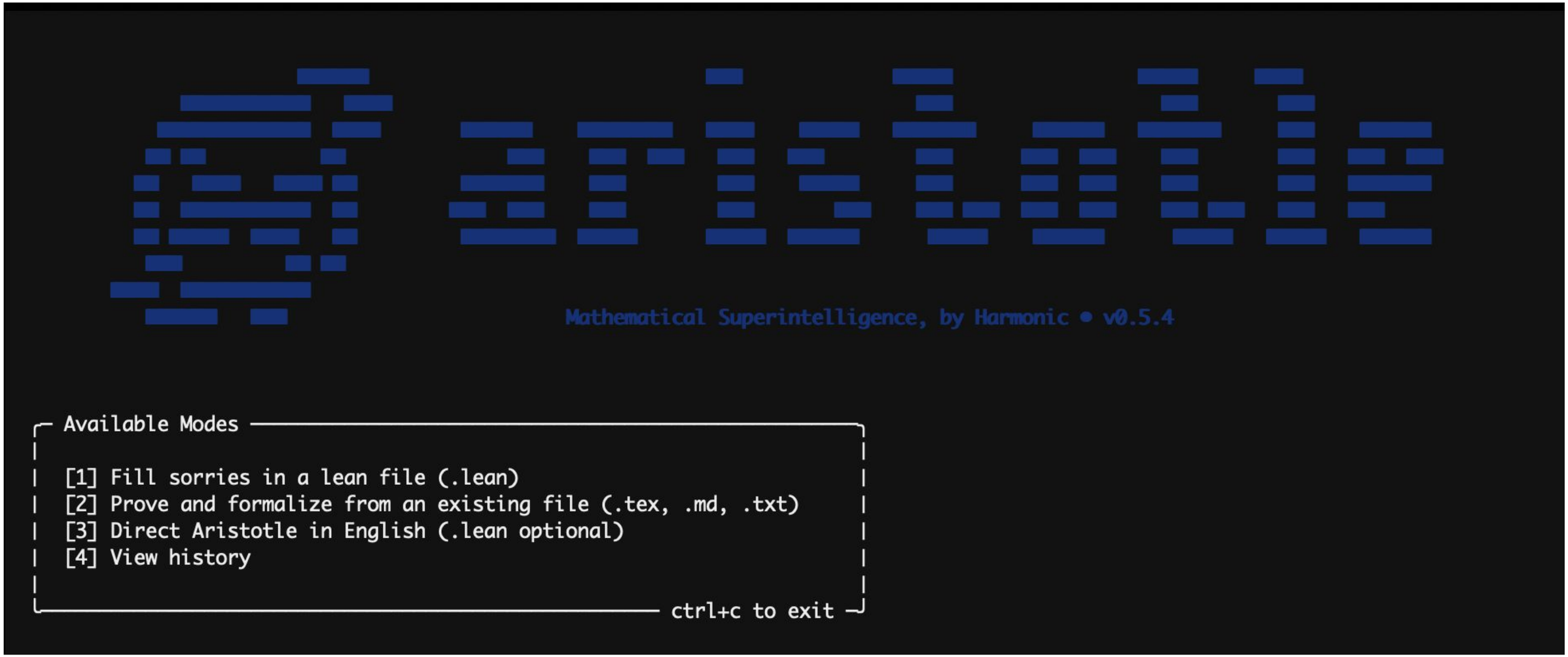

aristotle

Getting Started with Aristotle



Harmonic

Setup

1. [Recommended] Install UV

2. set $ARISTOTLE_API_KEY in your shell init to avoid having to type it frequently.

3. `uvx --from aristotlelib@latest aristotle`

   [Recommended] create an alias to avoid having to type the above every time

4. Aristotle runs on a fixed Lean/Mathlib version (currently 4.24.0).

   You may experience incompatibilities if you are writing in / running a different Lean/Mathlib version.

Filling Sorries in a lean file

Sorry filling mode is designed to work within existing lean projects

```
1 import Mathlib
2
3 lemma sign_cases (a : ℤ) : a.sign =
4   if a < 0 then -1 else if a = 0 then 0 else 1 := by
5   sorry
```

```
1 /-
2 This file was edited by Aristotle.
3 Lean Toolchain version: leanprover/lean4:v4.20.0-rc5
4 Mathlib version: d62eab0cc36ea522904895389c301cf8d844fd69 (May 9, 2025)
5 Your Lean code is run in a custom environment, which uses these headers:
6 set_option maxHeartbeats 0
7 set_option maxRecDepth 4000
8 set_option synthInstance.maxHeartbeats 20000
9 set_option synthInstance.maxSize 128
10
11 The following was proved by Aristotle:
12 - lemma sign_cases (a : ℤ) : a.sign = if a < 0 then -1 else if a = 0 then 0 else 1
13 -/
14
15 import Mathlib
16 lemma sign_cases (a : ℤ) : a.sign = if a < 0 then -1 else if a = 0 then 0 else 1 := by
17   -- We can split into cases based on the value of `a`.
18   cases' lt_trichotomy a 0 with h h <;> aesop
```

Harmonic

Tips for sorry filling mode - Imports

```
/-
Copyright (c) 2024 Sidharth Hariharan. All rights reserved.
Released under Apache 2.0 license as described in the file LICENSE.
Authors: Sidharth Hariharan

M4R File
-/

import SpherePacking.ForMathlib.Fourier
import SpherePacking.ForMathlib.SpecificLimits
import SpherePacking.ForMathlib.tprod
import SpherePacking.ModularForms.Eisenstein
```

- Aristotle will see all files transitively imported by the input file.
- Aristotle will not see files not transitively imported. Aristotle will not add imports.
- Aristotle will not modify any contents of transitively imported files.

Harmonic

Tips for sorry filling mode - Providing an informal solution

When proving a sorry, Aristotle will only see comments above the command.

```
def magic_number : Nat := 42
def is_magic (n : Nat) : Prop := n = magic_number

/-
PROVIDED SOLUTION

Aristotle will see this context.
To solve this, it must look up `is_magic` in Intermediate,
and then `magic_number` in Base to realize that 42 = 42.
-/
theorem check_magic : is_magic 42 := by
  -- Aristotle will NOT see this comment inside the proof block
  sorry

/-
PROVIDED SOLUTION

Aristotle sees this only when proving `not_magic`.
-/
theorem not_magic : ¬ is_magic 10 := by sorry

```

Harmonic

Tips for sorry filling mode - Disproofs

Aristotle will automatically attempt to disprove statements

```
/-
Source: Introduction to Analytic Number Theory, Hildebrand, Ch.4, Ex.3. pg.139.
Statement: Let `f(n) = ∑_{d | n} (log d)/d`, and let `F(s) = ∑_{n=1}^∞ f(n)n^(-s)`.
Evaluate `F(s)` in terms of the Riemann zeta function.
-/

import Mathlib

open Complex Nat

/-- Define `f(n) = ∑_{d | n} (log d) / d`. -/
noncomputable def f (n : ℕ) : ℂ :=
  ∑ d ∈ divisors n, (Complex.log d) / d

/-- Define the Dirichlet series `F(s) = ∑_{n ≥ 1} f(n) / n^s`. -/
noncomputable def F (s : ℂ) : ℂ :=
  ∑' (n : ℕ), f n / n ^ s

theorem hildebrand_4_3 (s : ℂ) :
    F s = - (deriv riemannZeta (s + 1)) * riemannZeta s := by
  sorry
```

```
theorem hildebrand_4_3 (s : ℂ) :
    F s = - (deriv riemannZeta (s + 1)) * riemannZeta s := by
  -- Wait, there's a mistake. We can actually prove the opposite.
  negate_state;
  -- Proof starts here:
  -- By definition of $F$, we know that $F(1) = 0$.
  use 1
  simp [F_one_eq_zero];
  -- We'll use the fact that the derivative of the Riemann zeta function at 2 is not zero.
  apply And.intro;
  · norm_num +zetaDelta at *;
    -- Apply the lemma that states the derivative of the Riemann zeta function at 2 is not zero.
    apply deriv_riemannZeta_two_ne_zero;
  · -- The Riemann zeta function at 1 is the harmonic series, which diverges.
    apply riemannZeta_one_ne_zero

-/
theorem hildebrand_4_3 (s : ℂ) :
    F s = - (deriv riemannZeta (s + 1)) * riemannZeta s := by
  sorry
```

Harmonic

Tips for sorry filling mode - Disproofs

and sometimes it will correct them

```
/-
The theorem holds for Re(s) > 1.
We use the convolution theorem for L-series.
The L-series of f is the product of L-series of (log n)/n and 1.
L((log n)/n) is -zeta'(s+1).
L(1) is zeta(s).
We check convergence conditions (abscissas are 0 and 1 respectively).
-/
theorem hildebrand_4_3_correct (s : ℂ) (hs : 1 < s.re) :
  F s = - (deriv riemannZeta (s + 1)) * riemannZeta s := by
    -- Start by using the convolution theorem to express $F(s)$ as a product of L-series. We have $F(s) = LSeries (LSeries.convolution (fun n => if n = 0 then 0 else (Complex.log n) / n) (fun n => if n = 0 then 0 else 1)) s$.
    have hF_conv : F s = LSeries (LSeries.convolution (fun n => if n = 0 then 0 else (Complex.log n) / n) (fun n => if n = 0 then 0 else 1)) s := by
      -- Substitute the definition of `f` into `F(s)`.
      have hF_subst : F s = LSeries f s := by
        exact tsum_congr fun n => by unfold LSeries.term; aesop;
      rw [ hF_subst, f_eq_convolution ];
    rw [ hF_conv, LSeries_convolution ];
    · -- Substitute the results from the lemmas into the product.
      rw [LSeries_log_div_n_eq_neg_deriv_zeta s hs, LSeries_one_eq_zeta s hs];
    · refine' lt_of_le_of_lt _ ( EReal.coe_lt_coe_iff.mpr hs );
      -- Apply the lemma that states the abscissa of absolute convergence of the sequence `n ↦ (log n)/n` is at most 0.
      apply le_trans (abscissa_log_div_n_le_zero) (by norm_num);
    · refine' lt_of_le_of_lt ( abscissa_one_le_one ) _;
      exact_mod_cast hs
```

```
noncomputable section AristotleLemmas

#check LSeries
#check LSeries_convolution
#check ArithmeticFunction
#check ArithmeticFunction.zeta
#check deriv

/-
The function f is the Dirichlet convolution of (log n)/n and 1.
-/
lemma f_eq_convolution : f = LSeries.convolution (fun n => if n = 0 then 0 else (Complex.log n) / n) (fun n => if n = 0 then 0 else 1) := by
  ext1 m; erw [ LSeries.convolution ] ; aesop;
  simp +decide [ mul_comm, div_eq_mul_inv, zero_lt_one, Finset.sum_filter, Nat.divisorsAntidiagonal, toArithmeticFunction ];
  refine' Finset.sum_bij ( fun x hx => ( x, m / x ) ) _ _ _ _ <;> aesop;
  · exact Nat.pos_of_dvd_of_pos left ( Nat.pos_of_ne_zero right );
  · exact Nat.le_of_dvd ( Nat.pos_of_ne_zero right ) left;
  · rw [ Nat.mul_div_cancel' left ];
  · exact dvd_of_mul_right_eq _ left_1;
  · exact absurd h_2 ( not_lt_of_ge ( Nat.le_of_dvd ( Nat.pos_of_ne_zero right ) left ) )

/-
The L-series of the function that is 1 for n > 0 is the Riemann zeta function.
-/
lemma LSeries_one_eq_zeta (s : ℂ) (hs : 1 < s.re) : LSeries (fun n => if n = 0 then 0 else 1) s = riemannZeta s := by
  -- The sum over n of (if n = 0 then 0 else 1) / n^s is equal to the sum over n ≥ 1 of 1 / n^s.
  have h_sum : ∑' n : ℕ, (if n = 0 then 0 else 1) / (n : ℂ) ^ s = ∑' n : ℕ+, (1 : ℂ) / (n : ℂ) ^ s := by
    rw [ tsum_congr ] ; aesop;
    rw [ Complex.zero_cpow ( by rintro rfl; norm_num at hs ) ];
  convert h_sum using 1;
  · exact tsum_congr fun n => by aesop;
  · exact?

/-
The L-series of (log n)/n at s is equal to -zeta'(s+1).
Proof idea:
1. L((log n)/n, s) = sum (log n)/n * n^-s = sum log n * n^-(s+1).
2. zeta'(z) = - sum log n * n^-z (derivative of Dirichlet series).
3. So -zeta'(s+1) = sum log n * n^-(s+1).
4. They match.
Use LSeries.deriv and the fact that zeta is LSeries of 1.
-/
lemma LSeries_log_div_n_eq_neg_deriv_zeta (s : ℂ) (hs : 1 < s.re) :
  LSeries (fun n => if n = 0 then 0 else (Complex.log n) / n) s = - deriv riemannZeta (s + 1) := by
    have h1 : deriv riemannZeta (s + 1) = -LSeries (LSeries.logMul (fun n => if n = 0 then 0 else 1)) (s + 1) := by
      -- Apply the formula for the derivative of the L-series of a function.
      have h_deriv : deriv (LSeries (fun n => if n = 0 then 0 else 1)) (s + 1) = -LSeries (LSeries.logMul (fun n => if n = 0 then 0 else 1)) (s + 1) := by
        -- Apply the fact that the derivative of the L-series is the sum of the derivatives of each term.
        have h_deriv : deriv (LSeries (fun n => if n = 0 then 0 else 1)) (s + 1) = -LSeries (LSeries.logMul (fun n => if n = 0 then 0 else 1)) (s + 1) := by
          have h_abs_conv : LSeries.abscissaOfAbsConv (fun n => if n = 0 then 0 else 1) < (s + 1).re := by
            refine' lt_of_le_of_lt ( csInf_le _ ⟨ 2, _, rfl ⟩ ) _ <;> norm_num;
            · -- The series $\sum_{n=1}^{\infty} n^{-2}$ is a p-series with $p=2$, which converges.
              have h_pseries : Summable (fun n : ℕ => (n : ℂ)⁻¹ ^ 2) := by
                exact Summable.of_norm <| by simp;
              rw [ LSeriesSummable ];
              convert h_pseries using 1;
              ext ( _ | n ) <;> norm_num [ LSeries.term ];
              · exact_mod_cast ( by linarith : ( 2 : ℝ ) < s.re + 1 )
            exact?;
          exact h_deriv;
        convert h_deriv using 1;
        refine' Filter.EventuallyEq.deriv_eq _;
        filter_upwards [ IsOpen.mem_nhds ( isOpen_lt continuous_const Complex.continuous_re ) ( show 1 < ( s + 1 |> Complex.re ) by norm_num; linarith ) ] with x hx using LSeries_one_eq_zeta x hx ▸ rfl;
      simp_all +decide [ LSeries, LSeries.logMul ];
      -- Since the terms of the two series are identical, their sums must be equal.
      simp [LSeries.term, LSeries.logMul];
      refine' tsum_congr fun n => _ ; aesop;
    rw [ Complex.cpow_add _ _ ] <;> norm_num [ h ] ; ring
```

Harmonic

Tips for sorry filling mode - Working with data

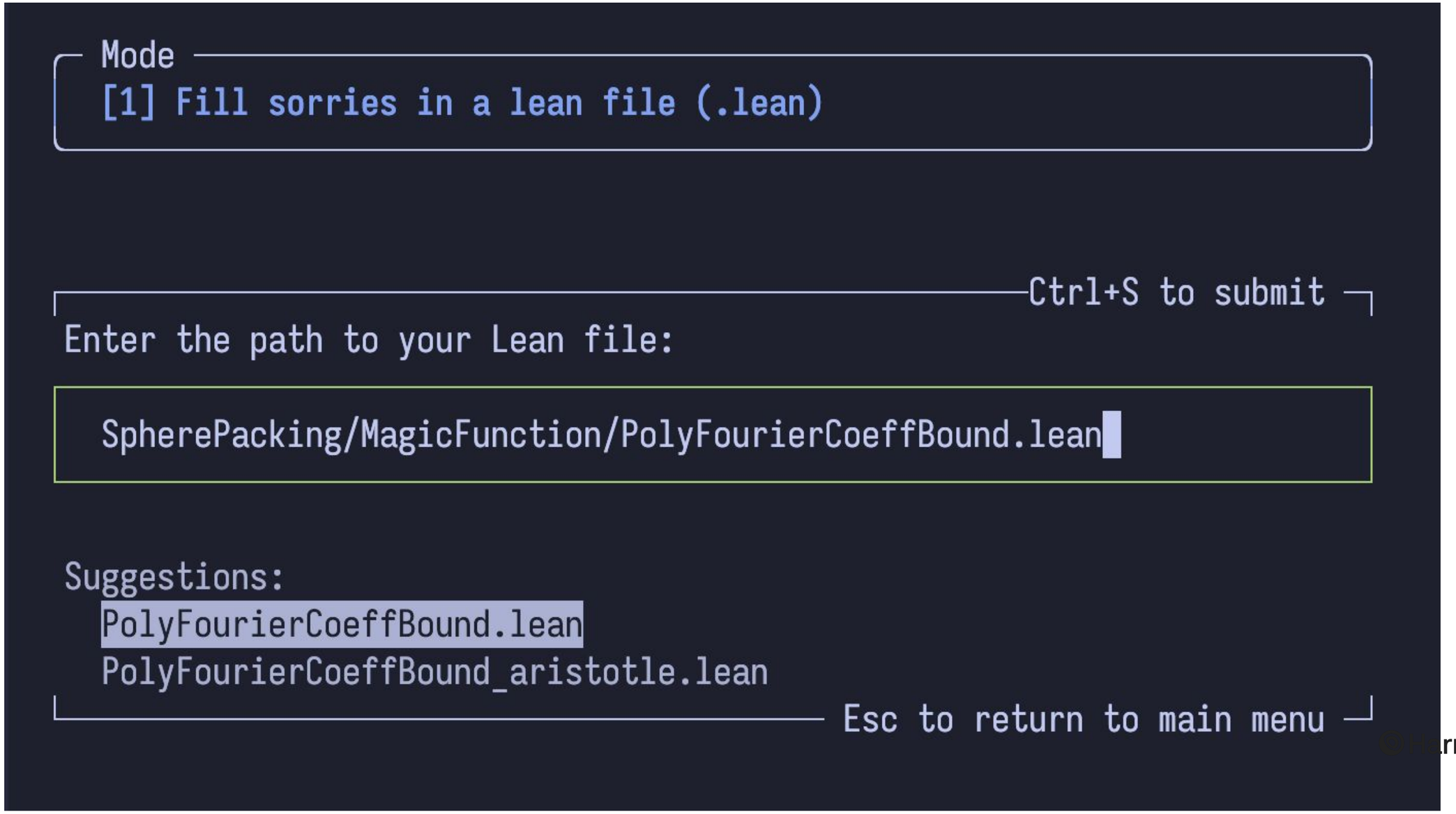
Aristotle will not touch your data, unless you make it irrelevant e.g. using `Nonempty`:

```
def foo : Nat := sorry -- Aristotle will not touch this
def bar : Nonempty Nat := sorry -- Aristotle will prove this
```

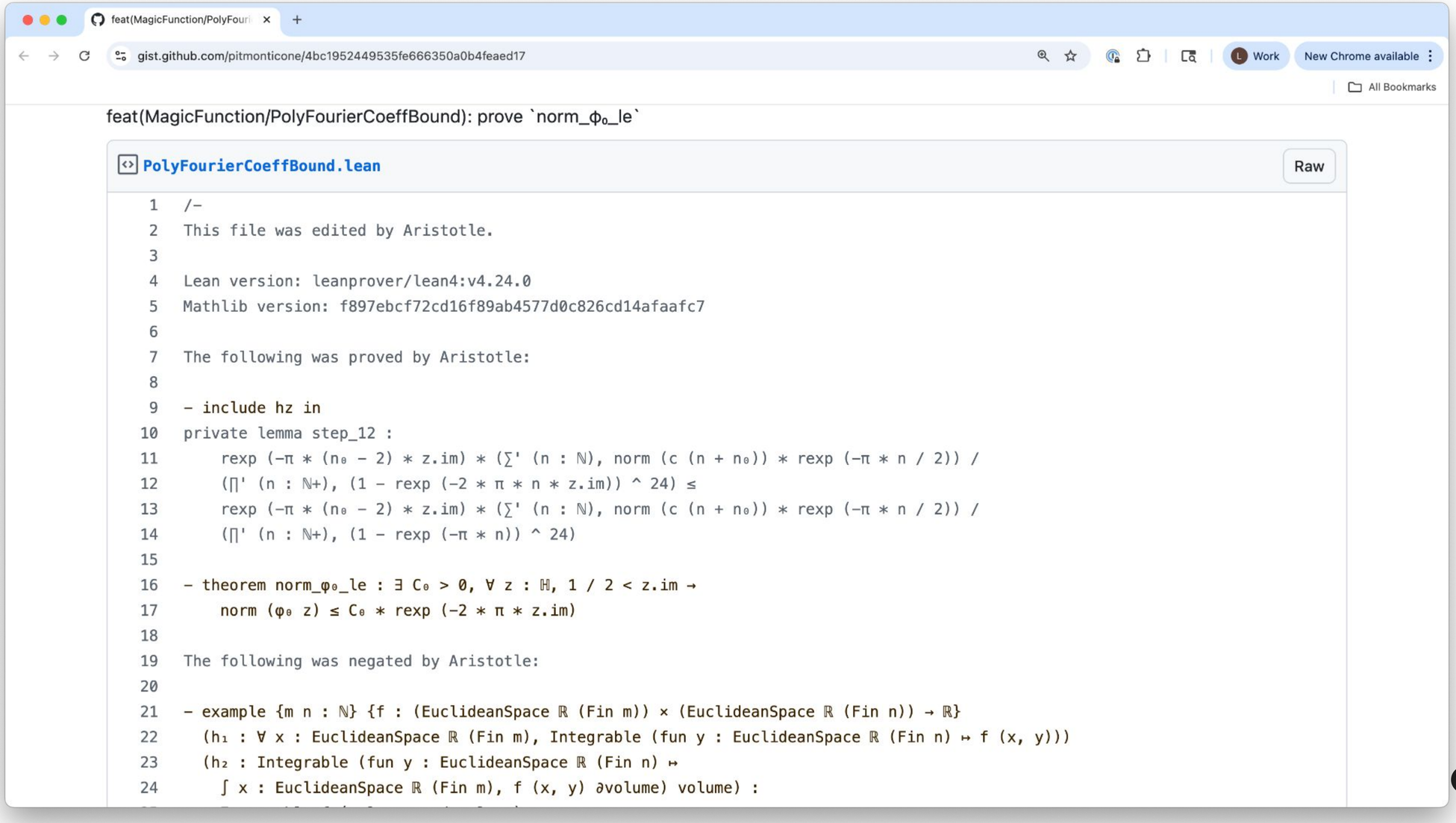
Aristotle will, however, happily create its own data. e.g. it may output

```
noncomputable section AristotleLemmas
def some_aux_construction := …
end AristotleLemmas
theorem baz : … := …
```

Sorry filling mode - real example from the SpherePackingProject



Sorry filling mode - real example from the SpherePackingProject



Harmonic

Directing Aristotle in English


```lean
/-
This file was generated by Aristotle.
Lean Toolchain version: leanprover/lean4:v4.20.0-rc5
Mathlib version: d62eab0cc36ea522904895389c301cf8d844fd69 (May 9, 2025)
-/
/-
This module implements Newton's method for root finding and proves its local convergence.
It defines `newton_step` and `newton_seq`, and proves `newton_convergence_local`, which states that
if a function $f$ is $C^2$, $f(x^*) = 0$, and $f'(x^*) \neq 0$, then for any initial guess $x_0$
sufficiently close to $x^*$, the Newton sequence converges to $x^*$.
-/

noncomputable section
def newton_step (f f' : ℝ → ℝ) (x : ℝ) : ℝ := x - f x / f' x
def newton_seq (f f' : ℝ → ℝ) (x0 : ℝ) : ℕ → ℝ
| 0 => x0
| n + 1 => newton_step f f' (newton_seq f f' x0 n)

theorem newton_convergence_local
  (f f' : ℝ → ℝ) (x_star : ℝ)
  (h_root : f x_star = 0)
  (h_deriv : ∀ x, HasDerivAt f (f' x) x)
  (h_c2 : ContDiff ℝ 2 f)
  (h_f'_ne_0 : f' x_star ≠ 0) :
∃ δ > 0, ∀ x0, |x0 - x_star| < δ →
  Filter.Tendsto (newton_seq f f' x0) Filter.atTop (nhds x_star) := by
    -- We'll use the fact that if the derivative is non-zero and continuous, then the Newton step is a contraction mapping.
    obtain ⟨δ1, hδ1_pos, hδ1⟩ : ∃ δ1 > 0, ∀ x, abs (x - x_star) < δ1 → f' x ≠ 0 := by
      -- Since $f'$ is continuous at $x^*$ and $f'(x^*) \neq 0$, there exists a $\delta_1 > 0$ such that for all $x$ with $|x - x^*| < \delta_1$, $|f'(x) - f'(x^*)| < |f'(x^*)|$.
      have h_cont : ContinuousAt f' x_star := by
        rw [ show f' = fun x => deriv f x from funext fun x => HasDerivAt.deriv ( h_deriv x ) ▸ rfl ] ; exact h_c2.continuous_deriv ( by norm_num ) |> Continuous.continuousAt;
      exact Metric.mem_nhds_iff.mp ( h_cont.eventually_ne h_f'_ne_0 )
    have h_cont : ContinuousOn (fun x => f x / f' x) (Metric.ball x_star δ1) := by
      -- The function $f(x)$ is continuous on $Metric.ball x_star δ1$ since it is the composition of continuous functions.
      have h_f_cont : ContinuousOn f (Metric.ball x_star δ1) := by
        exact h_c2.continuous.continuousOn;
      refine' h_f_cont.div _ _;
      · exact ContinuousOn.congr ( show ContinuousOn ( fun x => deriv f x ) ( Metric.ball x_star δ1 ) from h_c2.continuous_deriv ( by norm_num ) |> Continuous.continuousOn ) fun x hx => HasDerivAt.deriv ( h_deriv x ) ▸ rfl;
      · aesop
    obtain ⟨δ, hδ_pos, hδ⟩ : ∃ δ > 0, ∀ x, abs (x - x_star) < δ → abs ((x - f x / f' x) - x_star) ≤ 1 / 2 * abs (x - x_star) := by
      have h_deriv_zero : HasDerivAt (fun x => x - f x / f' x) 0 x_star := by
        convert HasDerivAt.sub ( hasDerivAt_id x_star ) ( HasDerivAt.div ( h_deriv x_star ) ( hasDerivAt_deriv_iff.mpr _ ) _ ) using 1 <;> aesop;
        · rw [ sq, mul_div_mul_right ] <;> aesop;
        · have h_diff : ContDiff ℝ 1 f' := by
            have h_diff : ContDiff ℝ (1 + 1) f := by
              exact h_c2;
            rw [ show f' = fun x => deriv f x from funext fun x => HasDerivAt.deriv ( h_deriv x ) ▸ rfl ] ; erw [ contDiff_succ_iff_deriv ] at h_diff ; tauto;
          exact h_diff.contDiffAt.differentiableAt ( by norm_num );
      have := Metric.tendsto_nhds_nhds.1 h_deriv_zero.isLittleO.tendsto_div_nhds_zero; aesop;
      obtain ⟨ δ, hδ_pos, H ⟩ := this ( 2⁻¹ ) ( by norm_num ) ; exact ⟨ δ, hδ_pos, fun x hx => if h : x = x_star then by norm_num [ h, h_root ] else by have := H hx; rw [ div_lt_iff₀ ( abs_pos.mpr ( sub_ne_zero.mpr h ) ) ] at this; linarith ⟩ ;
    -- By induction, we can show that for all n, the nth term of the sequence is within δ of x_star.
    have h_seq_bound : ∀ x0, abs (x0 - x_star) < δ → ∀ n, abs ((newton_seq f f' x0 n) - x_star) ≤ (1 / 2) ^ n * abs (x0 - x_star) := by
      intro x0 hx0 n; induction n <;> simp_all +decide [ pow_succ', mul_assoc ] ;
      · rfl;
      · convert le_trans ( hδ _ _ ) ( mul_le_mul_of_nonneg_left ‹_› ( by norm_num ) ) using 1;
        exact lt_of_le_of_lt ‹_› ( by exact lt_of_le_of_lt ( mul_le_of_le_one_left ( abs_nonneg _ ) ( inv_le_one_of_one_le₀ ( one_le_pow₀ ( by norm_num ) ) ) ) hx0 );
    exact ⟨ δ, hδ_pos, fun x0 hx0 => tendsto_iff_norm_sub_tendsto_zero.mpr <| squeeze_zero ( fun _ => norm_nonneg _ ) ( fun n =>
```

Harmonic

Directing Aristotle in English - Autoformalizing and proving

Note: Anything you put here is treated as assumptions, *not* sorries to fill. Aristotle will not touch any of the existing content in this file, and will treat everything in it as true.

Example: if you put `theorem MyFLT : FermatLastTheorem := sorry` in the formal context, then prompt “Prove FLT for n=19”: Aristotle will not attempt to prove FLT; it will prove FLT for n=19 assuming FLT is true.

Harmonic

Directing Aristotle in English - Giving Aristotle access to your library

MyLibrary.lean:

```
import Mathlib
import MyLibrary
import CSLib
```

```
description of Newton
Iteration
```

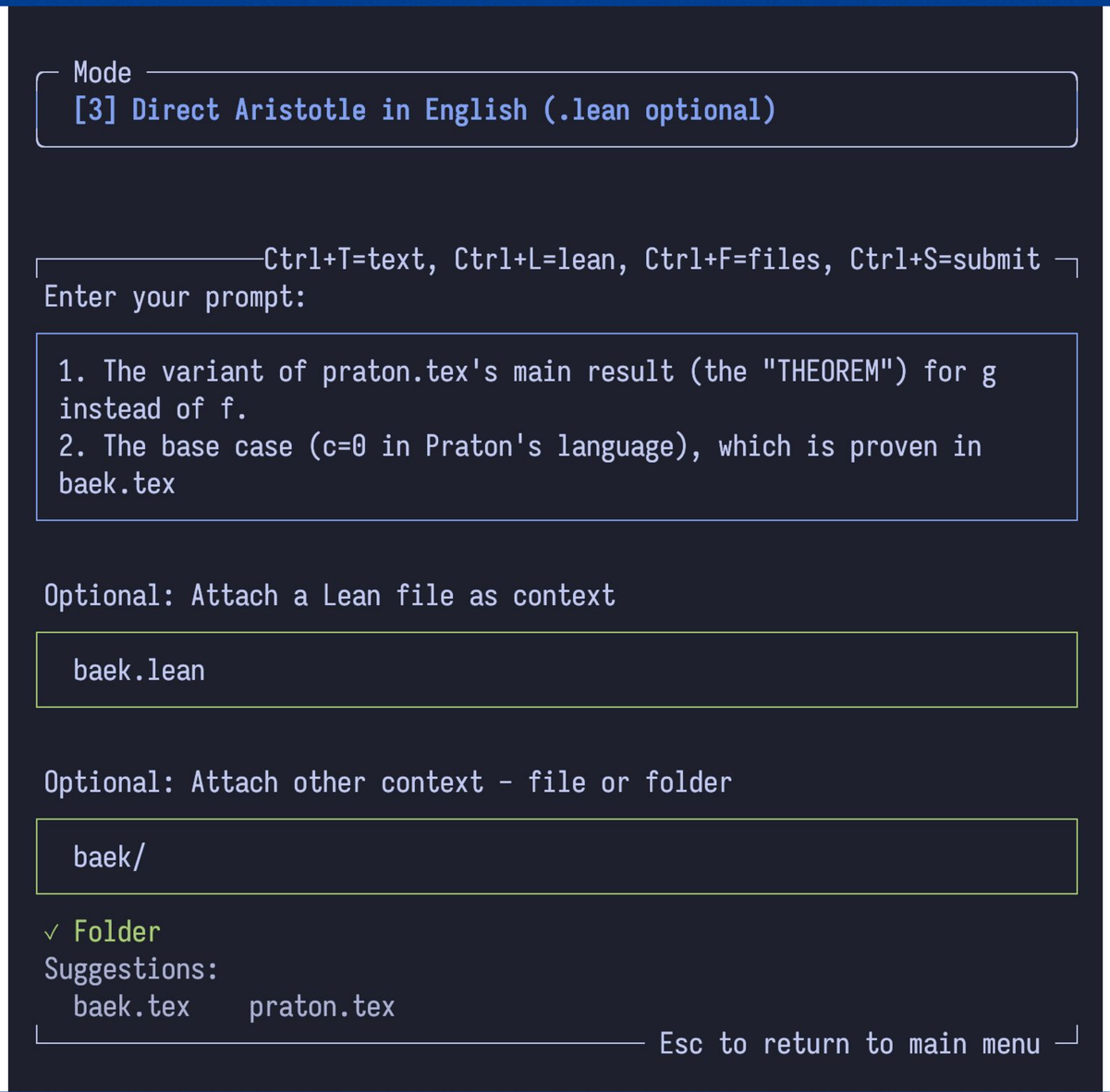
Harmonic

Directing Aristotle in English - Continuing a query


```
Fix three positive integers
n, k, m

Prove that a subgroup H of
S_{6+(n+k+m)} generated by

g1:=G!(1,6,4,3,a_1,...a_n);
g2:=G!(1,2,4,5,b_1,...,b_k);
g3:=G!(5,6,2,3,c_1,...,c_m);
H:=sub<G|[g1,g2,g3]>;

satisfies H = S_{6+(n+k+m)}.

Use a theorem of Jordan on
primitive groups to show that
the
group H is either A_{6+(n+k+m)}
or S_{6+(n+k+m)}. Conclude by
pointing to a permutation which
is odd.
```

Harmonic

Directing Aristotle in English - Real example (Erdos 106, 1026)



Harmonic

Directing Aristotle in English - Real example (Erdos 106, 1026)

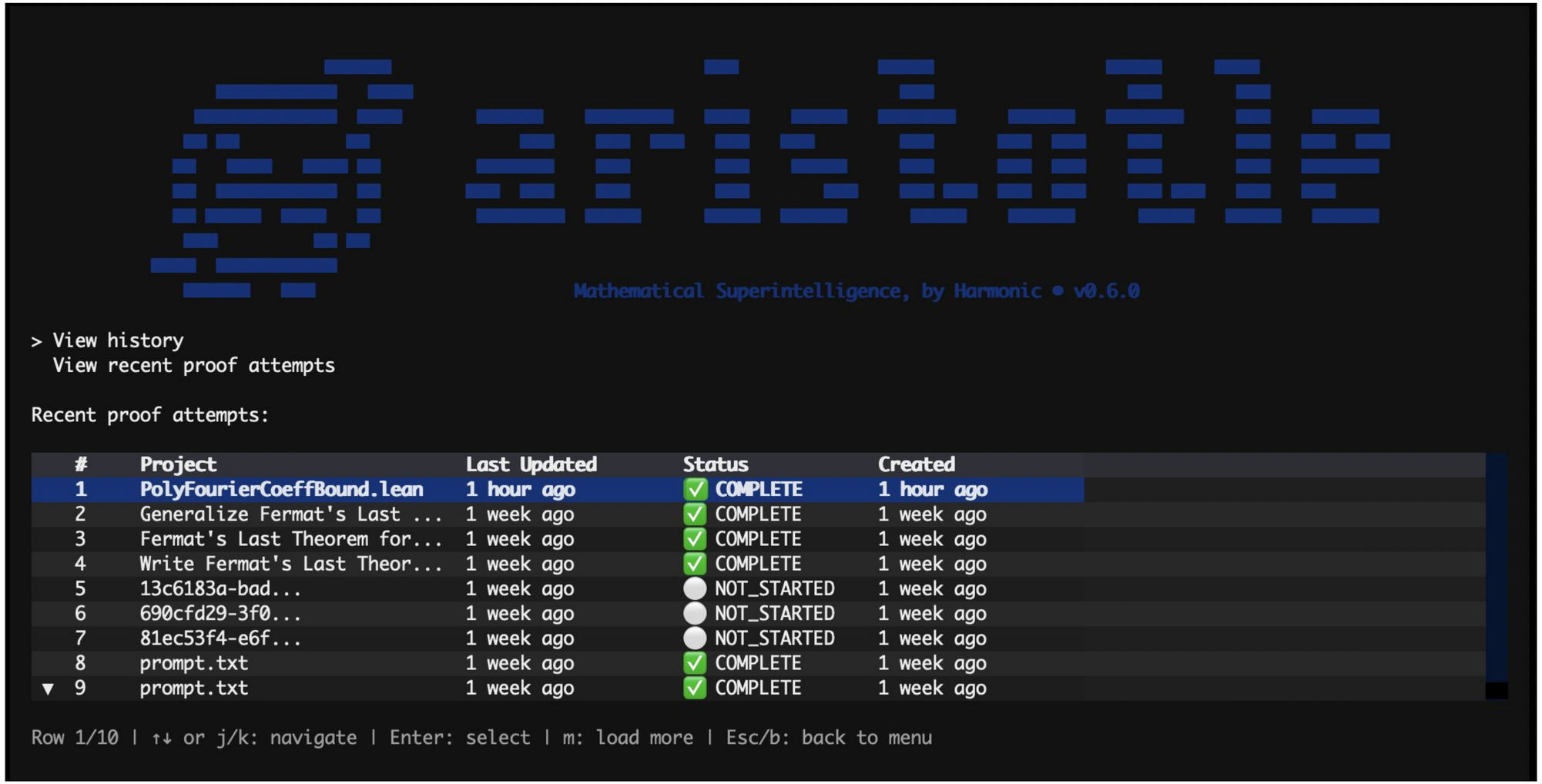
vim 22233a27-a0b5-463d-9f7d-7d1e52d1388b-output.txt

```
theorem base_case (k : ℕ) (hk : k > 0) : g (k ^ 2 + 1) = k := by
  simp +zetaDelta at *;
  by_contra h_contra;
  -- Since $g$ is the supremum of total side lengths, if $g > k$, there exists a packing $P$ such that $P.total_side_length > k$.
  obtain ⟨P, hP_valid, hP_len⟩ : ∃ P : Packing (k ^ 2 + 1), P.is_valid ∧ P.total_side_length > k := by
    by_cases h : g ( k ^ 2 + 1 ) > k;
    · have := exists_lt_of_lt_csSup ?_ h; aesop;
      exact ⟨ _, ⟨ fun _ => ⟨ 0, 0, 0, by norm_num, by norm_num, by norm_num, by norm_num, by norm_num ⟩, fun _ _ _ => by norm_num [ Square.disjoint ], rfl ⟩ ⟩;
    · exact False.elim <| h_contra <| le_antisymm ( le_of_not_gt h ) <| base_case_lower k hk |> le_trans ( by norm_num );
  -- By `exists_shift_ho`, there exists a shift `a` in `[0, 1/k)` such that `∑ i, count_intersections_ho k a (P i) ≥ k^2 + 1`.
  obtain ⟨a, ha₀, ha₁⟩ : ∃ a, 0 ≤ a ∧ a < 1 / (k : ℝ) ∧ ∑ i, count_intersections_ho k a (P i) ≥ k * k + 1 := by
    field_simp;
    exact Exists.elim ( exists_shift_ho k hk P hP_len ) fun a ha => ⟨ a, ha.1, by nlinarith [ show ( k : ℝ ) ≥ 1 by norm_cast, mul_div_cancel₀ 1 ( by positivity : ( k : ℝ ) ≠ 0 ), ha.2.1 ], ha.2.2 ⟩;
  -- Let `m i = count_intersections_ho k a (P i)` and `d i = (P i).s`.
  set m : Fin (k ^ 2 + 1) → ℕ := fun i => count_intersections_ho k a (P i)
  set d : Fin (k ^ 2 + 1) → ℝ := fun i => (P i).s;
  -- We verify the conditions for `baek_combinatorial`:
  have h_sum_m : ∑ i, m i ≥ k * k + 1 := by
    exact ha₁.2
  have h_sum_m_d : ∑ i, m i * d i ≤ k := by
    convert sum_intersections_le_k_ho k hk P hP_valid a ⟨ ha₀, ha₁.1 ⟩ using 1
  have h_m_zero : ∀ i, m i = 0 → d i ≤ 1 / k := by
    bound;
    convert small_squares_miss_lines_ho k hk a ( P i ) ⟨ ha₀, left ⟩ a_1 using 1
  have h_m_two : ∀ i, m i ≥ 2 → d i ≥ 1 / k := by
    intros i hi; exact (by
    apply large_squares_hit_lines_ho k hk a (P i) ⟨ha₀, ha₁.left⟩ hi);
  -- Applying `baek_combinatorial`, we get `∑ d i ≤ k`.
  have h_sum_d : ∑ i, d i ≤ k := by
    apply baek_combinatorial;
    any_goals tauto;
    ring;
  exact hP_len.not_le h_sum_d

/-
For all integers c, g(k^2 + 2c + 1) = k + c/k for all k > |c|.
-/
theorem main_theorem (c : ℤ) : Q c := by
  apply praton_variant_proven;
  use 0;
  unfold Q;
  aesop;
  cases k <;> aesop;
  convert base_case a_1 a using 1
```

22233a27-a0b5-463d-9f7d-7d1e52d1388b-output.txt 902,1 Bot

Harmonic

Monitoring and Accessing Previous Results



Harmonic