



Getting Started with Aristotle



Available Modes

- [1] Fill sorries in a lean file (.lean)
- [2] Prove and formalize from an existing file (.tex, .md, .txt)
- [3] Direct Aristotle in English (.lean optional)
- [4] View history

ctrl+c to exit

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
```

Tips for sorry filling mode - Imports

```
1  /-
2  Copyright (c) 2024 Sidharth Hariharan. All rights reserved.
3  Released under Apache 2.0 license as described in the file LICENSE.
4  Authors: Sidharth Hariharan
5
6  M4R File
7  -/
8
9  import SpherePacking.ForMathlib.Fourier
10 import SpherePacking.ForMathlib.SpecificLimits
11 import SpherePacking.ForMathlib.tprod
12 import SpherePacking.ModularForms.Eisenstein
13
```

- 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.

Tips for sorry filling mode - Providing an informal solution

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

```
1 def magic_number : Nat := 42
2 def is_magic (n : Nat) : Prop := n = magic_number
3
4 -/
5 PROVIDED SOLUTION
6
7 Aristotle will see this context.
8 To solve this, it must look up `is_magic` in Intermediate,
9 and then `magic_number` in Base to realize that 42 = 42.
10 -/
11 theorem check_magic : is_magic 42 := by
12 | -- Aristotle will NOT see this comment inside the proof block
13 | sorry
14 |
15 -/
16 PROVIDED SOLUTION
17
18 Aristotle sees this only when proving `not_magic`.
19 -/
20 theorem not_magic : ¬ is_magic 10 := by sorry
21
```

Tips for sorry filling mode - Disproofs

Aristotle will automatically attempt to disprove statements

```
1 /-
2 Source: Introduction to Analytic Number Theory, Hildebrand, Ch.4, Ex.3. pg.139.
3 Statement: Let ' $f(n) = \sum_{d|n} (\log d)/d$ ', and let ' $F(s) = \sum_{n=1}^{\infty} f(n)n^{-s}$ '. Evaluate ' $F(s)$ ' in terms of the Riemann zeta function.
4 -/
5
6 import Mathlib
7
8 open Complex Nat
9
10 /-- Define ' $f(n) = \sum_{d|n} (\log d)/d$ '. -/
11 noncomputable def f (n : ℕ) : ℂ :=
12 | Σ d ∈ divisors n, (Complex.log d) / d
13
14 /-- Define the Dirichlet series ' $F(s) = \sum_{n \geq 1} f(n) / n^s$ '. -/
15 noncomputable def F (s : ℂ) : ℂ :=
16 | Σ' (n : ℕ), f n / n ^ s
17
18 theorem hildebrand_4_3 (s : ℂ) :
19 | F s = - (deriv riemannZeta (s + 1)) * riemannZeta s := by
20 sorry
```

```
358 theorem hildebrand_4_3 (s : ℂ) :
359 | F s = - (deriv riemannZeta (s + 1)) * riemannZeta s := by
360 -- Wait, there's a mistake. We can actually prove the opposite.
361 negate_state;
362 -- Proof starts here:
363 -- By definition of $F$, we know that $F(1) = 0$.
364 use 1
365 simp [F_one_eq_zero];
366 -- We'll use the fact that the derivative of the Riemann zeta function at 2 is not zero.
367 apply And.intro;
368 | norm_num +zetaDelta at *;
369 -- Apply the lemma that states the derivative of the Riemann zeta function at 2 is not zero.
370 apply deriv_riemannZeta_two_ne_zero;
371 -- The Riemann zeta function at 1 is the harmonic series, which diverges.
372 | apply riemannZeta_one_ne_zero
373
374 -/
375 theorem hildebrand_4_3 (s : ℂ) :
376 | F s = - (deriv riemannZeta (s + 1)) * riemannZeta s := by
377 sorry
```

Tips for sorry filling mode - Disproofs

and sometimes it will correct them

```
192 /-
193 The theorem holds for  $\operatorname{Re}(s) > 1$ .
194 We use the convolution theorem for L-series.
195 The L-series of  $f$  is the product of L-series of  $(\log n)/n$  and 1.
196  $L((\log n)/n)$  is  $-\zeta'(s+1)$ .
197  $L(1)$  is  $\zeta(s)$ .
198 We check convergence conditions (abscissas are 0 and 1 respectively).
199 -/
200 theorem hildebrand_4_3_correct (s : ℂ) (hs : 1 < s.re) :
201 | F s = - (deriv riemannZeta (s + 1)) * riemannZeta s := by
202 | -- Start by using the convolution theorem to express  $F(s)$  as a product of L-series. We have  $F(s) = \text{LSeries}(\text{convolution}(\text{fun } n \Rightarrow \text{if } n = 0 \text{ then } 0 \text{ else } (\text{Complex.log } n) / n)(\text{fun } n \Rightarrow \text{if } n = 0 \text{ then } 0 \text{ else } 1)) s$  := by
203 | have hF_conv : F s = 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
204 | -- Substitute the definition of 'F' into 'F(s)'.
205 | have hF_subst : F s = LSeries f s := by
206 | | exact tsum_congr fun n => by unfold LSeries.term; aesop;
207 | rw [ hF_subst, f_eq_convolution ];
208 | rw [ hF_conv, LSeries_convolution ];
209 | -- Substitute the results from the lemmas into the product.
210 | rw [ LSeries_log_div_n_eq_neg_deriv_zeta s hs, LSeries_one_eq_zeta s hs];
211 | refine' lt_of_le_of_lt _ ( EReal.coe_lt_coe_iff.mpr hs );
212 | -- Apply the lemma that states the abscissa of absolute convergence of the sequence ' $n \mapsto (\log n)/n$ ' is at most 0.
213 | apply le_trans (abscissa_log_div_n_le_zero) (by norm_num);
214 | refine' lt_of_le_of_lt (abscissa_one_le_one) _;
215 | exact_mod_cast hs
```

```
57 noncomputable section AristotleLemmas
58
59 #check LSeries
60 #check LSeries.convolution
61 #check ArithmeticFunction
62 #check ArithmeticFunction.zeta
63 #check deriv
64
65 /-
66 The function  $f$  is the Dirichlet convolution of  $(\log n)/n$  and 1.
67 -/
68 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
69 ext1 m; erw [ LSeries.convolution ]; aesop;
70 simp +decide [ mul.comm, div_eq_mul_inv, zero_lt_one, Finset.sum_filter, Nat.divisorsAntidiagonal, toArithmeticFunction ];
71 refine' Finset.sum_bij ( fun x hx => (x, m / x) ) _ _ _ <=> aesop;
72 · exact Nat.pos_of_dvd_left (Nat.pos_of_ne_zero_right);
73 · exact Nat.le_of_dvd (Nat.pos_of_ne_zero_right) left;
74 · rw [ Nat.mul_div_cancel' left ];
75 · exact dvd_of_mul_right_eq_left;
76 · exact absurd h.2 (not_lt_of_ge (Nat.le_of_dvd (Nat.pos_of_ne_zero_right) left));
77 -/
78 /-
79 The L-series of the function that is 1 for  $n > 0$  is the Riemann zeta function.
80 -/
81 lemma LSeries_one_eq_zeta (s : ℂ) (hs : 1 < s.re) : LSeries (fun n => if n = 0 then 0 else 1) s := by
82 | -- The sum over  $n$  of  $(\text{if } n = 0 \text{ then } 0 \text{ else } 1) / n^s$  is equal to the sum over  $n \geq 1$  of  $1 / n^s$ .
83 | have h_sum : Σ' n : ℕ, (if n = 0 then 0 else 1) / (n : ℂ) ^ s = Σ' n : ℕ, (1 : ℂ) / (n : ℂ) ^ s := by
84 | | rw [ tsum_congr ];
85 | | rw [ Complex.zero_cpow (by rintro rfl; norm_num at hs) ];
86 | convert h_sum using 1;
87 | · exact tsum_congr fun n => by aesop;
88 | · aesop
89 -/
90 /-
91 The L-series of  $(\log n)/n$  at  $s$  is equal to  $-\zeta'(s+1)$ .
92 Proof idea:
93 1.  $L((\log n)/n, s) = \sum (\log n)/n * n^{s-1} = \sum \log n * n^{s-1}$ .
94 2.  $\zeta'(s+1) = -\sum \log n * n^{s-1}$  (derivative of Dirichlet series).
95 3. So  $-\zeta'(s+1) = \sum \log n * n^{s-1}$ .
96 4. They match.
97 Use Lseries.deriv and the fact that zeta is LSeries of 1.
98 -/
99 lemma LSeries_log_div_n_eq_neg_deriv_zeta (s : ℂ) (hs : 1 < s.re) :
100 LSeries (fun n => if n = 0 then 0 else (Complex.log n) / n) s = - deriv riemannZeta (s + 1) := by
101 | have h1 : deriv riemannZeta (s + 1) = -LSeries (LSeries.logMul (fun n => if n = 0 then 0 else 1)) (s + 1) := by
102 | -- Apply the formula for the derivative of the L-series of a function.
103 | have h2 : 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
104 | -- Apply the formula that the derivative of the L-series is the sum of the derivatives of each term.
105 | have h3 : 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
106 | have h_abs_conv : LSeries.abscissaOfAbsConv (fun n => if n = 0 then 0 else 1) < (s + 1).re := by
107 | refine' lt_of_le_of_lt (csInfl_le _ (2 : ℝ, rfl)) _ <=> norm_num;
108 | -- The series  $\sum_{n=1}^{\infty} n^{s-1}$  is a p-series with  $p=2$ , which converges.
109 | have h_pseries : Summable (fun n => (n : ℂ)^{s-1}) := by
110 | | exact Summable.of_norm <| by simp;
111 | | rw [ LSeriesSummable ];
112 | | convert h_pseries using 1;
113 | | ext _ _ <=> norm_num [ LSeries.term ];
114 | | . exact_mod_cast (by linarith : (2 : ℝ) < s.re + 1)
115 | exact;
116 | exact h_deriv;
117 | convert h_deriv using 1;
118 | refine' Filter.EventuallyEq.deriv_eq _;
119 | filter_upwards [ isOpen.mem_nnhs (isOpen_lt continuous_const Complex.continuous_re) (show 1 < (s + 1) |> Complex.re) by norm_num; linarith ];
120 simp_all +decide [ LSeries, LSeries.logMul ];
121 -- Since the terms of the two series are identical, their sums must be equal.
122 simp [ LSeries.term, LSeries.logMul ];
123 refine' tsum_congr fun n => _ ; aesop;
124 rw [ Complex.cpow_add _ _ ] <=> norm_num [ h ] ; ring
125
```

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

Mode

[1] Fill sorries in a lean file (.lean)

Ctrl+S to submit ↵

Enter the path to your Lean file:

SpherePacking/MagicFunction/PolyFourierCoeffBound.lean█

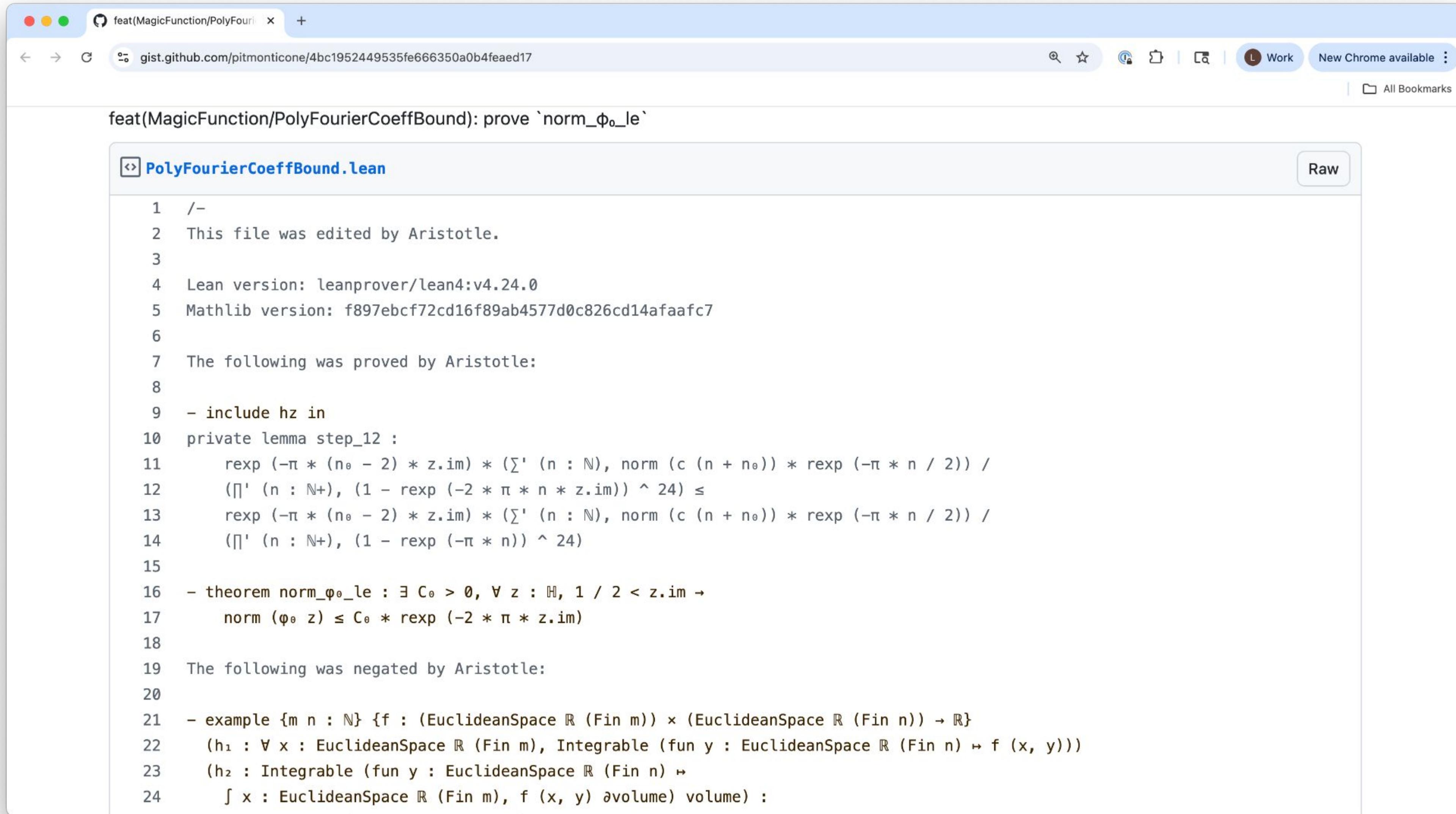
Suggestions:

PolyFourierCoeffBound.lean

PolyFourierCoeffBound_aristotle.lean

Esc to return to main menu ↵

Sorry filling mode - real example from the SpherePackingProject



A screenshot of a web browser window displaying a Lean code file. The browser interface includes a title bar with a logo, a tab labeled 'feat(MagicFunction/PolyFourierCoeffBound)', a back/forward button, a search bar, and a bookmarks bar. The main content area shows the code in a monospaced font.

The code is a Lean file named `PolyFourierCoeffBound.lean`. It contains comments and a private lemma `step_12` that proves a bound involving exponential and summation expressions. It also defines a theorem `norm_phi_le` and provides a counterexample related to Euclidean spaces and integrability.

```
feat(MagicFunction/PolyFourierCoeffBound): prove `norm_phi_le`  
  
PolyFourierCoeffBound.lean  
Raw  
  
1 /-  
2 This file was edited by Aristotle.  
3  
4 Lean version: leanprover/lean4:v4.24.0  
5 Mathlib version: f897ebcf72cd16f89ab4577d0c826cd14afaafc7  
6  
7 The following was proved by Aristotle:  
8  
9 - include hz in  
10 private lemma step_12 :  
11   rexp (-π * (n₀ - 2) * z.im) * (∑' (n : ℕ), norm (c (n + n₀)) * rexp (-π * n / 2)) /  
12   (Π' (n : ℕ+), (1 - rexp (-2 * π * n * z.im)) ^ 24) ≤  
13   rexp (-π * (n₀ - 2) * z.im) * (∑' (n : ℕ), norm (c (n + n₀)) * rexp (-π * n / 2)) /  
14   (Π' (n : ℕ+), (1 - rexp (-π * n)) ^ 24)  
15  
16 - theorem norm_phi_le : ∃ C₀ > 0, ∀ z : ℍ, 1 / 2 < z.im →  
17   norm (φ₀ z) ≤ C₀ * rexp (-2 * π * z.im)  
18  
19 The following was negated by Aristotle:  
20  
21 - example {m n : ℕ} {f : (EuclideanSpace ℝ (Fin m)) × (EuclideanSpace ℝ (Fin n)) → ℝ}  
22   (h₁ : ∀ x : EuclideanSpace ℝ (Fin m), Integrable (fun y : EuclideanSpace ℝ (Fin n) ↦ f (x, y)))  
23   (h₂ : Integrable (fun y : EuclideanSpace ℝ (Fin n) ↦  
24     ∫ x : EuclideanSpace ℝ (Fin m), f (x, y) ∂volume) volume) :
```

Directing Aristotle in English



```
1 /-
2 This file was generated by Aristotle.
3 Lean Toolchain version: leanprover/lean4:v4.20.0-rc5
4 Mathlib version: d62eab0cc36ea522904895389c301cf8d844fd69 (May 9, 2025)
5 -/
6 -/
7 This module implements Newton's method for root finding and proves its local convergence.
8 It defines `newton_step` and `newton_seq`, and proves `newton_convergence_local`, which states that
9 if a function $$ is  $C^2$ ,  $f(x^*) = 0$ , and  $f'(x^*) \neq 0$ , then for any initial guess  $x_0$ 
10 sufficiently close to  $x^*$ , the Newton sequence converges to  $x^*$ .
11 -/
12
13 noncomputable section
14 def newton_step (f f' : R → R) (x : R) : R := x - f x / f' x
15 def newton_seq (f f' : R → R) (x0 : R) : N → R
16 | 0 => x0
17 | n + 1 => newton_step f f' (newton_seq f f' x0 n)
18 |
19 theorem newton_convergence_local
20  (f f' : R → R) (x_star : R)
21  (h_root : f x_star = 0)
22  (h_deriv : ∀ x, HasDerivAt f (f' x) x)
23  (h_c2 : ContDiff R 2 f)
24  (h_f'_ne_0 : f' x_star ≠ 0) :
25  ∃ δ > 0, ∀ x0, |x0 - x_star| < δ →
26  Filter.Tendsto (newton_seq f f' x0) Filter.atTop (nhds x_star) := by
27  -- We'll use the fact that if the derivative is non-zero and continuous, then the Newton step is a contraction mapping.
28  obtain ⟨δ1, hδ1_pos, hδ1⟩ : ∃ δ1 > 0, ∀ x, abs (x - x_star) < δ1 → f' x ≠ 0 := by
29  -- Since f' is continuous at x* and f'(x*) ≠ 0, there exists a δ such that for all x with |x - x*/δ| <
30  |δ1|, |f'(x) - f'(x*)| < |f'(x*)|.
31  have h_cont : ContinuousAt f' x_star := by
32  | rw [ show f' = fun x => deriv f x from funext fun x => HasDerivAt.deriv (h_deriv x) ▷ rfl ] ; exact h_c2.continuous_deriv (by
33  norm_num) |> Continuous.continuousAt;
34  exact Metric.mem_nhds_iff.mp (h_cont.eventually_ne h_f'_ne_0)
35  have h_cont : ContinuousOn (fun x => f x / f' x) (Metric.ball x_star δ1) := by
36  -- The function f(x)/f'(x) is continuous on Metric.ball x_star δ1 since it is the composition of continuous functions.
37  have h_f_cont : Continuous f (Metric.ball x_star δ1) := by
38  | exact h_c2.continuous.continuousOn;
39  refine' h_f_cont.div _ ;
40  . exact ContinuousOn.congr ( show ContinuousOn ( fun x => deriv f x ) ( Metric.ball x_star δ1 ) from h_c2.continuous_deriv (by
41  norm_num) |> Continuous.continuousOn ) fun x hx => HasDerivAt.deriv (h_deriv x) ▷ rfl;
42  . aesop
43  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
44  have h_deriv_zero : HasDerivAt (fun x => x - f x / f' x) 0 x_star := by
45  convert HasDerivAt.sub (hasDerivAt_id x_star) (HasDerivAt.div (h_deriv x_star) (hasDerivAt_deriv_iff.mpr _)) using 1 <>;
46  aesop;
47  · rw [ sq, mul_div_mul_right ] <;> aesop;
48  · have h_diff : ContDiff R 1 f' := by
49  | have h_diff : ContDiff R (1 + 1) f := by
50  | exact h_c2;
51  | rw [ show f' = fun x => deriv f x from funext fun x => HasDerivAt.deriv (h_deriv x) ▷ rfl ] ; erw [ contDiff_succ_iff_deriv
52  ] at h_diff ; tauto;
53  | exact h_diff.contDiffAt.differentiableAt (by norm_num);
54  have := Metric.tendsto_nhds_nhds.1 h_deriv.zero.tendsto_div_nhds_zero; aesop;
55  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,
56  h_root ] else have := H hx; rw [ div_lt_iff, (abs_pos.mpr (sub_ne_zero.mpr h)) ] at this; linarith⟩;
57  -- By induction, we can show that for all n, the nth term of the sequence is within δ of x_star.
58  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
59  intro x0 hx0 n; induction n <=> simp_all +decide [ pow_succ', mul_assoc ];
60  · rfl;
61  · convert le_trans (hδ _ _) (mul_le_of_nonneg_left _ (by norm_num)) using 1;
62  | exact lt_le_of_lt (by exact lt_of_le_of_lt (mul_le_of_le_one_left (abs_nonneg _)) (inv_le_one_of_le_one_le _)) hx0 );
63  | exact (δ, hδ_pos, fun x0 hx0 => tendsto_iff_norm_sub_tendsto_zero.mpr <| squeeze_zero (fun _ => norm_nonneg _ ) (fun n =>
```

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.

Directing Aristotle in English - Giving Aristotle access to your library



MyLibrary.lean:

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

description of Newton Iteration

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.

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

Mode — [3] Direct Aristotle in English (.lean optional)

Ctrl+T=text, Ctrl+L=lean, Ctrl+F=files, Ctrl+S=submit —

Enter your prompt:

1. The variant of praton.tex's main result (the "THEOREM") for g instead of f.
2. The base case ($c=0$ in Praton's language), which is proven in baek.tex

Optional: Attach a Lean file as context

baek.lean

Optional: Attach other context - file or folder

baek/

✓ Folder

Suggestions:

baek.tex praton.tex

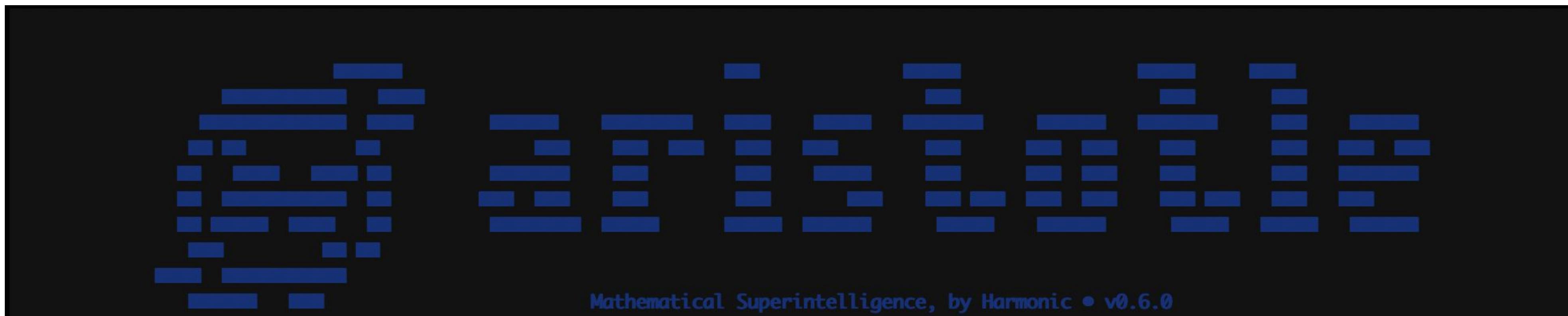
Esc to return to main menu —

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

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

859 theorem base_case (k : N) (hk : k > 0) : g (k ^ 2 + 1) = k := by
860   simp +zetaDelta at *;
861   by_contra h_contra;
862   -- Since $g$ is the supremum of total side lengths, if $g > k$, there exists a packing $P$ such that $P.total_side_length > k$.
863   obtain ⟨P, hP_valid, hP_len⟩ : ∃ P : Packing (k ^ 2 + 1), P.is_valid ∧ P.total_side_length > k := by
864     by_cases h : g (k ^ 2 + 1) > k;
865     · have := exists_lt_of_lt_csSup ?_ h; aesop;
866       exact ⟨_, ⟨ fun _ => ⟨ 0, 0, 0, by norm_num, by norm_num, by norm_num, by norm_num ⟩, fun _ _ _ => by norm_num [ Square.disjoint ], rfl ⟩⟩;
867       · exact False.elim <| h_contra <| le_antisymm (le_of_not_gt h) <| base_case_lower k hk |> le_trans (by norm_num);
868     -- By `exists_shift_ho`, there exists a shift `a` in `[0, 1/k)` such that `Σ i, count_intersections_ho k a (P i) ≥ k^2 + 1`.
869   obtain ⟨a, ha0, ha1⟩ : ∃ a, 0 ≤ a ∧ a < 1 / (k : R) ∧ Σ i, count_intersections_ho k a (P i) ≥ k * k + 1 := by
870     field_simp;
871     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_cancel 1 (by positivity : (k : R) ≠ 0), ha.2.1 ], ha.2.2 ⟩;
872   -- Let `m i = count_intersections_ho k a (P i)` and `d i = (P i).s`.
873   set m : Fin (k ^ 2 + 1) → N := fun i => count_intersections_ho k a (P i)
874   set d : Fin (k ^ 2 + 1) → R := fun i => (P i).s;
875   -- We verify the conditions for `baek_combinatorial`:
876   have h_sum_m : Σ i, m i ≥ k * k + 1 := by
877     exact ha1.2
878   have h_sum_m_d : Σ i, m i * d i ≤ k := by
879     convert sum_intersections_le_k_ho k hk P hP_valid a < ha0, ha1.1 > using 1
880   have h_m_zero : ∀ i, m i = 0 → d i ≤ 1 / k := by
881     bound;
882     convert small_squares_miss_lines_ho k hk a (P i) < ha0, left > a_1 using 1
883   have h_m_two : ∀ i, m i ≥ 2 → d i ≥ 1 / k := by
884     intros i hi; exact (by
885       apply large_squares_hit_lines_ho k hk a (P i) < ha0, ha1.left > hi);
886   -- Applying `baek_combinatorial`, we get `Σ d i ≤ k`.
887   have h_sum_d : Σ i, d i ≤ k := by
888     apply baek_combinatorial;
889     any_goals tauto;
890     ring;
891   exact hP_len.not_le h_sum_d
892 
893 /-
894 For all integers c, g(k^2 + 2c + 1) = k + c/k for all k > |c|.
895 -/
896 theorem main_theorem (c : Z) : Q c := by
897   apply proton_variant_proven;
898   use 0;
899   unfold Q;
900   aesop;
901   cases k <;> aesop;
902   convert base_case a_1 a using 1
22233a27-a0b5-463d-9f7d-7d1e52d1388b-output.txt
```

Monitoring and Accessing Previous Results



Mathematical Superintelligence, by Harmonic • v0.6.0

> View history
View recent proof attempts

Recent proof attempts:

| # | Project | Last Updated | Status | Created |
|-----|------------------------------|--------------|---------------|------------|
| 1 | PolyFourierCoeffBound.lean | 1 hour ago | ✓ COMPLETE | 1 hour ago |
| 2 | Generalize Fermat's Last ... | 1 week ago | ✓ COMPLETE | 1 week ago |
| 3 | Fermat's Last Theorem for... | 1 week ago | ✓ COMPLETE | 1 week ago |
| 4 | Write Fermat's Last Theor... | 1 week ago | ✓ COMPLETE | 1 week ago |
| 5 | 13c6183a-bad... | 1 week ago | ● NOT_STARTED | 1 week ago |
| 6 | 690cf29-3f0... | 1 week ago | ● NOT_STARTED | 1 week ago |
| 7 | 81ec53f4-e6f... | 1 week ago | ● NOT_STARTED | 1 week ago |
| 8 | prompt.txt | 1 week ago | ✓ COMPLETE | 1 week ago |
| ▼ 9 | prompt.txt | 1 week ago | ✓ COMPLETE | 1 week ago |

Row 1/10 | ↑↓ or j/k: navigate | Enter: select | m: load more | Esc/b: back to menu