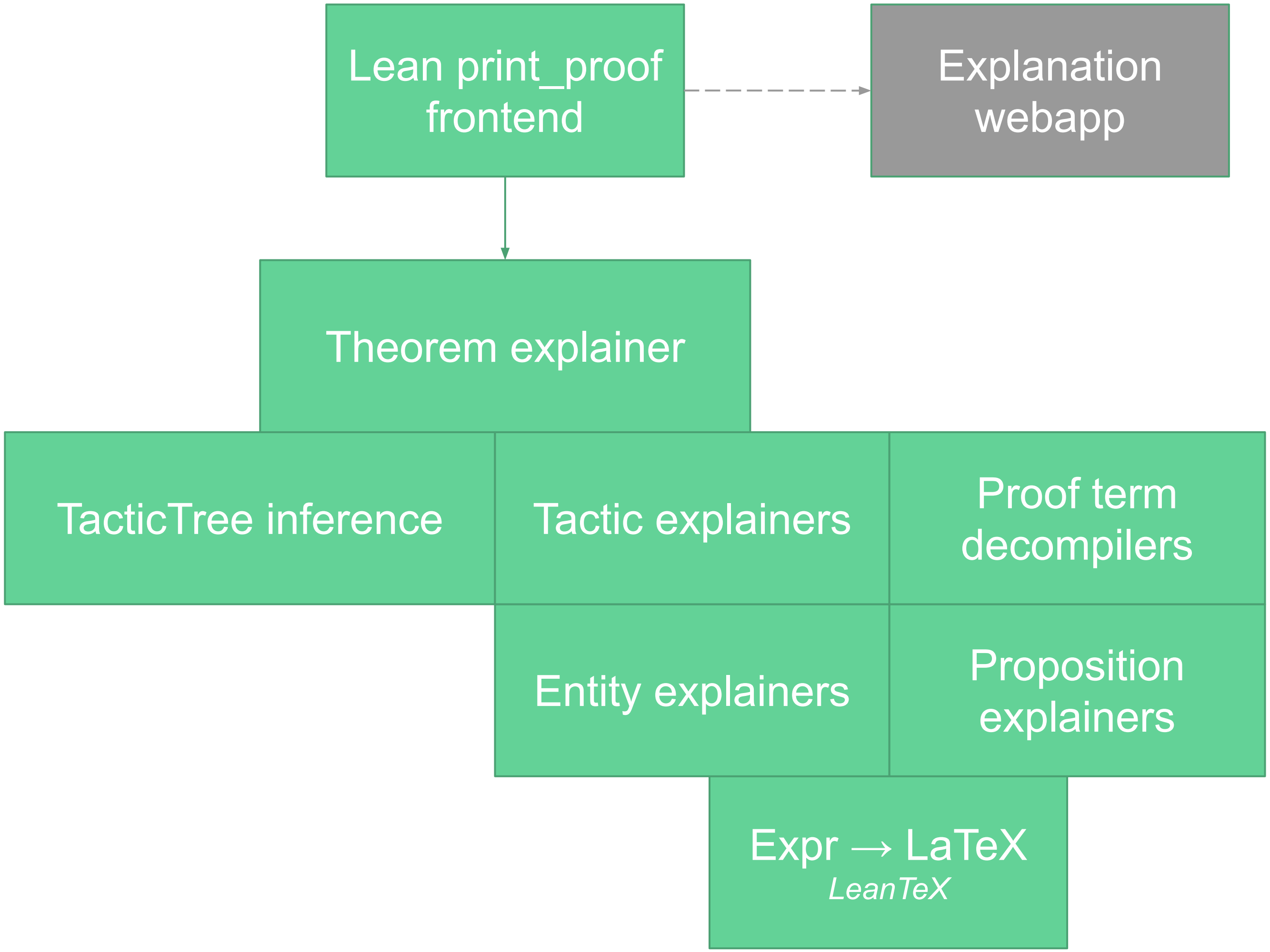
- For every finite subset $s \subseteq \iota$, the cardinality of s is less than or equal to the cardinality of the union of the sets $t(x)$ over all $x \in s$; that is,

$$\forall s \subseteq_{\text{fin}} \iota, \quad \#s \leq \# \bigcup_{x \in s} t(x),$$

- There exists an injective function $f : \iota \to \alpha$ such that for every $x \in \iota$, $f(x) \in t(x)$.

Demo

General architecture



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

```
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.

Lean and English

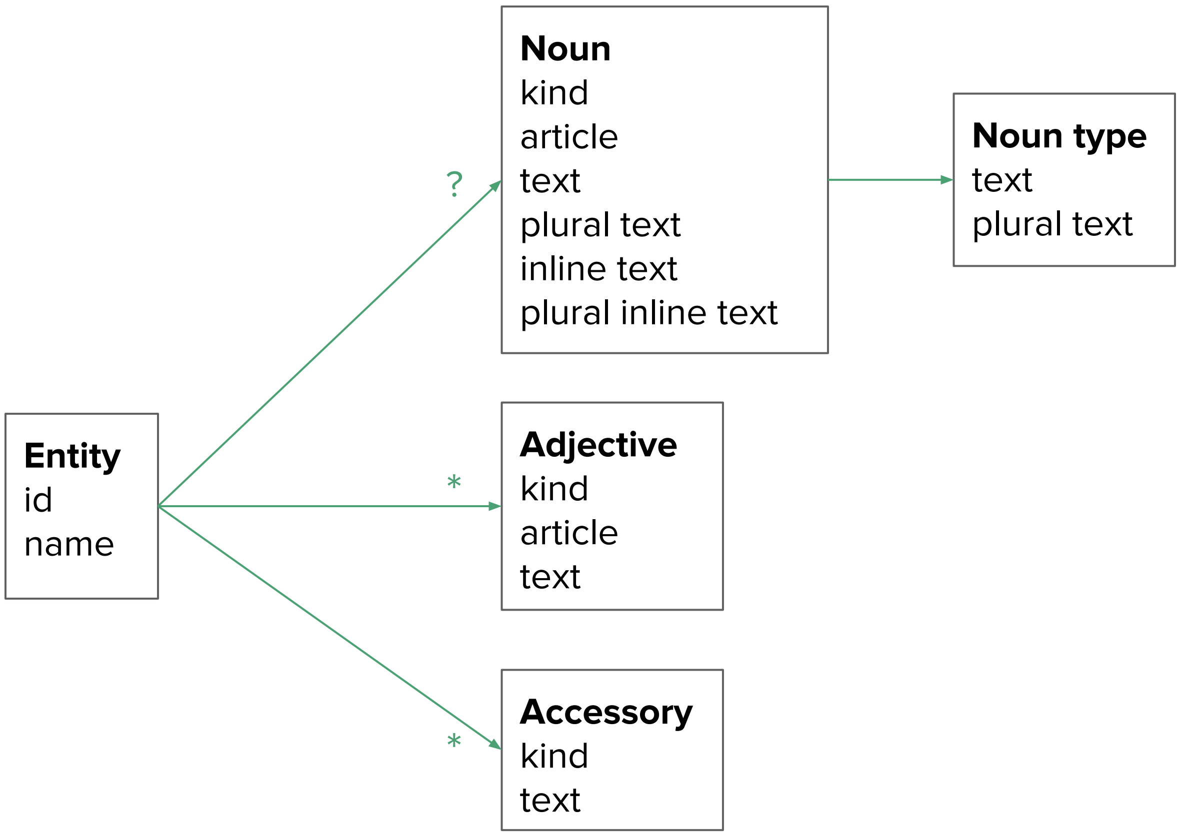
Lean 4 has *expressions*, *declarations*, *metavariables*, *local contexts*, *tactic states*, *tactics*, and so on. (Lean 4 is written in Lean 4, and this is all accessible to us!)

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



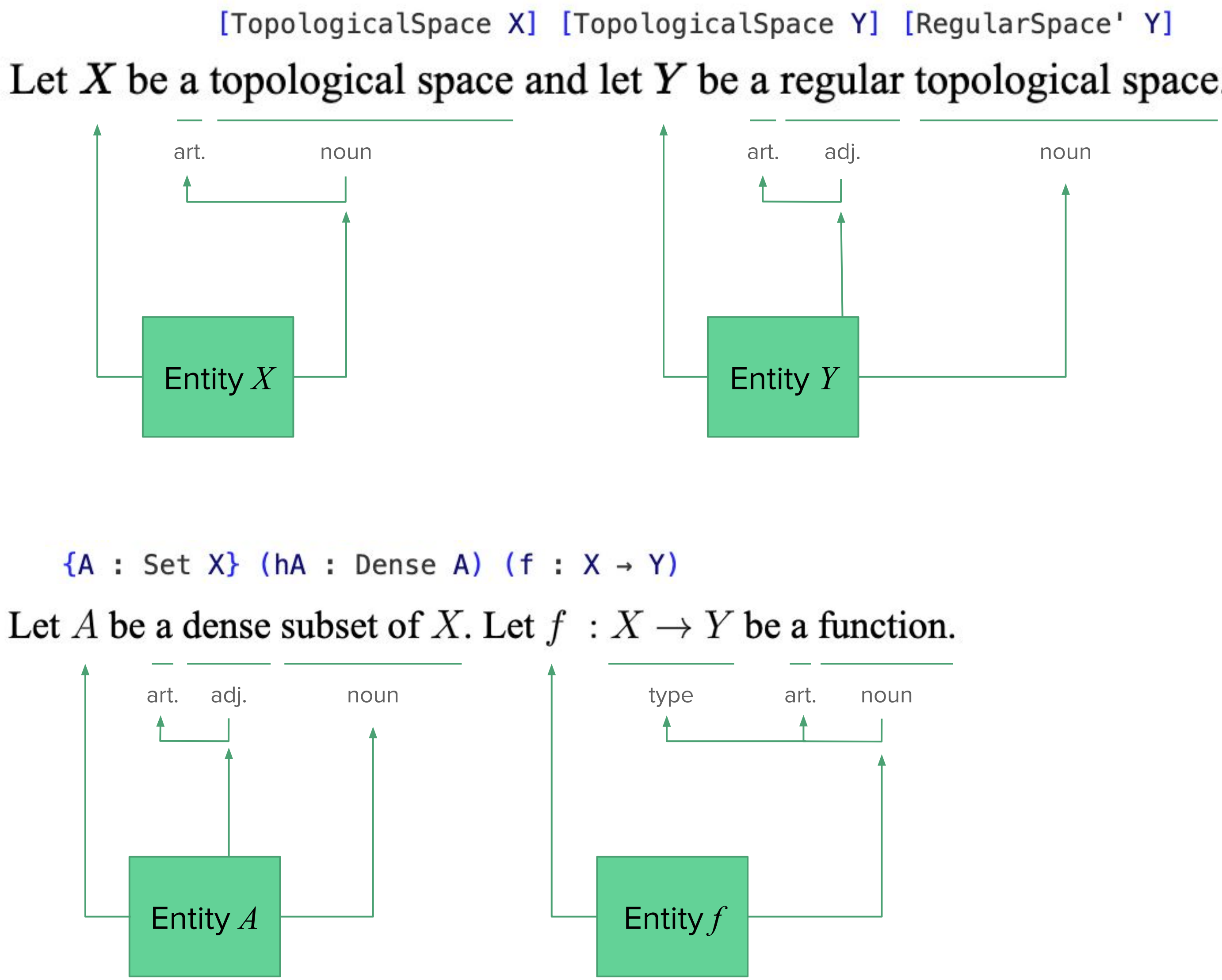
```lean
structure Entity where
  fvarid : FVarId
  entityName : String
  noun : Option Noun
  provides : Array FVarId := #[fvarid]
  adjectives : Array Adjective := #[]
  accessories : Array Accessory := #[]
```

```lean
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

```
@[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

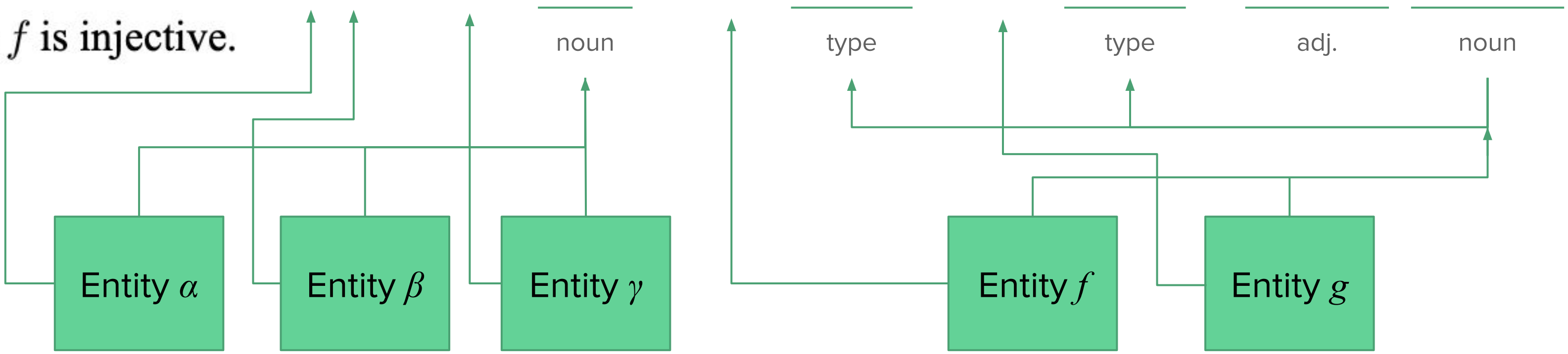
If consecutive entities have compatible data, we can merge their introductions into a single sentence.

```
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

Small wrinkle: 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 get to 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

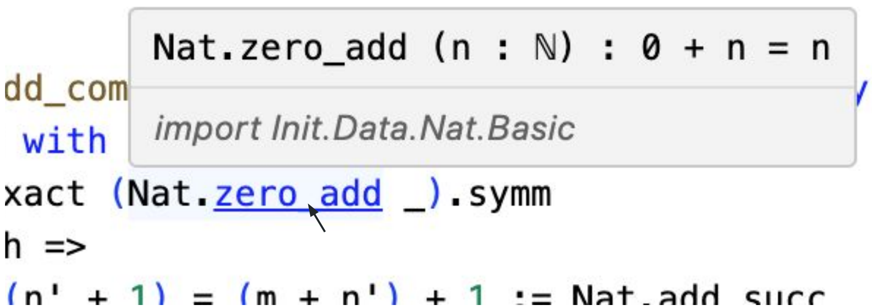
The next big part: representing proofs

Main elements:

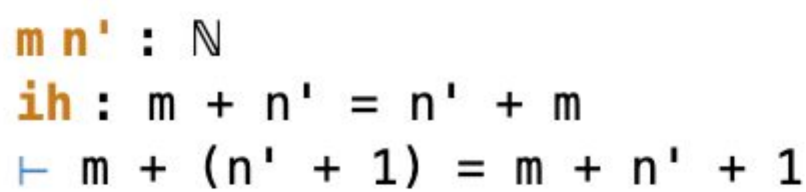
- Deducing tactic proof structure
- Tactic describers
- Proof term decompiler

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 InfoTree data

Hovering on

```
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.

Explanations

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

- Block indentation, paragraph breaks, tooltips
- Expansion widgets and highlights that cross paragraph boundaries
- Goal states
- Multiline equations

There is a JavaScript renderer for `Explanation`s.

This is not specialized to LeanInformal.

Proof term explanations

For many tactics, we decompile its generated proof term into an equivalent sequence of simpler tactics, then compile *that* into English.

> **Reason 1.** We want to avoid recapitulating tactic implementations.
>
> **Reason 2.** Lean files are written primarily to be understood by Lean.
>
> **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 others.

```
exact hf x y key
```

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

LeanTeX

https://github.com/kmill/LeanTeX-mathlib

```
example (a b c : ℝ)
    (ha : 0 < a) (hb : 0 < b) (hc : 0 < c) (h : a*b*c = 1) :
    3 ≤ Real.sqrt ((a + b) / (a + 1)) + Real.sqrt ((b + c) / (b + 1)) + Real.sqrt ((c + a) / (c + 1)) := by
  texify
  sorry
```

▼HTML Display

$a, b, c : \mathbb{R}$

$ha : 0 < a$

$hb : 0 < b$

$hc : 0 < c$

$h : a \cdot b \cdot c = 1$

$$\vdash 3 \leq \sqrt{\frac{a+b}{a+1}} + \sqrt{\frac{b+c}{b+1}} + \sqrt{\frac{c+a}{c+1}}$$

```
example (n : ℕ) : ∑ i ∈ Finset.range n, i = n * (n - 1) / 2 := by
  texify
  sorry
```

▼HTML Display

$n : \mathbb{N}$

$$\vdash \sum_{i \in [0,n)} i = \frac{n \cdot (n-1)}{2}$$

InformalLean: Ongoing and future work

- Nailing down a spec for the various kinds of describers.
    - Once that’s done, we’ll release InformalLean.
    - We’ll need help getting mathlib coverage!
- More sophisticated interactions.
    - Communicating information about typeclass instances.
    - Answering queries (“where is this hypothesis used”)
- Using InformalLean in the classroom.
    - Do interactive proofs improve learning outcomes?
    - Or is the struggle to read a book like Rudin essential to the process?
- Generating that synthetic AI training data!

Natural Language for Writing Proofs

Verbose Lean [Massot 2024]

```
Example "The squeeze theorem."
  Given: (u v w : ℕ → ℝ) (l : ℝ)
  Assume: (hu : u converges to l) (hw : w converges to l)
    (h : ∀ n, u n ≤ v n)
    (h' : ∀ n, v n ≤ w n)
  Conclusion: v converges to l
Proof:
  Let's prove that ∀ ε > 0, ∃ N, ∀ n ≥ N, |v n - l| ≤ ε
  Fix ε > 0
  Since u converges to l and ε > 0 we get N such that hN : ∀ n ≥ N, |u n - l| ≤ ε
  Since w converges to l and ε > 0 we get N' such that hN' : ∀ n ≥ N', |w n - l| ≤ ε
  Let's prove that max N N' works : ∀ n ≥ max N N', |v n - l| ≤ ε
  Fix n ≥ max N N'
  Since ∀ n ≥ N,  |u n - l| ≤ ε and n ≥ N  we get hNl  : |u n - l| ≤ ε
  Since ∀ n ≥ N', |w n - l| ≤ ε and n ≥ N' we get hN'l : |w n - l| ≤ ε
  Let's prove that |v n - l| ≤ ε
  Let's first prove that -ε ≤ v n - l
  Calc -ε ≤ u n - l since |u n - l| ≤ ε
       _  ≤ v n - l since u n ≤ v n
  Let's now prove that v n - l ≤ ε
  Calc v n - l ≤ w n - l  since v n ≤ w n
       _       ≤ ε        since |w n - l| ≤ ε
QED
```

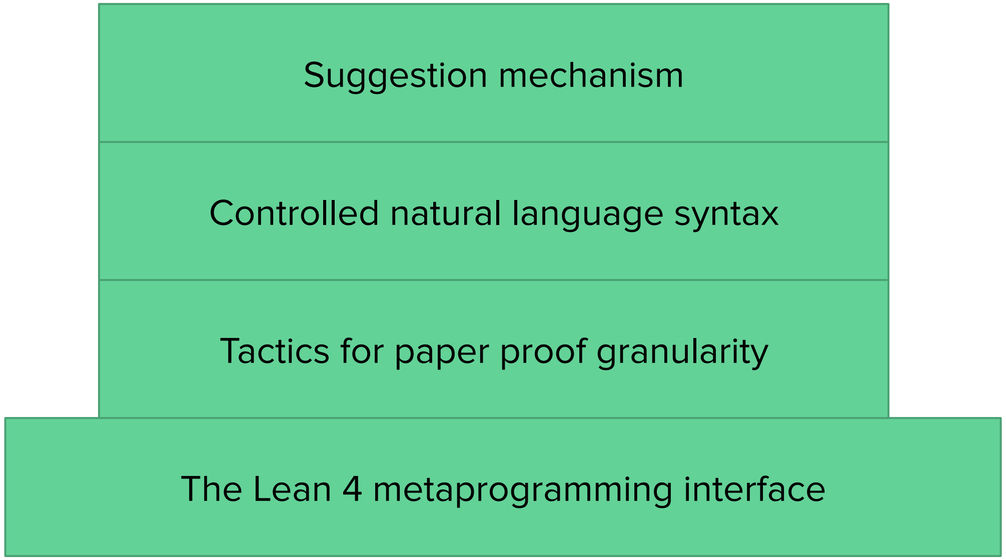
Motivation

- It's for learning to read and write traditional proofs.
  - Meant to be an aid while developing a paper proof.
  - Students can transfer their finished proofs to paper with minor editing.
- It is *not* meant to be a controlled natural language interface for Lean.
  - Learning to formalize is limited by formalization knowledge and skill, not surface syntax.
  - Controlled natural language has a worse "guess the grammar" problem.
- Design goals:
  - Giving clear proof states (essential for learning how to write proofs, but rarely provided!)
  - Training students how to apply the obvious goal-motivated proof steps.

    Routine steps ⇄ Risky steps

  - Training the difference between free and bound variables.
  - Being clear on stating / proving / using
    - Professionals blur the distinction but "know what they're doing" so it is "ok".
    - That is not a good model for students learning to be rigorous.
  - Localizable to students' native language. (Comes with English and French syntax.)

Rough architecture



Demo

Other systems

Waterproof for Coq. Very similar, with similar pedagogical goals.

```
Goal 2 is the infimum of [2, 5).
Proof.
We need to show that (2 is a _lower bound_ for [2, 5)
  ∧ (∀ l ∈ ℝ, l is a _lower bound_ for [2, 5) ⇒ l ≤ 2)).
We show both statements.
  - We need to show that (2 is a lower bound for [2, 5)).
    We need to show that (∀ c ∈ [2, 5), 2 ≤ c).
    Take c ∈ [2, 5).
    We conclude that (2 ≤ c).
  - We need to show that
      (∀ l ∈ ℝ, l is a lower bound for [2, 5) ⇒ l ≤ 2).
    Take l ∈ ℝ. Assume that (l is a lower bound for [2, 5)).
    We conclude that (l ≤ 2).
Qed.
```

Other systems

Naproche. Aims to be a natural language proof language.

Uses Formal Theory Language (ForTheL), translates its custom logic to Isabelle, and uses a "mile marker" declarative approach to proving, relying on a high-powered tactic to fill in the arguments.

```
Proposition. q^2 = p for no positive rational number q.
Proof by contradiction.
Assume the contrary.
Take a positive rational number q such that p = q^2.
Take coprime m,n such that m * q = n. Then p * m^2 = n^2.
Therefore p divides n. Take a natural number k such that n = k * p.
Then p * m^2 = p * (k * n).
Therefore m * m is equal to p * k^2.
Hence p divides m. Contradiction.
QED.
```

Other systems

Mizar. Uses a wordy/friendly grammar. Has the ability to output to natural language, used by the Journal of Formalized Mathematics.

```
i+k = j+k implies i=j;
proof
  defpred P[natural number] means
          i+$1 = j+$1 implies i=j;
  A1: P[0]
  proof
   assume B0: i+0 = j+0;
   B1: i+0 = i by INDUCT:3;
   B2: j+0 = j by INDUCT:3;
   hence thesis by B0,B1,B2;
  end;
  A2: for k st P[k] holds P[succ k]
  proof
    let l such that C1: P[l];
    assume C2: i+succ l=j+succ l;
    then C3: succ(i+l) = j+succ l by C2,INDUCT:4
    .= succ(j+l) by INDUCT:4;
    hence thesis by C1,INDUCT:2;
  end;
  for k holds P[k] from INDUCT:sch 1(A1,A2);
  hence thesis;
end;
```

The Future

The Future

What are other directions?

- What kinds of mathematical knowledge aren't being captured by formalization. Can they be?
- Use LLMs to drive stylistic choices to make high quality and accurate informalizations?
- Would a formally verified informalization tool have any applications?
- For natural language translations for undergraduate audiences, automatic specialization to appropriate settings? (e.g. "modules" -> "vector spaces")
- With classical NLP, we can feel responsible for the translations, even ones we have not reviewed personally. When can ML translations be trustworthy enough to not need human review? Maybe this is audience-dependent.
- Less stringent proof checkers for less formal proofs? ("Interactive Plausibility Assistants"?)