Informalizing formalized mathematics using the Lean theorem prover

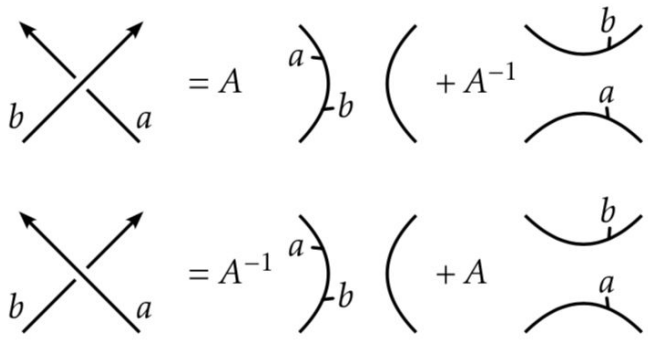
Kyle Miller, joint with Patrick Massot
UC Santa Cruz & Université Paris-Saclay

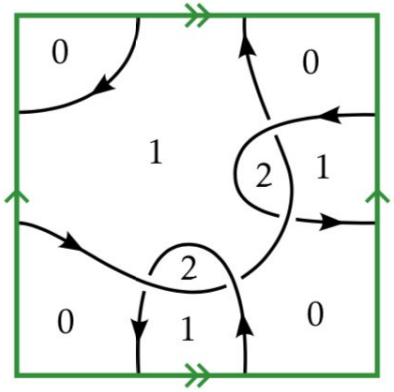
Workshop on Libraries of Formal Proofs and Natural Mathematical Language
8 September, 2023

Patrick Massot

Who am I?

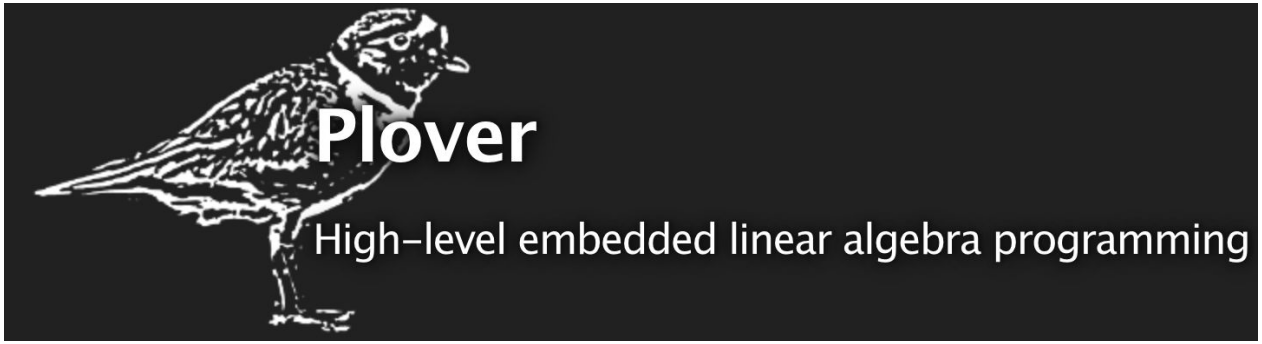
- Low-dimensional topologist





Who am I?

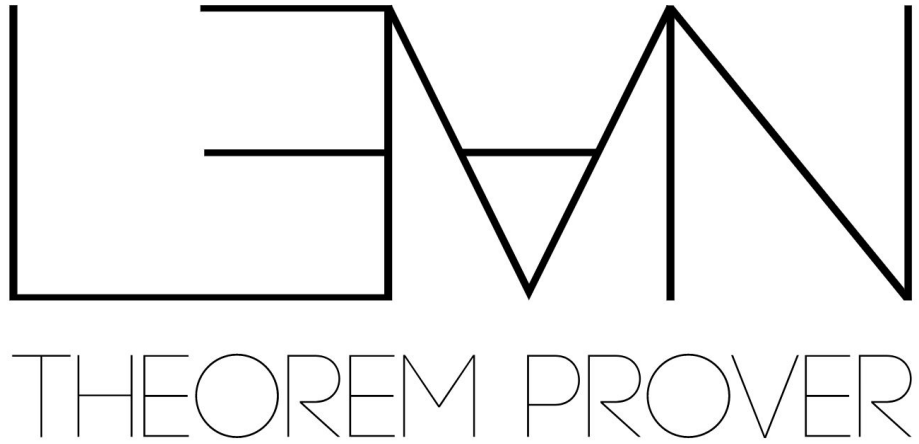
- Low-dimensional topologist
- PL enthusiast



(2015, Swift Navigation)

Who am I?

- Low-dimensional topologist
- PL enthusiast
- Contributor and maintainer for `mathlib`, the Lean mathematical library



Overview

1. Formalization, Lean, & Mathlib
2. Informalization?
3. Developing an informalizer

What is formalization?

Formalized mathematics, briefly

A *formal system* consists of some collection of rules that one may apply to strings of symbols to derive other strings.

In formal logic systems, the strings represent logical statements. For example, first-order logic and type theory.

Whatever our personal philosophies about the nature of mathematics may be, a common belief is that every mathematical truth can be represented formally.

(At least in theory!)

In practice, fully written formal proofs can be extremely long and non-illuminating.

Mathematicians often write a *recipe* for a proof instead.

("by a straightforward calculation..." "by adapting Theorem 2.2" etc.)

Computer formalization, briefly

Computers are great at processing long and non-illuminating symbolic reasoning!

A system design:

- A *kernel* is a small program to verify completed formal proofs.
- *Tactics* are composable programs to generate pieces of formal proofs.
- An *elaborator* is a program that takes a human-written input and fills in certain missing details and executes invoked tactics.

Then, one can use “recipe-like” proofs to complete a formalization effort.

Often there is a code editor with special support, giving an *interactive* experience.

To name a few systems: Agda, Coq, HOL Light, Isabelle, Lean, Mizar, (Metamath)

What is Lean?

The Lean theorem prover

- Lean is one of the newest interactive theorem provers.
  - Lean 1 was released in 2013, Lean 3 in 2017, Lean 4 in 2021
  - Agda is 1999, Coq is 1989, Isabelle is 1986, Mizar is 1973
- From Microsoft Research (Lean 4: Leonardo de Moura, Sebastian Ullrich, et al.)
- Uses a dependent type system based on the Calculus of Constructions
  - That is, function arguments each have types that can depend on the preceding arguments
  - Every mathematical proposition can be encoded as a type;
    Proofs are “programs”! (the Curry–Howard correspondence)
  - The encoding is *sound*: it is equiconsistent with ZFC + some inaccessible large cardinals
  - Similar to Coq or Agda, but with features like proof irrelevance and quotient types
- Perhaps by accident, mathematicians are a large fraction of Lean users

Lean 4

- First public release was January 2021
- It is a full programming language that is also a theorem prover
  - Much of Lean 4 is written in Lean 4
- 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, the tactic framework of proof-writing programs
- Can extend Lean 4 from within Lean 4
  - Syntax, macros, elaboration rules, etc.
  - One language for everything!

Examples

```
def TendsToInfty (f : ℕ → ℕ) : Prop :=
  ∀ n, ∃ m, ∀ k ≥ m, n ≤ f k

theorem TendsToInfty_id : TendsToInfty (id : ℕ → ℕ) := by
  intro n
  use n
  intro k hk
  exact hk

theorem TendsToInfty_mul (hf : TendsToInfty f) (hg : TendsToInfty g) :
    TendsToInfty (λ x => f x * g x) := by
  intro n
  rcases hf n with ⟨nf, hf⟩
  rcases hg n with ⟨ng, hg⟩
  use max nf ng
  intro k hk
  have hf' : n ≤ f k := by exact hf k ((Nat.le_max_left _ _).trans hk)
  have hg' : n ≤ g k := by exact hg k ((Nat.le_max_right _ _).trans hk)
  calc n ≤ n * n := Nat.le_mul_self _
       _ ≤ f k * g k := Nat.mul_le_mul hf' hg'
```

Examples

```
theorem TendsToInfty_id' : TendsToInfty (id : ℕ → ℕ) :=
  fun n => ⟨n, fun _ hk => hk⟩

theorem TendsToInfty_comp' (hf : TendsToInfty f) (hg : TendsToInfty g) :
    TendsToInfty (f ∘ g) := by
  intro n
  rcases hf n with ⟨n', hf⟩
  rcases hg n' with ⟨n'', hg⟩
  exact ⟨n'', fun k hk => hf _ (hg _ hk)⟩
```

What is mathlib?

The Lean mathematical library (mathlib)

Mathlib is a community-driven project to formalize all of mathematics as a single cohesive library.

Has over a million lines of Lean code produced by over a hundred contributors.

Like a digital Bourbaki, but undergoes constant revision and refactoring.

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

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 in Lean? Why not $THEOREM_PROVER?)

Experts in ITPs discuss defects in Lean's underlying theory
(Non-transitivity of definitional equality! Failure of subject reduction! No Church-Rosser property!)

I am not an expert. I'm generally ignorant about the effect of these defects.

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)
- The existence of a sphere eversion (Massot, Nash, and Van Doorn)
- Scholze's main theorem of liquid modules (Commelin)

Why formalize mathematics?

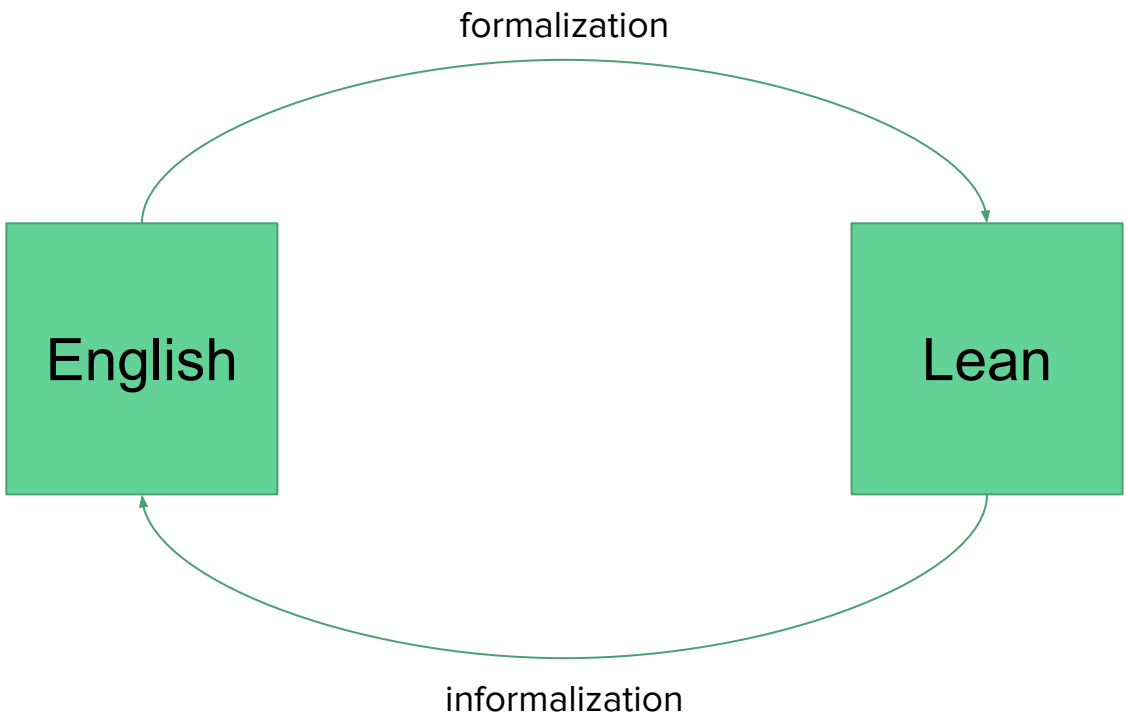
Mathematicians for now seem not to be so interested in computer verification of proofs – while mistakes do slip into most papers, they're often believed to be "minor" since the *idea* is generally "right."

So why go through all this effort? Some reasons:

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. 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?)

Another reason: informalization?



Why informalize?

1. We can try to make formalized math more accessible to the wider community.

Which of the following takes less specialized training to read?

- ```
  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 : 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, $f(x) \in t_x$.
  ```

Why informalize?

2. We can create documents that answer basic questions about a proof.

What if we had an “IDE extension” for English proofs?

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.

Proof context: *E* is a subset of $\mathbf{R}^n$.
The complement of E is closed.
x is an element of *E*.
Goal: *x* is in the interior of *E*.

Why informalize?

3. We can create documents with multiple levels of detail available for readers.

We see that $(n + 1)^2$ is positive.

… *Why?*

Since $n + 1$ is positive, $(n + 1)^2$ is positive.

… *Why?*

Since $n + 1$ is positive by lemma Nat.succ_pos, then by lemma Nat.pow_pos_of_pos we obtain that $(n + 1)^2$ is positive.

Authoring these interactive documents by hand would be a chore — can't we take advantage of having already done the tedious work of writing a Lean proof?

Why informalize?

4. We can create an accurate database of informal/formal proof pairs

Machine learning researchers want this data to train their ML models, but it's not a natural product of formalization work.

Given an informalization system, generating these pairs after the fact is easy.

Maybe auto-formalization is learnable as an inverse to informalization?

How not to informalize

The cool thing to do would be to apply an LLM to translate Lean code to English.

However:

- We want output that, while informal, is **not wrong**.
  - Ambiguity, omissions, lack of clarity: these are admissible (but still bugs!).
- LLMs can produce informal text, but we can’t guarantee it’s not wrong.

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 mention the family of sets *t*. What recourse do we have?

How to informalize

Patrick Massot and I have developed a prototype auto-informalizer.

Input: a Lean 4 module

Output: an interactive document (HTML)

We use data produced during Lean 4's elaboration procedure including

- Every proof term
- Every tactic used
- Every tactic state (full local contexts and metavariable contexts)
- The hierarchical structure of the tactic proof

It uses "good old-fashioned AI" principles.

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. $\square$

```
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 theorem statement

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
```

English

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.

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:

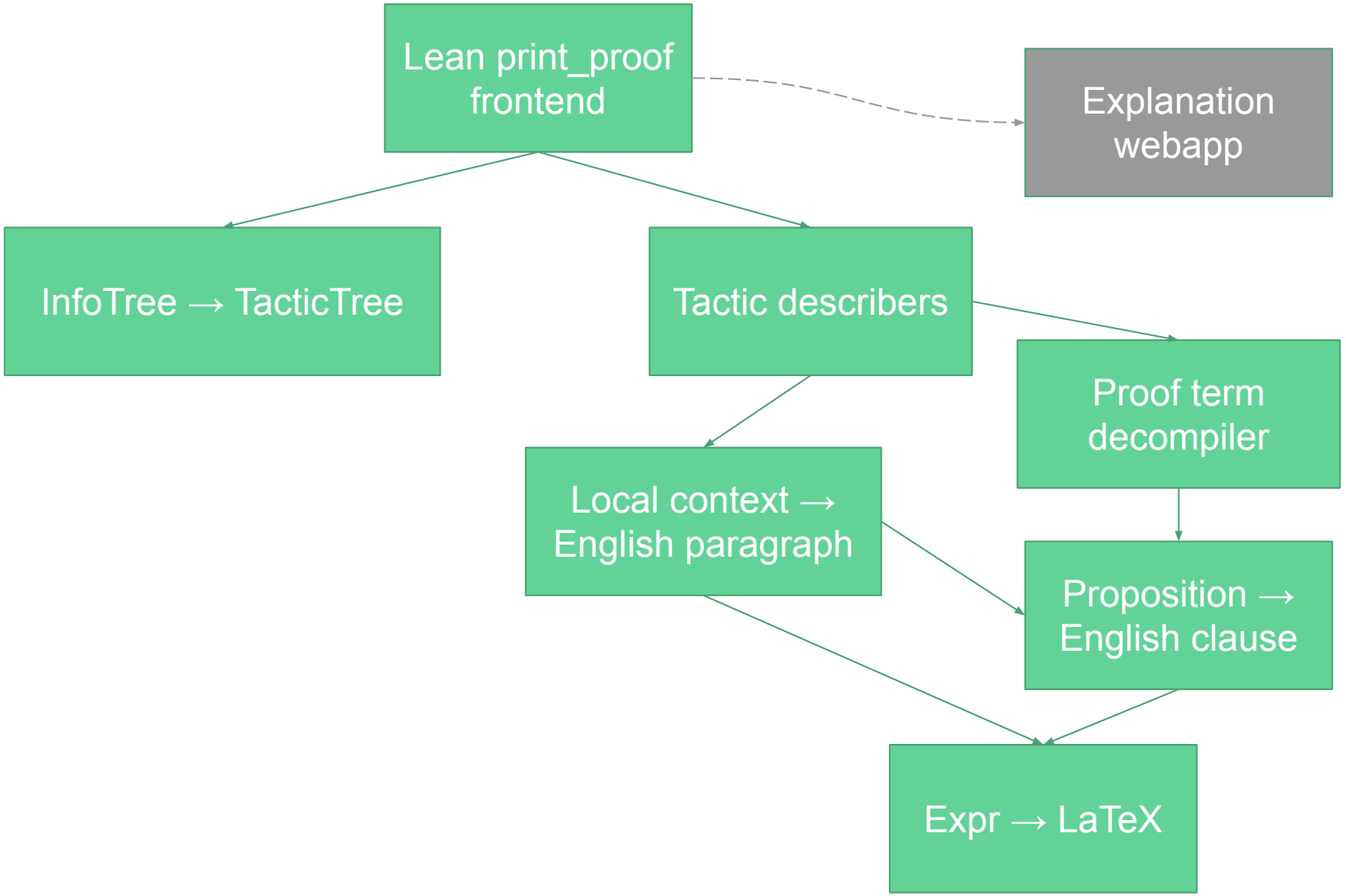
$f(z) \in \overline{f[V \cap A]}$	⊕Using our assumption that $f(z_1)$ tends to $f(z)$ as z_1 tends to z within $V \cap A$ proves this step.
$\subseteq \overline{V'}$	⊕Using our assumption that $f[V \cap A] \subseteq V'$ proves this step.
$= V'$	⊕Using our assumption that V' is closed proves this step.

This completes the proof. □

Let’s look at a demo

Implementation

Rough architecture



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.

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

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)
```

Expressions to LaTeX – some examples

```
#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)
```

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

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
```

Expressions to LaTeX: function applications

Another “impedance mismatch” is from Lean preferring curried form.

> $f(a,b)(c,d)$ is the same as `f a b c d` is the same as $f(a,b,c,d)$.

Principle:

> We allow output to have the usual harmless ambiguities in common notation.
>
> (Though we try not to be ambiguous if possible!)

Propositions to English

We have two main subsystems, the second making heavy use of the first:

```
#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

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

> 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.

Ontologies

In AI, an *ontology* is a formal-ish model of some aspect of the world:
it is a description of what is and what can be.

An ontology permits one to collect data about the world, perform computations with this data, and infer things about the world from these calculations.

Lean 4 has expressions, declarations, metavariables, local contexts, tactic states, tactics, etc. etc. etc.

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 better the calculation we can do to improve how natural the output is.

An ontology for theorem-style paragraphs

```
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)
```

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

Let *entityName* **:** *noun.payload* **be** *article adjectives noun.text* **with** *accessories* **.**

Let *n* be a natural number.

Let *f* : *X* -> *Y* be an injective function.

For all *adjectives noun.inlineText* ***with*** *accessories*, **...**

For all finite types *T* with decidable equality, ...

Basic algorithm for entity construction

We keep a list of entities actively under construction.

For each local variable,

- We use the variable's type to choose a handler.
- The handler decides what the local variable is about.
  - *Ex.* (T : Type) is about T *Ex.* (h : U ∈ Nhd x) is about U
- It looks for the corresponding entity entry, taking into account dependencies.
- It alters the Noun or attaches an Adjective or Accessory, as appropriate.

Example

```
#english ∀ {α : Type _} [DecidableEq α] [Fintype α], 1 + 1 = 2
```

Let α be a finite type with decidable equality. Then $1 + 1 = 2$.

Log:

```
[English] Will handle parameter α : Type ?u.2662.

[English] Using the english_param handler for Type

[English] Have added entity \alpha ▶

[English] Will handle parameter inst.332 : DecidableEq α.

[English] Will look for an entity describing \alpha with
deps #[\alpha] ▶

[English] Found it!

[English] Have updated entity \alpha ▶

[English] Will handle parameter inst.335 : Fintype α.

[English] Will look for an entity describing \alpha with
deps #[\alpha] ▶

[English] Found it!

[English] Have updated entity \alpha ▶

[English] Will handle conclusion 1 + 1 = 2.

[English] Entities flushed ▶
```

```
@[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
```

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.

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 just due to a current limitation.)

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

Key input data structure: the InfoTree

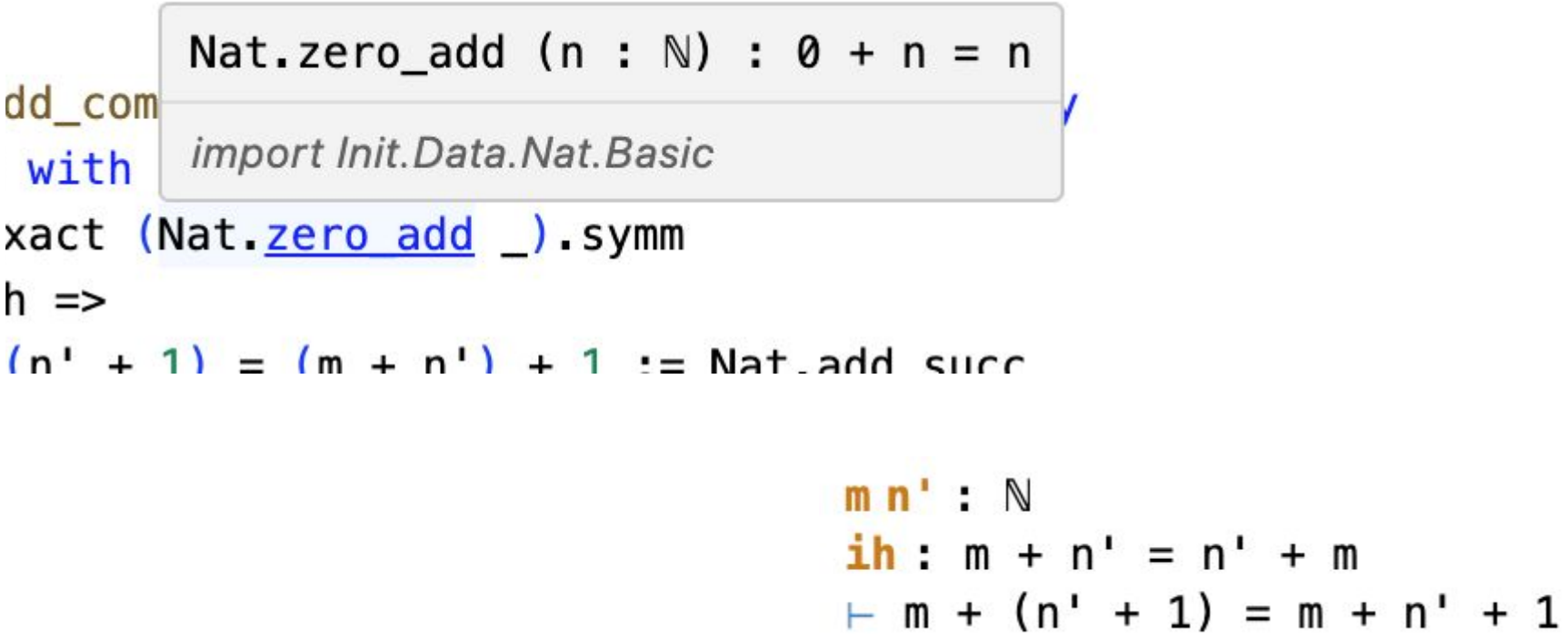
- Lean 4 produces InfoTree objects during elaboration.
- These contain more data than the fully elaborated terms.
- Original purpose: providing all the information one sees in VS Code.

Incidentally, InfoTrees have a hierarchical structure reflecting proof structure.

But it's *rough*.

InfoTrees?

One side-effect of the elaboration process is the creation of “InfoTrees,” which record information needed to support mouseover text, jump-to-definition, and the Infoview.



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>
                                    ...
```

Conversion to TacticTree

As a first pass, we preprocess these to try to link up different steps of a tactic proof.

```
<tactic-record name="[anonymous]" pos="489-573">
  <tactic-state>Goals: [Lean.Name.mkNum `_uniq 245]</tactic-state>
  <tactic-state>Goals: []</tactic-state>
  <syntax>
  by
    intros x y h
    have key : f x = f y := by exact hg _ _ h
    exact hf x y key</syntax>
  <tactic-record name="Lean.Elab.Tactic.evalIntros" pos="494-506">
    <tactic-state>Goals: [Lean.Name.mkNum `_uniq 245]</tactic-state>
    <tactic-state>Goals: [Lean.Name.mkNum `_uniq 250]</tactic-state>
    <syntax>
    intros x y h</syntax>
  </tactic-record>
  <tactic-record name="Lean.Parser.Tactic._aux_Init_Tactics___macroRules_Lean_Parser_Tactic_tacticHave__1" pos="509-554">
    <tactic-state>Goals: [Lean.Name.mkNum `_uniq 250]</tactic-state>
    <tactic-state>Goals: [Lean.Name.mkNum `_uniq 257]</tactic-state>
    <syntax>
    have key : f x = f y := by exact hg _ _ h</syntax>
    <tactic-record name="Lean.Parser.Tactic._aux_Init_Tactics___macroRules_Lean_Parser_Tactic_tacticRefine_lift__1" pos="509-5
      <tactic-state>Goals: [Lean.Name.mkNum `_uniq 250]</tactic-state>
      <tactic-state>Goals: [Lean.Name.mkNum `_uniq 257]</tactic-state>
      <syntax>
      refine_lift
        have key : f x = f y := by exact hg _ _ h;
        ?_</syntax>
      <tactic-record name="[anonymous]" pos="509-554">
        <tactic-state>Goals: [Lean.Name.mkNum `_uniq 250]</tactic-state>
        <tactic-state>Goals: [Lean.Name.mkNum `_uniq 257]</tactic-state>
        <syntax>
        focus
          (refine
              no_implicit_lambda%
                (have key : f x = f y := by exact hg _ _ h;
```

Tactic describers

Then we have “tactic describers” that 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
```

```
/-- 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)
```

What are side goals? Many tactics produce multiple goals that tend to be focused on and proved immediately after. Tactic describers may elect handle them.

```
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
```

Explanations

The output of a tactic describer is more-or-less an Explanation, a piece of a structured document. For example:

- 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 Explanations.

> Getting paragraph breaks to yield correct HTML is more subtle than it might seem. Same with highlighting text on mouseover across block elements.

Proof term explanations

For a number of tactics, we currently compute a proof term and then use a generic procedure to “compile” that proof term to English.

> **Reason 1.** Ideally, we do not want to tactic describers to recapitulate every tactic implementation.
>
> **Reason 2.** The Laziness Principle: a Lean author writes primarily to be understood *by the computer*. We cannot trust that the arguments supplied to tactics or lemmas are meaningful as an explanation for a human.
>
> **Reason 3.** Even given non-lazy authors, not everything that a computer wants to see is similarly desired by a human reader.

```
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

v

Synthesized tactic trees, with synthesized intermediate goal states

v

Rendered Explanations

Ex: if a proof term is a lambda and the expected type is not a forall, we can compute WHNF of the type to discover which definitions are unfolded and then insert this intermediate step.

Future ideas

- Expand the suite of tactic describers and pretty printers
  - How much can be auto-generated from mathlib conventions alone?
- Generate a mathematician-friendly mathlib4_docs
- Write a (chapter or two of a) textbook using this technology
  - How would students respond to it? Looking into using it in Math 100 and Math 110 at UCSC
- Incorporate machine learning to make stylistic decisions
  - This circumscribes ML to a problem where it can be not wrong.
  - This could be used to establish baseline levels of detail.
- What are practical improvements to our ontology that would yield higher-quality output? What does the literature on mathematical language say?

Distant future:

- Build an informalizer that provably produces output that has at least one interpretation that is equivalent to the original Lean.