

Tacq - Context Aware Tactic Recommendation for Rocq

JFLA 2026

Jules Viennot, Guillaume Baudart, Emilio Jesùs Gallego Arias, Marc Lelarge, et Théo Stoskopf

Recent outbreak of Reasoning models



July 21, 2025 Research

Advanced version of Gemini with Deep Think officially achieves **gold-medal** standard at the International Mathematical Olympiad

Thang Luong and Edward Lockhart

Share 



OpenAI 

@OpenAI



We achieved **gold medal-level** performance  on the 2025 International Mathematical Olympiad with a general-purpose reasoning LLM!

Our model solved world-class math problems—at the level of top human contestants. A major milestone for AI and mathematics.



PROBLEM 5 SUBMISSION

...

1. Some elementary facts.

A function on $[0, \sqrt{2}]$:

$f(t) := t + \sqrt{2 - t^2}$, domain $t \in [0, \sqrt{2}]$.

Identity: $(f(t))^2 = t^2 + 2 - t^2 + 2t\sqrt{2 - t^2} = 2 + 2t\sqrt{2 - t^2} \geq 2$ ($t \geq 0$). So $f(t) \geq \sqrt{2}$. So:

(**) For $0 \leq t \leq \sqrt{2}$: $t + \sqrt{2 - t^2} \geq \sqrt{2}$.
Equality at $t=0$, $t=\sqrt{2}$.

So any pair (a,b) with $a^2 + b^2 = 2$ and a, b nonneg has $\text{sum} \geq \sqrt{2}$.

...

July 21, 2025 Research

Advanced version of Gemini with Deep Think officially achieves **gold-medal** standard at the International Mathematical Olympiad

Thang Luong and Edward Lockhart

Share 

Natural language



human verification for proofs



OpenAI 

@OpenAI



We achieved **gold medal-level** performance  on the 2025 International Mathematical Olympiad with a general-purpose reasoning LLM!

Our model solved world-class math problems—at the level of top human contestants. A major milestone for AI and mathematics.



PROBLEM 5 SUBMISSION

...

1. Some elementary facts.

A function on $[0, \sqrt{2}]$:

$f(t) := t + \sqrt{2 - t^2}$, domain $t \in [0, \sqrt{2}]$.

Identity: $(f(t))^2 = t^2 + 2 - t^2 + 2t\sqrt{2 - t^2} = 2 + 2t\sqrt{2 - t^2} \geq 2$ ($t \geq 0$). So $f(t) \geq \sqrt{2}$. So:

(**) For $0 \leq t \leq \sqrt{2}$: $t + \sqrt{2 - t^2} \geq \sqrt{2}$.
Equality at $t=0$, $t=\sqrt{2}$.

So any pair (a,b) with $a^2 + b^2 = 2$ and a, b nonneg has $\text{sum} \geq \sqrt{2}$.

...

July 21, 2025 Research

Advanced version of Gemini with Deep Think officially achieves **gold-medal** standard at the International Mathematical Olympiad

Thang Luong and Edward Lockhart

Share 

Natural language



human verification for proofs



doesn't scale up



OpenAI 

@OpenAI

...

We achieved **gold medal-level** performance  on the 2025 International Mathematical Olympiad with a general-purpose reasoning LLM!

Our model solved world-class math problems—at the level of top human contestants. A major milestone for AI and mathematics.



PROBLEM 5 SUBMISSION

...

1. Some elementary facts.

A function on $[0, \sqrt{2}]$:

$f(t) := t + \sqrt{2 - t^2}$, domain $t \in [0, \sqrt{2}]$.

Identity: $(f(t))^2 = t^2 + 2 - t^2 + 2t\sqrt{2 - t^2} = 2 + 2t\sqrt{2 - t^2} \geq 2$ ($t \geq 0$). So $f(t) \geq \sqrt{2}$. So:

(**) For $0 \leq t \leq \sqrt{2}$: $t + \sqrt{2 - t^2} \geq \sqrt{2}$. Equality at $t=0$, $t=\sqrt{2}$.

So any pair (a,b) with $a^2 + b^2 = 2$ and a, b nonneg has $\text{sum} \geq \sqrt{2}$.

...



SCIENCE

AI achieves silver-medal standard solving International Mathematical Olympiad problems

25 JULY 2024
AlphaProof and AlphaGeometry teams

Share



- News
- About
- Careers

Aristotle Achieves Gold Medal-Level Performance at the International Mathematical Olympiad, iOS App Beta Launch 07.28.2025

Today we announced live on X that our advanced formal reasoning model Aristotle achieved Gold Medal-level performance at the 2025 International Mathematical Olympiad, widely regarded as the world's most prestigious mathematics competition. We are releasing Aristotle's full proofs, which critically did not require human checking since they were formally verified, publicly on Github. This result places Aristotle at the forefront of advanced mathematical reasoning performance among frontier AI models.

As the world is witnessing an explosion in AI-generated content, from news articles to computer code, an unintended side-effect of this has been to turn the people using these AI tools from creators into human verifiers. In technical fields like software engineering, this means engineers spend less time coding and more time confirming the correctness of AI-generated code. We call this the verification problem, and believe it is a major bottleneck in scaling the usefulness of AI tools.

Aristotle overcomes this by design. Every solution it generates is formally verified down to foundational axioms using the Lean4 proof assistant, providing a machine-checkable guarantee of correctness. We're starting to deploy this technology with mathematics, and we're already working on expanding support to dependent disciplines such as physics and software engineering.

As an early look into what we expect this new verified AI technology to look like, we are launching Aristotle for iOS in Beta. When someone asks Aristotle a complex question, they not only see the solution in English but also the corresponding Lean code that verifies the correctness. Aristotle for iOS also supports photo-based solving and solving multiple questions in parallel, all with just a couple taps. Users can sign up for our waitlist at aristotle.harmonic.fun now, with the rollout already underway.

[Live Announcement on X](#)
[Aristotle for iOS](#)

ByteDance | Seed

EN



ByteDance Seed Prover Achieves Silver Medal Score in IMO 2025

Date
2025-07-23

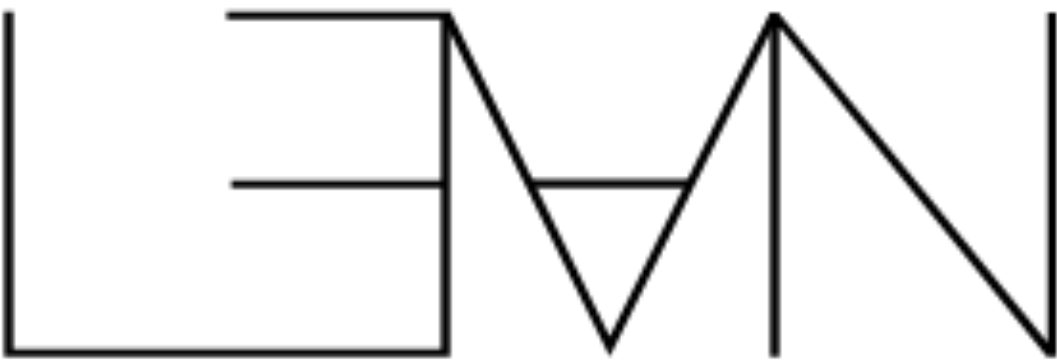
Category

Team News

By official invitation, the ByteDance Seed team participated in IMO (The International Mathematical Olympiad) 2025, deploying our specialized formal mathematical reasoning model, Seed Prover. After three days of sustained effort, Seed Prover fully solved 4 out of 6 problems and partially solved 1, receiving an IMO-certified score of 30 points, equivalent to a silver medal performance (scores per problem: 2/7/7/7/7/0). Our subsequent extended search efforts resulted in the successful completion of proofs for the first five problems.

Seed Prover is constructed upon the Lean proof assistant and trained using multi-stage reinforcement learning. It guarantees the absolute reliability of mathematical proofs by facilitating interaction between natural language reasoning and formally verified code. Through extended chain-of-thought reasoning and computational scaling via multi-agent interaction, Seed Prover is capable of deeply analyzing individual mathematical problems continuously over several days.

Breakthrough models AlphaProof and AlphaGeometry 2 solve advanced reasoning problems in mathematics

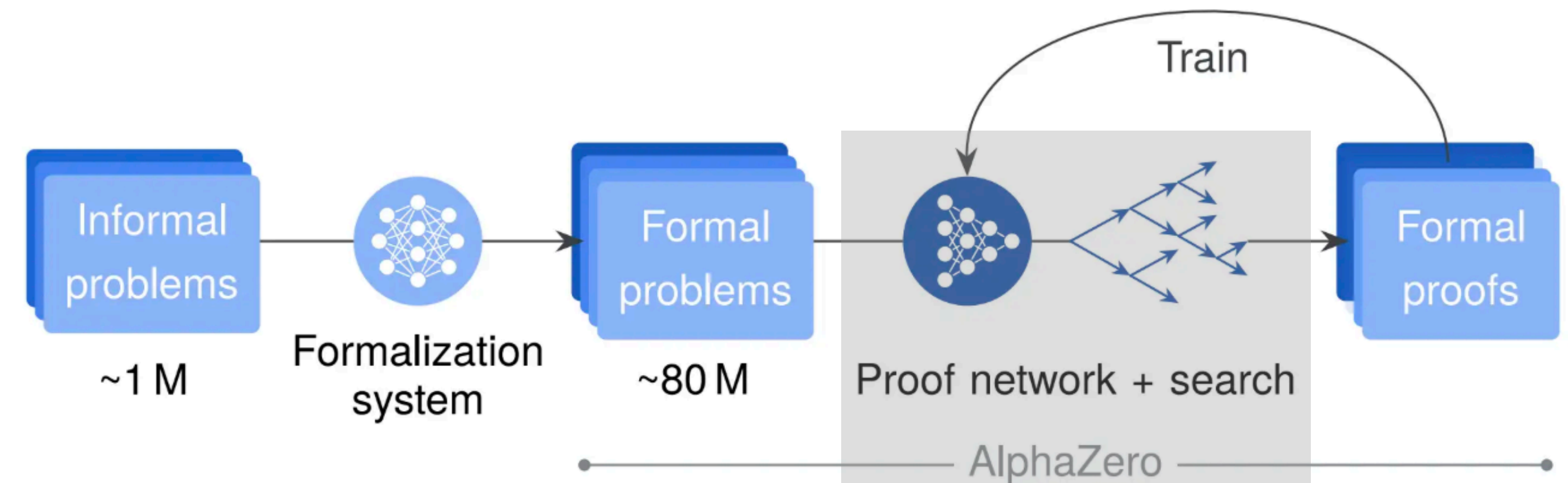


proof assistant

Why Tacq?

Motivations:

- Opaque models: what is important to them?



« ... by searching over possible proof steps in Lean ... »

→ More details **one year later**

Motivations:

- Opaque models: what is important to them?
- Useful for Rocq users

```

/-
A line in the plane is called sunny if it is not parallel to any of the $x$-axis, the $y$-axis, and the line $x+y=0$.
Let $n \geqslant 3$ be a given integer. Determine all nonnegative integers $k$ such that there exist $n$ distinct lines in the plane satisfying both of the following:
- for all positive integers $a$ and $b$ with $a+b \leqslant n+1$, the point $(a, b)$ is on at least one of the lines; and
- exactly $k$ of the $n$ lines are sunny.

Answer: 0, 1, 3
-/

import HarmonicLean.Imports

open scoped BigOperators
open scoped Real
open scoped Nat
open scoped Classical
open scoped Pointwise

set_option maxHeartbeats 0
set_option maxRecDepth 4000
set_option synthInstance.maxHeartbeats 20000
set_option synthInstance.maxSize 128

set_option pp.fullNames true
set_option pp.structureInstances true

set_option relaxedAutoImplicit false
set_option autoImplicit false

set_option pp.coercions.types true
set_option pp.funBinderTypes true
set_option pp.letVarTypes true
set_option pp.piBinderTypes true

set_option linter.all false

noncomputable section

namespace IM02025P1

/-
A line in the plane is called sunny if it is not parallel to the $x$-axis, the $y$-axis, or the line $x+y=0$. A line is non-sunny otherwise.
-/
def Sunny (l : Set (ℝ × ℝ)) : Prop :=
  ∃ a b c, l = { v | a * v.1 + b * v.2 + c = 0 } ∧ a ≠ 0 ∧ b ≠ 0 ∧ a ≠ b

def NonSunny (l : Set (ℝ × ℝ)) : Prop :=
  (∃ a b c, ¬ (a = 0 ∧ b = 0) ∧ l = { v | a * v.1 + b * v.2 + c = 0 }) ∧ ¬ Sunny l

/-
For a positive integer $n$, define the set of points $S_n = \{(a,b) \in \mathbb{Z}^+ \times \mathbb{Z}^+ : a+b \leq n+1\}$.
-/
def s (n : ℕ) : Set (ℝ × ℝ) :=
  { v | ∃ a b : ℕ, 0 < a ∧ 0 < b ∧ a + b ≤ n + 1 ∧ v = ((a : ℝ), (b : ℝ)) }

/-
A line is non-sunny if and only if its equation can be written in the form $x=c$, $y=c$, or $x+y=c$ for some constant $c$.
-
-/
lemma lem_non_sunny_form (l : Set (ℝ × ℝ))
  (h_is_line : ∃ a b c, ¬ (a = 0 ∧ b = 0) ∧ l = { v | a * v.1 + b * v.2 + c = 0 }) :
  NonSunny l ↔ (∃ c : ℝ, l = { v | v.1 = c }) ∨
    (∃ c : ℝ, l = { v | v.2 = c }) ∨
    (∃ c : ℝ, l = { v | v.1 + v.2 = c }) := by
  bound;
  · have := a;
    -- By definition of non-sunny, if a line is non-sunny, then it is not sunny, which means it is parallel to one of the axes or the line x+y=0.
    have h_parallel : ¬Sunny {v : ℝ × ℝ | w * v.1 + w_1 * v.2 + w_2 = 0} → (w = 0 ∨ w_1 = 0 ∨ w = w_1) := by
      -- By definition of sunny, if a line is not sunny, then it must be parallel to one of the axes or the line x+y=0.
      intro h_not_sunny
      by_contra h_contra;
      exact h_not_sunny { w, w_1, w_2, rfl, by tauto };
    rcases h_parallel this.2 with ( rfl | rfl | rfl );
    · exact 0r.inr <| 0r.inl ( -w_2 / w_1, by ext; simp ( config := { decide := Bool.true } ) [ field_simps, show w_1 ≠ 0 by aesop ] ; constructor <=> li
    · exact 0r.inl ( -w_2 / w, by ext; simp ( config := { decide := Bool.true } ) [ field_simps, show w ≠ 0 by aesop ] ; constructor <=> intros <=> li
    · exact 0r.inr <| 0r.inr <| ( -w_2 / w, by ext; simp ( config := { decide := Bool.true } ) [ field_simps, show w ≠ 0 by aesop ] ; constructor <=> li
  · -- If the line is of the form $x = c$, then it is parallel to the $y$-axis and thus not sunny.
    have h_not_sunny : ¬Sunny {v : ℝ × ℝ | v.1 = w_3} := by
      rintro ( a, b, c, h₁, h₂, h₃, h₄ );
      norm_num [ Set.Subset.antisymm_iff, Set.subset_def ] at h₁;
      exact h₃ ( by linarith [ h₁.1 w_3 0 rfl, h₁.1 w_3 1 rfl ] );
    -- Since the line $x = w_3$ is not sunny, we can conclude that it is non-sunny.
    apply And.intro;
    · exact ( w, w_1, w_2, left, rfl );
    · -- Since the line $x = w_3$ is not sunny, we can conclude that it is non-sunny by definition.

```

Motivations:

- Opaque models: what is important to them?
- Useful for Rocq users

```
- /- For any integer $n$ (ge 3$), it is impossible to have a configuration with $k=2$ sunny lines. -/
lemma lem_k2_impossible (n : ℕ) (hn : n ≥ 3) :
- (∃ lines : Finset (Set (ℝ × ℝ)),
  (∀ line ∈ lines, ∃ a b c, ¬ (a = 0 ∧ b = 0) ∧ line = { v | a * v.1 + b * v.2 + c = 0 }) ∧
  lines.card = n ∧
  (∀ p ∈ s n, ∃ l ∈ lines, p ∈ l) ∧
  (lines.filter Sunny).card = 2) := by
  -- Assume for contradiction that there exists a configuration with $k=2$ sunny lines.
  by_contra h_contra;
  -- By Lemma~\ref{lem:reduction}, if there exists a configuration with $k=2$ for $n$ \geq 4$, then there exists a configuration with $k=2$ for $n-1$.
  have h_reduction : ∀ n ≥ 4, (∃ lines : Finset (Set (ℝ × ℝ)), (∀ line ∈ lines, ∃ a b c, ¬ (a = 0 ∧ b = 0) ∧ line = {v : ℝ × ℝ | a * v.1 + b * v.2 + c = 0}) ∧ line
  -- Apply Lemma~\ref{lem:reduction} to the given configuration for $n$ \geq 4$.
  intros n hn h_config
  obtain (lines, hlines) := h_config;
  by_cases h_boundary : {v : ℝ × ℝ | v.1 = (1:ℝ)} ∈ lines v {v : ℝ × ℝ | v.2 = (1:ℝ)} ∈ lines v {v : ℝ × ℝ | v.1 + v.2 = (n + 1 : ℝ)} ∈ lines;
  · have := lem_reduction n 2 ( by linarith );
    exact this ( lines, hlines.1, hlines.2.1, hlines.2.2.1, hlines.2.2.2, h_boundary );
  · have := lem_boundary_lines_implication n ( by linarith ) lines;
    simp_all ( config := { decide := Bool.true } );
    exact absurd hlines.2.2.2 ( by rw [ Finset.filter_true_of_mem this ] ; linarith );
  -- By repeatedly applying the reduction lemma, we can reduce the problem to the case where $n=3$.
  have h_reduction_to_3 : ∀ n ≥ 4, (∃ lines : Finset (Set (ℝ × ℝ)), (∀ line ∈ lines, ∃ a b c, ¬ (a = 0 ∧ b = 0) ∧ line = {v : ℝ × ℝ | a * v.1 + b * v.2 + c = 0}) ∧
  intro n hn h;
  induction' hn with n hn ih;
  · exact h_reduction 4 ( by norm_num ) h;
  · exact ih ( h_reduction _ ( Nat.le_succ_of_le hn ) h );
  by_cases hn4 : n ≥ 4;
  · exact lem_k2_impossible_n3 ( h_reduction_to_3 n hn4 h_contra );
  · exact lem_k2_impossible_n3 ( by interval_cases n ; exact h_contra )

/-
A line in the plane is called sunny if it is not parallel to any of the $x$-axis, the $y$-axis, and the line $x+y=0$.
Let $n$ \geqslant 3$ be a given integer. Determine all nonnegative integers $k$ such that there exist $n$ distinct lines in the plane satisfying both of the following:
- for all positive integers $a$ and $b$ with $a+b$ \leqslant $n+1$, the point $(a, b)$ is on at least one of the lines; and
- exactly $k$ of the $n$ lines are sunny.
-/
theorem main_theorem
(n : ℕ)
(hn : n ≥ 3) :
{0, 1, 3} = { k | ∃ lines : Finset (Set (ℝ × ℝ)),
  (∀ line ∈ lines, ∃ a b c, ¬ (a = 0 ∧ b = 0) ∧ line = { v : ℝ × ℝ | a * v.1 + b * v.2 + c = 0 }) ∧
  lines.card = n ∧
  (∀ a b : ℕ, 0 < a ∧ 0 < b ∧ a + b ≤ n + 1 → ∃ (l : Set (ℝ × ℝ)), l ∈ lines ∧ ((a : ℝ), (b : ℝ)) ∈ l) ∧
  ((lines.filter (fun l => ∃ a b c, l = { v : ℝ × ℝ | a * v.1 + b * v.2 + c = 0 } ∧ a ≠ 0 ∧ b ≠ 0 ∧ a ≠ b)).card = k) } := by
  ext k;
  -- For the forward direction, if $k$ is 0, 1, or 3, then by the lemmas we have, there exists such a set of lines.
  apply Iff.intro;
  · simp +zetaDelta at *;
    -- For each case, we can apply the corresponding lemma to construct the required set of lines.
    intro hk
    cases' hk with hk0 hk1 hk3;
    · convert lem_k0_possible n hn;
      · aesop;
      · -- By definition of $$$, if $(a, b)$ \in s n$, then there exist $a$ and $b$ such that $0 < a$, $0 < b$, $a + b$ \leq $n + 1$, and $(a, b) = ((a : ℝ), (b : ℝ))$.
        unfold s; aesop;
      · -- By definition of Sunny, if there exist $a, b, c$ such that the line is defined by $a*x + b*y + c = 0$ and $a, b$ are non-zero and not equal, then the line is sunny.
        unfold Sunny; aesop;
    · bound;
      · convert lem_k1_possible n hn;
        · bound;
        · unfold s; aesop;
        · unfold Sunny; aesop;
      · convert lem_k3_possible n hn;
        · bound;
        · unfold s; aesop;
        · unfold Sunny; aesop;
    · -- By combining the results from the lemmas, we can conclude that $k$ must be 0, 1, or 3.
      intros hk
      obtain (lines, hlines) := hk;
      -- By Lemma~\ref{lem:k_ge_4_impossible}, $k$ \leq 3$.
      have hk_le_3 : k ≤ 3 := by
        have := lem_k_ge_4_impossible n k hn;
        contrapose! this;
        simp +zetaDelta at *;
        exact ( this, lines, hlines.1, hlines.2.1, fun a b hab => by rcases hab with ( a', b', ha', hb', hab', rfl, rfl ) ; exact hlines.2.2.1 a' b' ha' hb' hab',
        interval_cases k <=> norm_num;
      -- By Lemma~\ref{lem:k2_impossible}, there cannot be a configuration with $k=2$.
      apply lem_k2_impossible n hn;
      use lines;
      unfold Sunny; aesop;
      cases a_1 ; aesop

#print axioms main_theorem
```

Motivations:

- Opaque models: what is important to them?
- Useful for Rocq users
- Not enough compute power

Days of computing:

- AlphaProof: ~3 days
- ByteDance: more than one day

Big models:

- Aristotle: > 200B parameters
- DeepSeek-Prover-V2: 671B parameters

Motivations:

- Opaque models: what is important to them?
- Useful for Rocq users
- Not enough compute power

Tacq:

- **Experiments** on models and available information
- **Tools:**
 - *Pytanque*: context extraction
 - *LLM4Docq*: automatic natural language annotation
- **Next tactic prediction** pipeline

+ focus on the *MathComp* library

Current goal

```
R : fieldType
h : {poly R}
p : {poly %/ h}
hQM : hQ \is monic
hCp : coprimep hQ p
p_neq0 : p ≠ 0
F : (egcdp hQ p).1 * hQ
    + (egcdp hQ p).2 * p %= 1
```

```
((egcdp hQ p).2 * p) %% hQ %= 1 %% hQ →
(lead_coef (((egcdp hQ p).2 * p) %% hQ))^-1
 *: (((egcdp hQ p).2 * p) %% hQ)
 = 1%:P
```

Current goal

```
R : fieldType
h : {poly R}
p : {poly %/ h}
hQM : hQ \is monic
hCp : coprimep hQ p
p_neq0 : p ≠ 0
F : (egcdp hQ p).1 * hQ
    + (egcdp hQ p).2 * p %= 1
```

```
((egcdp hQ p).2 * p) %% hQ %= 1 %% hQ →
(lead_coef (((egcdp hQ p).2 * p) %% hQ))^-1
 *: (((egcdp hQ p).2 * p) %% hQ)
 = 1%:P
```



Dependencies

- `monic := [qualify p | monic_pred p]`
- `coprimep p q := size (gcdp p q) = 1%N`
- `egcdp p q := if size q ≤ size p then [ ... ]`
- `lead_coef p := p`_(size p).-1`
- `hQ := (mk_monic h).`

Notations

- `"{ 'poly' T }" := (polynomial T)`
- `"{ 'poly' '%/' p }" := (qpoly p)`
- `"c %:P" := (polyC c)`

Current goal

```
R : fieldType
h : {poly R}
p : {poly %/ h}
hQM : hQ \is monic
hCp : coprimep hQ p
p_neq0 : p ≠ 0
F : (egcdp hQ p).1 * hQ
    + (egcdp hQ p).2 * p %= 1
```

```
((egcdp hQ p).2 * p) %% hQ %= 1 %% hQ →
(lead_coef (((egcdp hQ p).2 * p) %% hQ))^-1
 *: (((egcdp hQ p).2 * p) %% hQ)
 = 1%:P
```

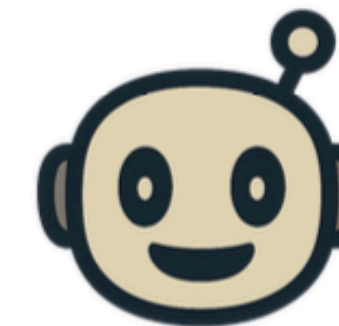


Dependencies

- `monic := [qualify p | monic_pred p]`
- `coprimep p q := size (gcdp p q) = 1%N`
- `egcdp p q := if size q ≤ size p then [ ... ]`
- `lead_coef p := p`_(size p).-1`
- `hQ := (mk_monich h).`

Notations

- `"{ 'poly' T }" := (polynomial T)`
- `"{ 'poly' '%/' p }" := (qpoly p)`
- `"c %:P" := (polyC c)`



Natural language docstrings

- Identifies monic polynomials.
 - Tests if polynomials `p` and `q` are coprime.
 - Computes the extended greatest common divisor.
 - Extracts the leading coefficient of `p`.
 - Monic version of polynomial `h`.
-
-
-
-
-
-
-
-
-
-
- Polynomials over type `T`.
 - Quotient polynomials modulo `p`.
 - Lift a scalar `c` to a constant polynomial.

Current goal

```
R : fieldType
h : {poly R}
p : {poly %/ h}
hQM : hQ \is monic
hCp : coprimep hQ p
p_neq0 : p ≠ 0
F : (egcdp hQ p).1 * hQ
    + (egcdp hQ p).2 * p %= 1
```

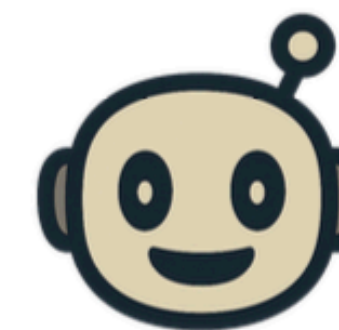
```
((egcdp hQ p).2 * p) %% hQ %= 1 %% hQ →
(lead_coef (((egcdp hQ p).2 * p) %% hQ))^-1
 *: (((egcdp hQ p).2 * p) %% hQ)
 = 1%:P
```

Dependencies

- `monic := [qualify p | monic_pred p]`
- `coprimep p q := size (gcdp p q) = 1%N`
- `egcdp p q := if size q ≤ size p then [ ... ]`
- `lead_coef p := p`_(size p).-1`
- `hQ := (mk_monich h).`

Notations

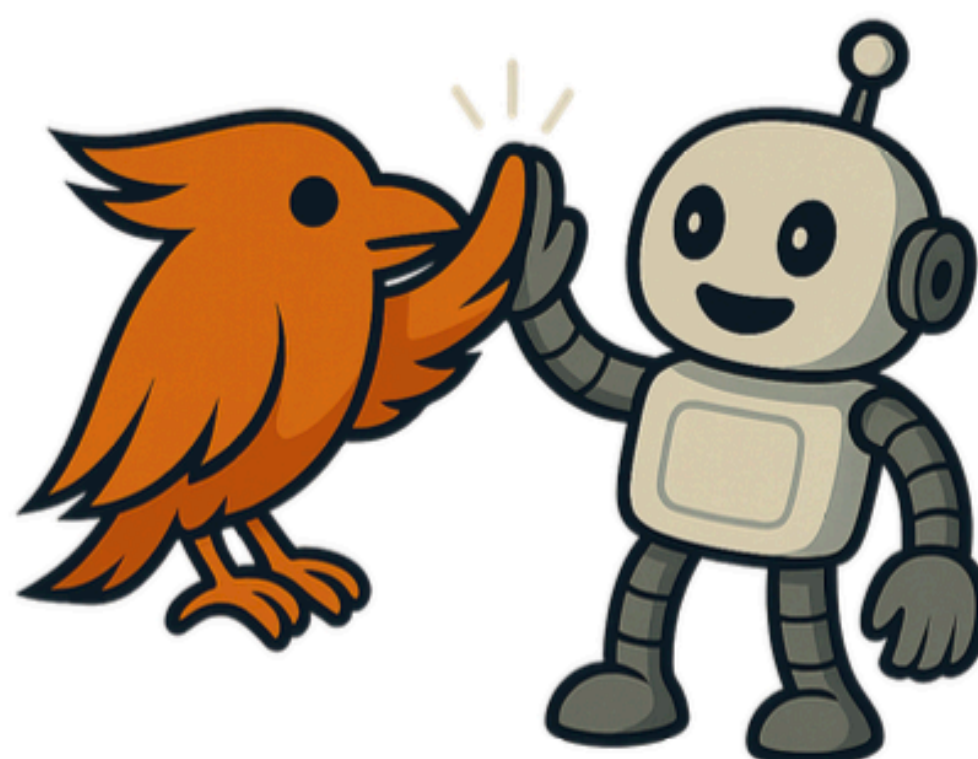
- `"{ 'poly' T }" := (polynomial T)`
- `"{ 'poly' '%/' p }" := (qpoly p)`
- `"c %:P" := (polyC c)`



Natural language docstrings

- Identifies monic polynomials.
 - Tests if polynomials p and q are coprime.
 - Computes the extended greatest common divisor.
 - Extracts the leading coefficient of p.
 - Monic version of polynomial h.
-
- Polynomials over type T.
 - Quotient polynomials modulo p.
 - Lift a scalar c to a constant polynomial.

current goal
+ dependencies
+ notations
+ docstrings



rewrite `[(1 %% _)%R]modp_small ⇒ //`.

Context extraction

Dependencies

Decomposition

Current goal

$n : \text{nat}$

$\text{fact_div} :$

$n \% | \text{factorial } n$

$0 < n$

Dependencies

Decomposition

Current goal

$n : \text{nat}$

$\text{fact_div} :$

$n \% | \text{factorial } n$

$0 < n$

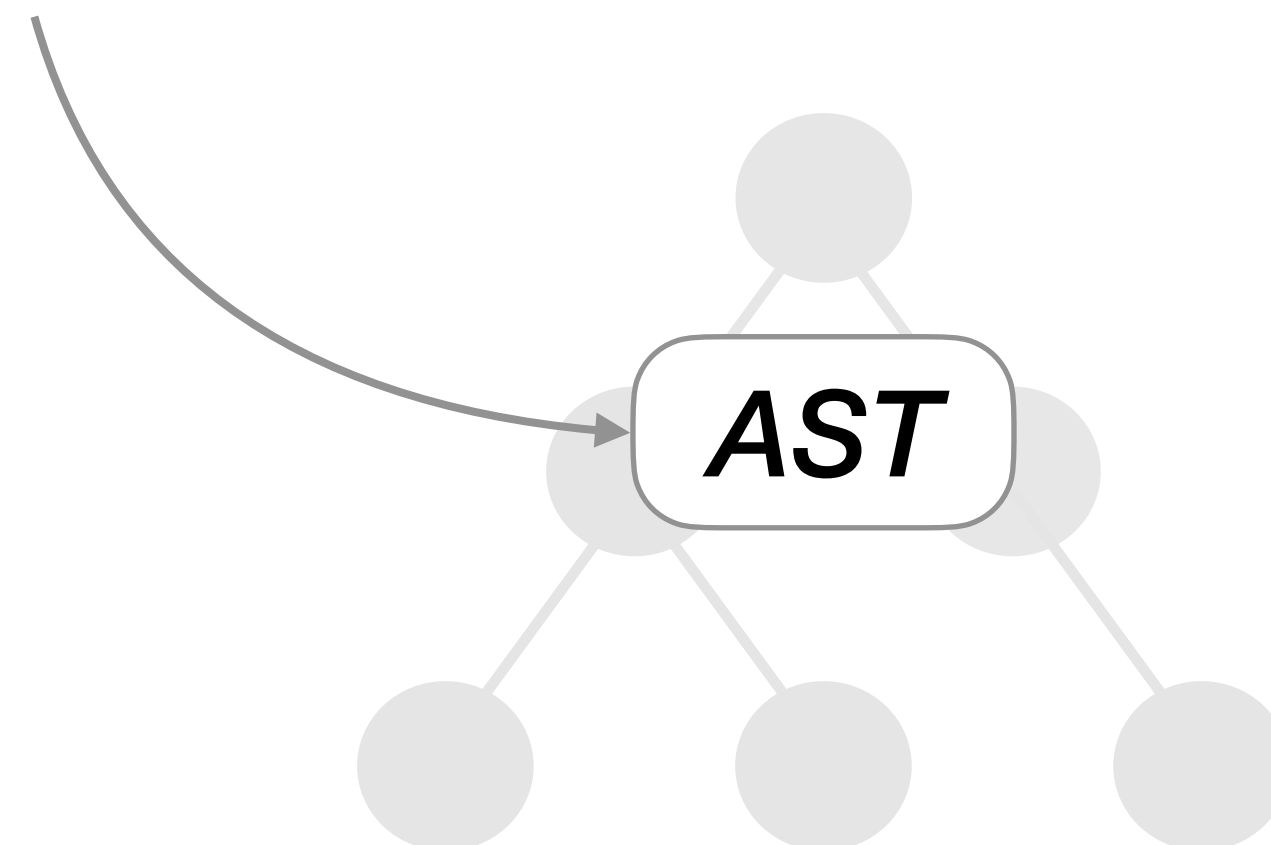
Isolation

Definition $n : \text{nat}.$

Definition $\text{fact_div} :$

$n \% | \text{factorial } n.$

Definition $\text{concl} : 0 < n.$



Dependencies

Decomposition

Current goal

```
n : nat
fact_div :
  n %| factorial n
-----
0 < n
```

Isolation

```
Definition n : nat.
Definition fact_div :
  n %| factorial n.
Definition concl : 0 < n.
```

Dependencies

Extraction

```
[nat]

[n, factorial, n]
[n]
```

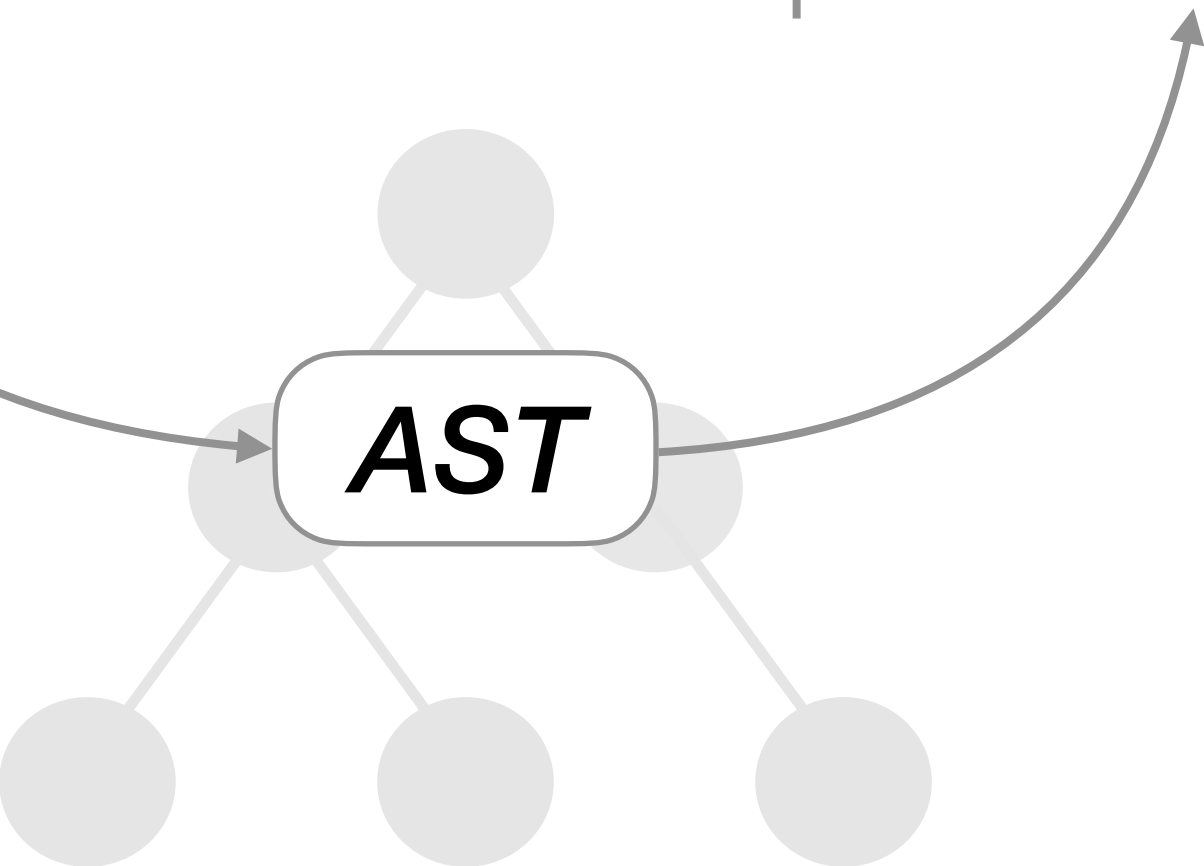
Filtering

```
[nat]

[factorial]
[]
```

Retrieval

```
nat : Set
factorial :
  nat → nat
```



Notations

$G1$ and $G2$ are groups that both have a " $_ + _$ " notation.

Lemma morph_id :

forall $\text{morph} : G1 \xrightarrow{1} G2$,

(**forall** $a\ b$, $\text{morph}\ a \xrightarrow{2} \text{morph}\ b \xrightarrow{3} \text{morph}\ (a \xrightarrow{4} b) \xrightarrow{5}$

$\text{morph}\ \text{id1} \xrightarrow{6} \text{id2}$.)

1. CNotation : " $_ \rightarrow _$ "

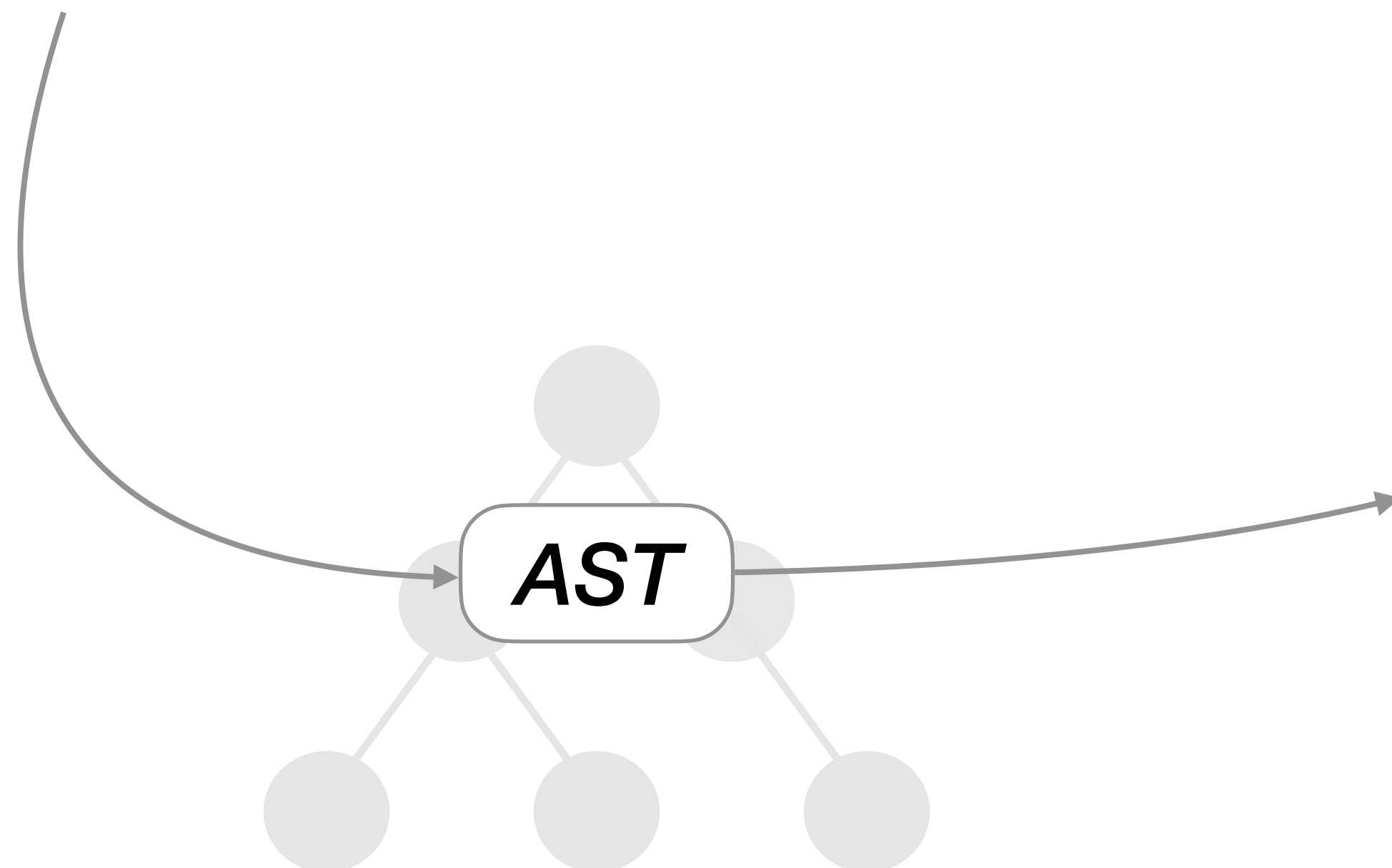
2. CNotation : " $_ + _$ "

3. CNotation : " $_ = _$ "

4. CNotation : " $_ + _$ "

5. CNotation : " $_ \rightarrow _$ "

6. CNotation : " $_ = _$ "



Notations

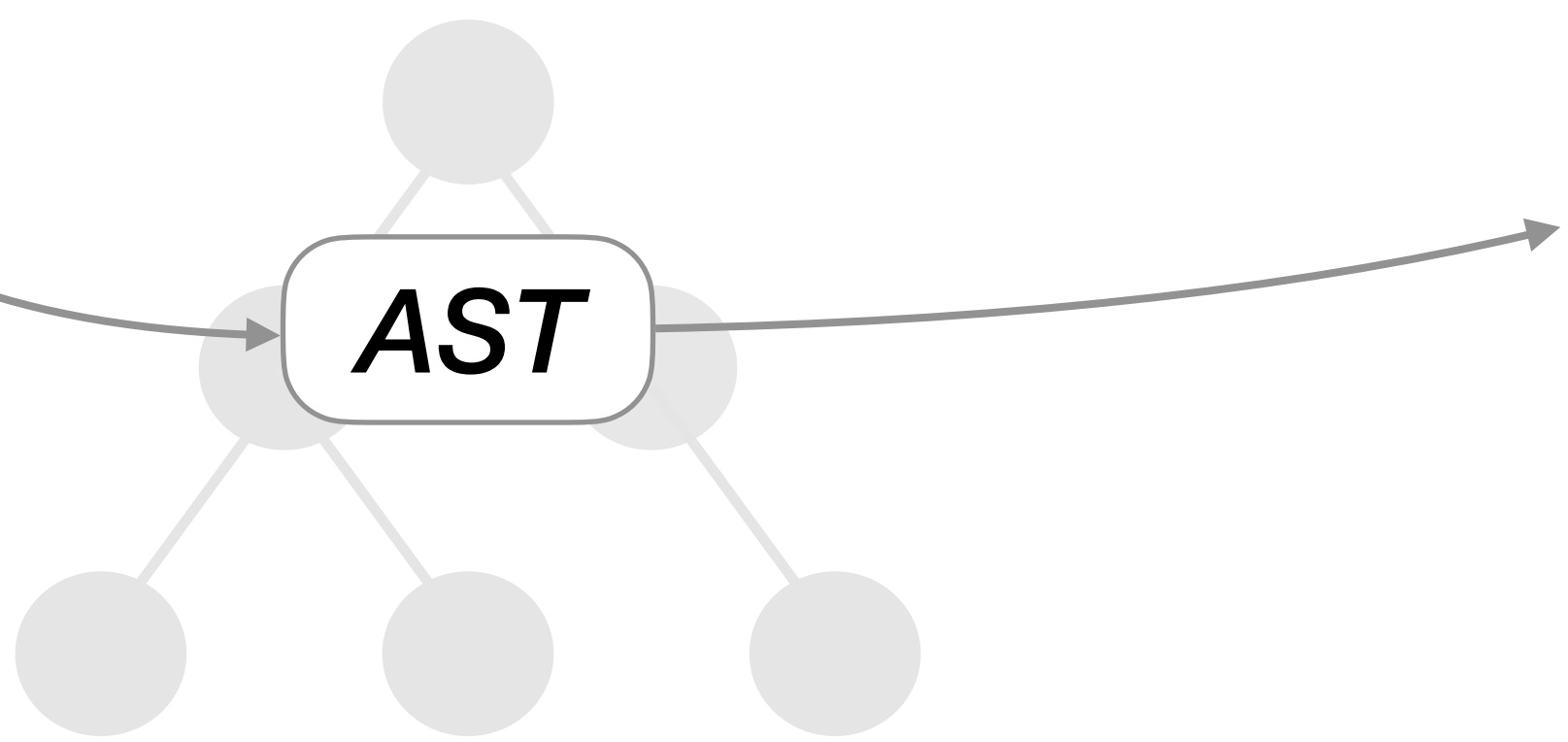
$G1$ and $G2$ are groups that both have a " $_ + _$ " notation.

Lemma `morph_id` :

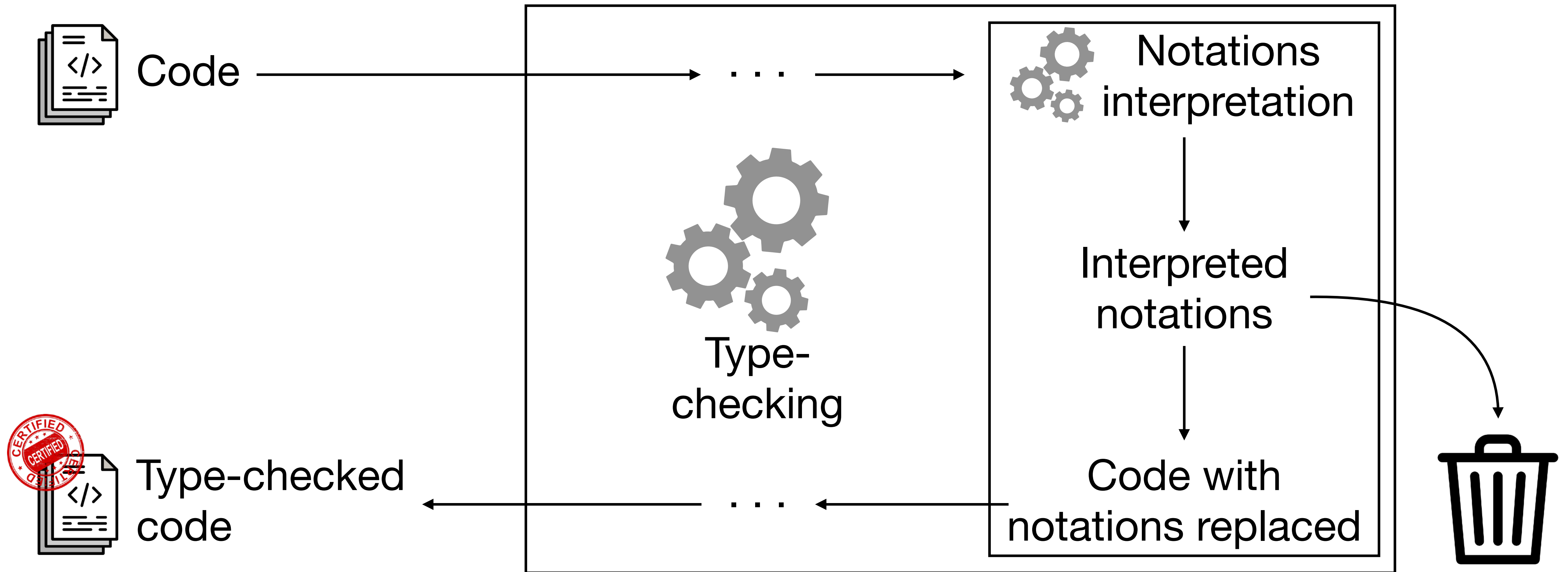
forall `morph` : $G1 \xrightarrow{1} G2$,
(**forall** `a b`, $\text{morph } a \xrightarrow{2} \text{morph } b \xrightarrow{3} \text{morph } (a \xrightarrow{4} b) \xrightarrow{5}$
 $\text{morph } id1 \xrightarrow{6} id2$.)

- 1. CNotation : " $_ \rightarrow _$ "
- 2. CNotation : " $_ + _$ "
- 3. CNotation : " $_ = _$ "
- 4. CNotation : " $_ + _$ "
- 5. CNotation : " $_ \rightarrow _$ "
- 6. CNotation : " $_ = _$ "

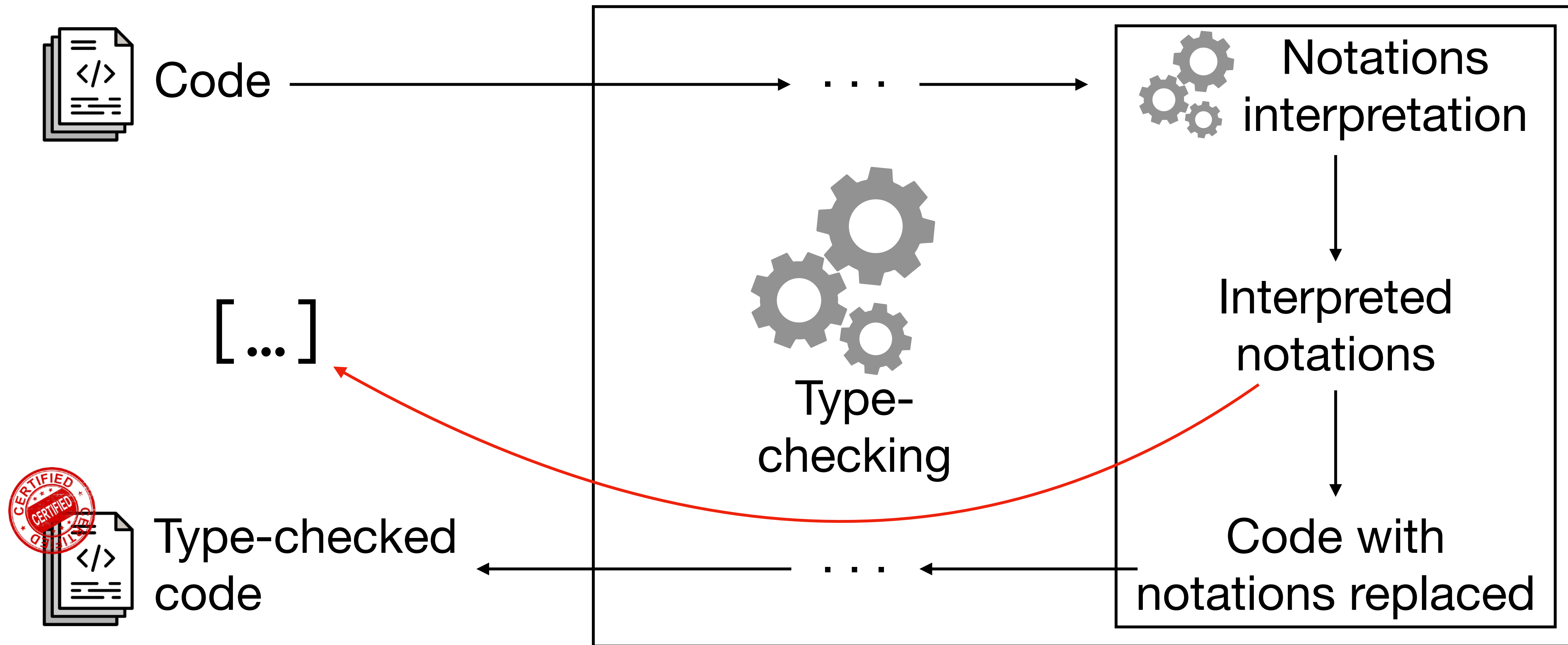
Impossible to know the scope...



Rocq interpreter



***Modified** Rocq interpreter*



```
1  let interp_notation = [...]
2  let instantiate_notation_constr n = [...]
3  let intern_notation n =
4    let notation = interp_notation n in
5    instantiate_notation_constr notation
6  let internalize (Wrap c) =
7    let rec intern = function
8      | CNotation n →
9        let c = intern_notation n in
10       intern c
11    | _ → [...]
12  in
13  intern c
14  let translate_vernac = function
15    | VernacSynPure (VernacDefinition e) →
16      internalize e
17    | _ → [...]
18  let interp v = translate_vernac v
```

```
1  let interp_notation = [...]
2  let instantiate_notation_constr n = [...]
3  let intern_notation n =
4    let notation = interp_notation n in
5    instantiate_notation_constr notation
6  let internalize (Wrap c) =
7    let rec intern = function
8      | CNotation n →
9        let c = intern_notation n in
10       intern c
11    | _ → [...]
12  in
13  intern c
14  let translate_vernac = function
15    | VernacSynPure (VernacDefinition e) →
16      internalize e
17    | _ → [...]
18  let interp v = translate_vernac v
```

```
1  let interp_notation = [...]
2  let instantiate_notation_constr n = [...]
3  let intern_notation n =
4    let notation = interp_notation n in
5    instantiate_notation_constr notation
6  let internalize (Wrap c) =
7    let rec intern = function
8      | CNotation n →
9        let c = intern_notation n in
10       intern c
11    | _ → [...]
12  in
13  intern c
```

```
14 let translate_vernac = function
15   | VernacSynPure (VernacDefinition e) →
16     internalize e
17   | _ → [...]
```

```
18 let interp v = translate_vernac v
```

```
1  let interp_notation = [...]
2  let instantiate_notation_constr n = [...]
3  let intern_notation n =
4    let notation = interp_notation n in
5    instantiate_notation_constr notation
6  let internalize (Wrap c) =
7    let rec intern = function
8      | CNotation n →
9        let c = intern_notation n in
10       intern c
11    | _ → [...]
12  in
13  intern c
```

```
14 let translate_vernac = function
15   | VernacSynPure (VernacDefinition e) →
16     internalize e
17   | _ → [...]
```

```
18 let interp v = translate_vernac v
```

```
1  let interp_notation = [...]  
2  let instantiate_notation_constr n = [...]  
  
3  let intern_notation n =  
4      let notation = interp_notation n in  
5      instantiate_notation_constr notation
```

```
6  let internalize (Wrap c) =  
7      let rec intern = function  
8          | CNotation n →  
9              let c = intern_notation n in  
10             intern c  
11      | _ → [...]  
12  in  
13  intern c
```

```
14 let translate_vernac = function  
15     | VernacSynPure (VernacDefinition e) →  
16         internalize e  
17     | _ → [...]
```

```
18 let interp v = translate_vernac v
```

```
1  let interp_notation = [...]  
2  let instantiate_notation_constr n = [...]  
  
3  let intern_notation n =  
4      let notation = interp_notation n in  
5      instantiate_notation_constr notation
```

```
6  let internalize (Wrap c) =  
7      let rec intern = function  
8          | CNotation n →  
9              let c = intern_notation n in  
10             intern c  
11      | _ → [...]  
12  in  
13  intern c
```

```
14 let translate_vernac = function  
15     | VernacSynPure (VernacDefinition e) →  
16         internalize e  
17     | _ → [...]
```

```
18 let interp v = translate_vernac v
```

```
1  let interp_notation = [...]
2  let instantiate_notation_constr n = [...]
3  let intern_notation n =
4    let notation = interp_notation n in
5    instantiate_notation_constr notation
6  let internalize (Wrap c) =
7    let rec intern = function
8      | CNotation n →
9        let c = intern_notation n in
10       intern c
11    | _ → [...]
12  in
13  intern c
14  let translate_vernac = function
15    | VernacSynPure (VernacDefinition e) →
16      internalize e
17    | _ → [...]
18  let interp v = translate_vernac v
```

```
1  let interp_notation = [...]
2  let instantiate_notation_constr n = [...]
3  let intern_notation n =
4    let notation = interp_notation n in
5    instantiate_notation_constr notation
6  let internalize (Wrap c) =
7    let rec intern = function
8      | CNotation n →
9        let c = intern_notation n in
10       intern c
11    | _ → [...]
12  in
13  intern c
14  let translate_vernac = function
15    | VernacSynPure (VernacDefinition e) →
16      internalize e
17    | _ → [...]
18  let interp v = translate_vernac v
```

```
1  let interp_notation = [...]
2  let instantiate_notation_constr n = [...]
3  let intern_notation n =
4    let notation = interp_notation n in
5    instantiate_notation_constr notation
6  let internalize (Wrap c) =
7    let rec intern = function
8      | CNotation n →
9        let c = intern_notation n in
10       intern c
11    | _ → [...]
12  in
13  intern c
14 let translate_vernac = function
15   | VernacSynPure (VernacDefinition e) →
16     internalize e
17   | _ → [...]
18 let interp v = translate_vernac v
```

```
1  let interp_notation = [...]
2  let instantiate_notation_constr n = [...]
3  let intern_notation n =
4    let notation = interp_notation n in
5    instantiate_notation_constr notation
6  let internalize (Wrap c) =
7    let rec intern = function
8      | CNotation n →
9        let c = intern_notation n in
10       intern c
11    | _ → [...]
12  in
13  intern c
14 let translate_vernac = function
15   | VernacSynPure (VernacDefinition e) →
16     internalize e
17   | _ → [...]
18 let interp v = translate_vernac v
```

```
1  let interp_notation = [...]
2  let instantiate_notation_constr n = [...]
3  let intern_notation n =
4    let notation = interp_notation n in
5  - instantiate_notation_constr notation
6  let internalize (Wrap c) =
7    let rec intern = function
8      | CNotation n →
9        let c = intern_notation n in
10       intern c
11    | _ → [...]
12  in
13  intern c
14 let translate_vernac = function
15   | VernacSynPure (VernacDefinition e) →
16     internalize e
17   | _ → [...]
18 let interp v = translate_vernac v
```

```
1  let interp_notation = [...]
2  let instantiate_notation_constr n = [...]
3  let intern_notation n =
4    let notation = interp_notation n in
5  + notation, instantiate_notation_constr notation
6  let internalize (Wrap c) =
7    let rec intern = function
8      | CNotation n →
9        let c = intern_notation n in
10       intern c
11    | _ → [...]
12  in
13  intern c
14 let translate_vernac = function
15   | VernacSynPure (VernacDefinition e) →
16     internalize e
17   | _ → [...]
18 let interp v = translate_vernac v
```

```

1  let interp_notation = [...]

2  let instantiate_notation_constr n = [...]

3  let intern_notation n =
4      let notation = interp_notation n in
5  - instantiate_notation_constr notation

6  let internalize (Wrap c) =
7  - let rec intern = function
8      | CNotation n →
9  -         let c = intern_notation n in
10 -         intern c
11 - | _ → [...]
12      in
13 - intern c

14 let translate_vernac = function
15     | VernacSynPure (VernacDefinition e) →
16         internalize e
17 - | _ → [...]

18 let interp v = translate_vernac v

```

```

1  let interp_notation = [...]

2  let instantiate_notation_constr n = [...]

3  let intern_notation n =
4      let notation = interp_notation n in
5  + notation, instantiate_notation_constr notation

6  let internalize (Wrap c) =
7  + let rec intern acc = function
8      | CNotation n →
9  +         let n, c = intern_notation n in
10 +         intern (n :: acc) c
11 + | _ → acc, [...]
12      in
13 + intern [] c

14 let translate_vernac = function
15     | VernacSynPure (VernacDefinition e) →
16         internalize e
17 + | _ → [], [...]

18 let interp v = translate_vernac v

```

Notations

Decomposition

Current goal

$n : \text{nat}$

$\text{fact_div} :$

$n \% | \text{factorial } n$

$0 < n$

Notations

Decomposition

Current goal

$n : \text{nat}$

$\text{fact_div} :$

$n \% \mid \text{factorial } n$

$0 < n$

Isolation

Definition $n : \text{nat}.$

Definition $\text{fact_div} :$

$n \% \mid \text{factorial } n.$

Definition $\text{concl} : 0 < n.$

Why isolation?

→ issue with *parsing only* notations

The coercion $x = y :> T$

is visualised as $x = y$

in the current goal.

Error occurs

when type-checking $x = y$

With isolation, error occurs only on concerned hypotheses.

Notations

Decomposition

Current goal

`n : nat`
`fact_div :`
`n %| factorial n`

`0 < n`

Isolation

Definition `n : nat.`
Definition `fact_div :`
`n %| factorial n.`
Definition `concl : 0 < n.`

Notations

Extraction

`[]`

`["_ %| _"]`

`["_ < _"]`

Retrieval

`"m %| d" := (dvdn m d) : nat_scope`

`"m < n" := (leq (S m) n) : nat_scope`

Modified Interpreter

Context enrichment:
LLM4Docq

MathComp: ~15,000 theorems

→ use LLMs to annotate the library

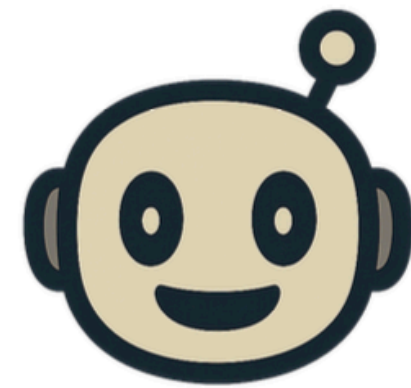
source file

```
Lemma addnC :  
  commutative addn.
```

```
Proof. [ ... ]
```

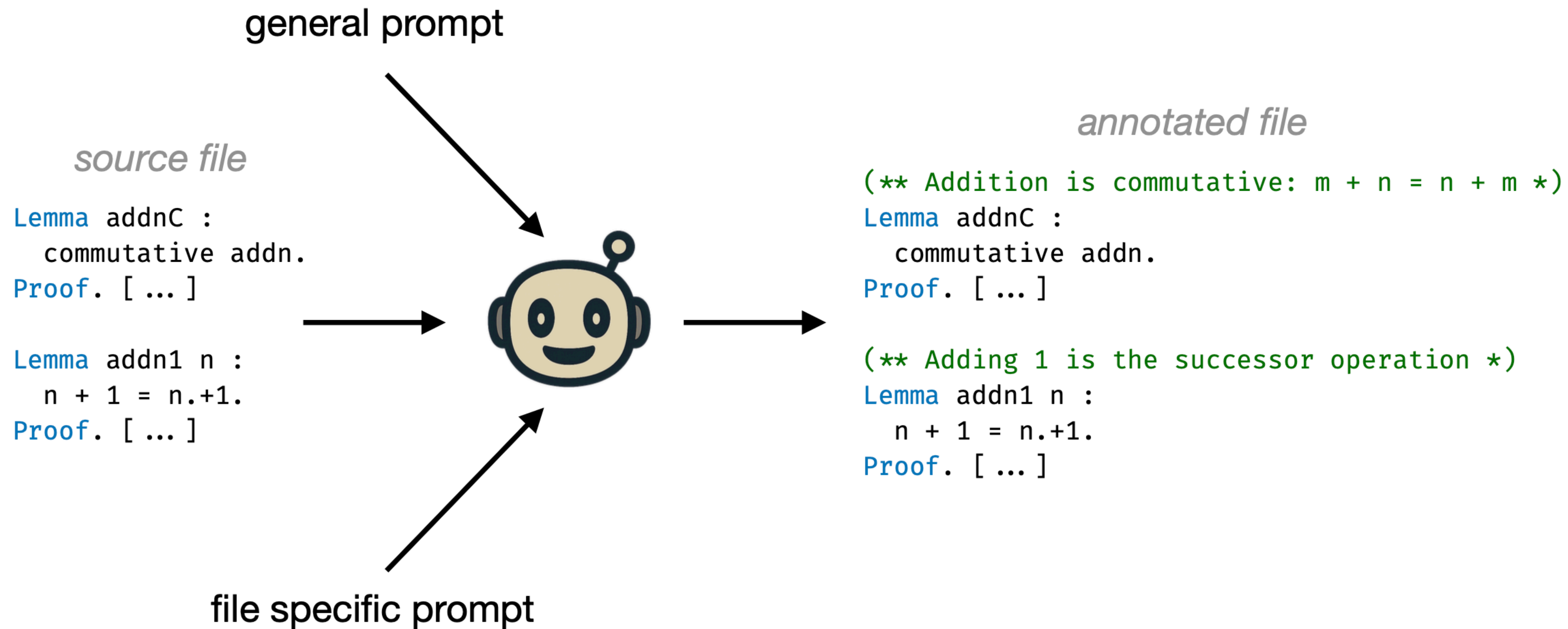
```
Lemma addn1 n :  
  n + 1 = n.+1.
```

```
Proof. [ ... ]
```



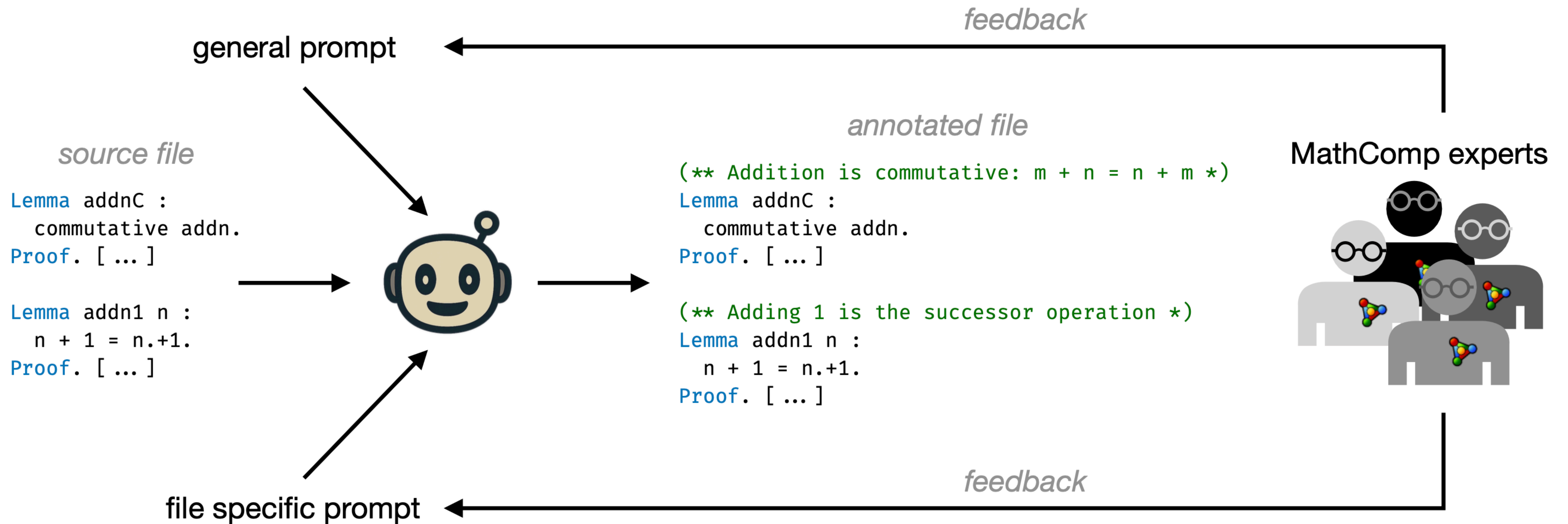
MathComp: ~15,000 theorems

→ use LLMs to annotate the library



MathComp: ~15,000 theorems

→ use LLMs to annotate the library



MathComp: ~15,000 theorems

→ use LLMs to annotate the library

source file

```
Lemma addnC :  
  commutative  
Proof. [ ... ]  
  
Lemma addn1 n  
  n + 1 = n.+1  
Proof. [ ... ]
```

f

#22483
2 of 2

Location: mathcomp.field.algebraics_fundamentals."<< E ; u >>"

Code

Notation "<< E ; u >>" := <<E; u>>%VS.

Docstring

A notation representing the vector subspace generated by the set E along with the element u, denoting the subspace spanned by combining E and u.

Annotation

☒ Acceptable^[1]
☐ Needs Improvement^[2]
☐ Incorrect^[3]

Improved version

Please provide additional comments

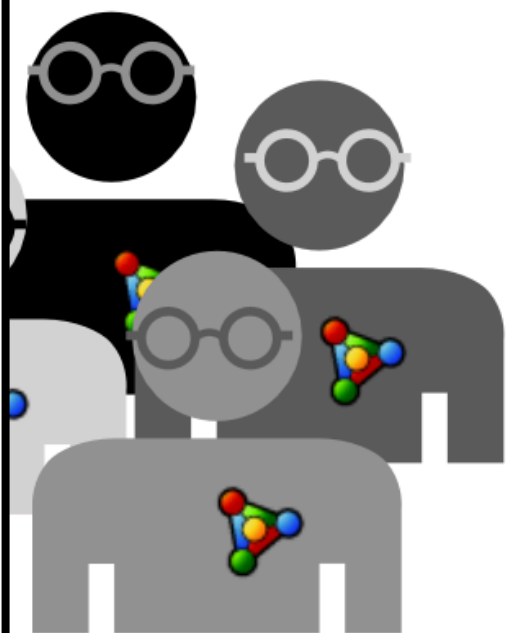
⏮ ⏪ ⏩ ⏭

Skip

Submit

▼

comp experts



LLM4Docq results

- 2 full iterations over MathComp = 20,000 objects annotated
- 600 docstrings reviewed:

Responses	Percentage		Need improvement issue	Percentage
Acceptable	82 %		Inconsistent phrasing or terminology	48 %
Need improvement	15 %		Too verbose	32 %
Incorrect	3 %		Lack precision / omit key details	20 %

Experiments

- **Configurations:** $G \rightarrow G + D + L \rightarrow G + D \& N \rightarrow G + D \& N + L$

Example:

```
n : nat
fact_div :
  n %| factorial n
  -----
  0 < n
```

```
Inductive nat : Set :=
  | 0 : nat
  | S : nat → nat.

Fixpoint factorial n :=
  if n is n'.+1 then n * factorial n'
  else 1.
```

```
Notation "m %| d" := (dvdn m d) : nat_scope.
```

```
Notation "m < n" := (leq (S m) n) : nat_scope.
```

This is an inductive type representing the set of natural numbers, ...

This definition computes the factorial of a natural number, ...

This notation provides a symbol for the divisibility relation ...

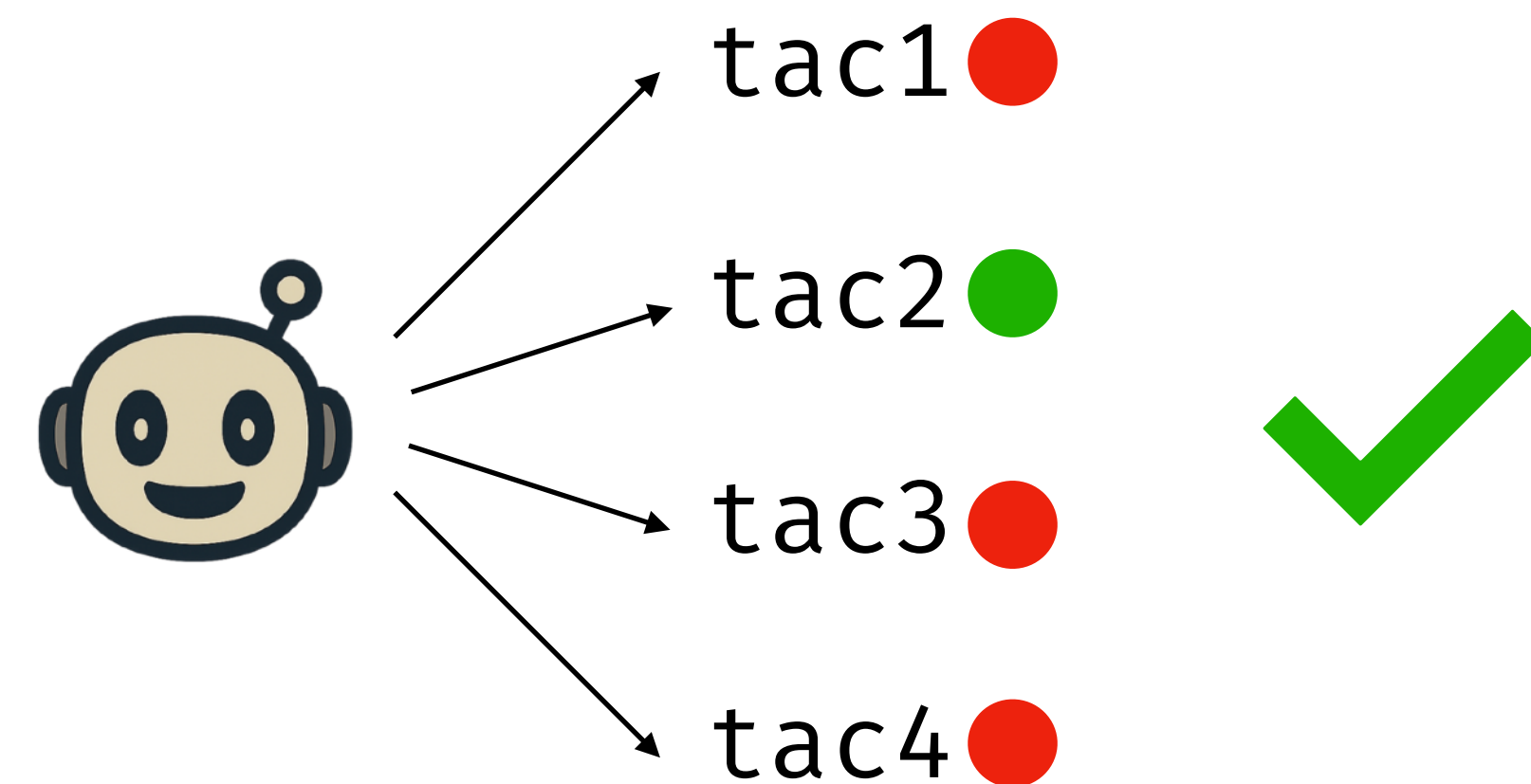
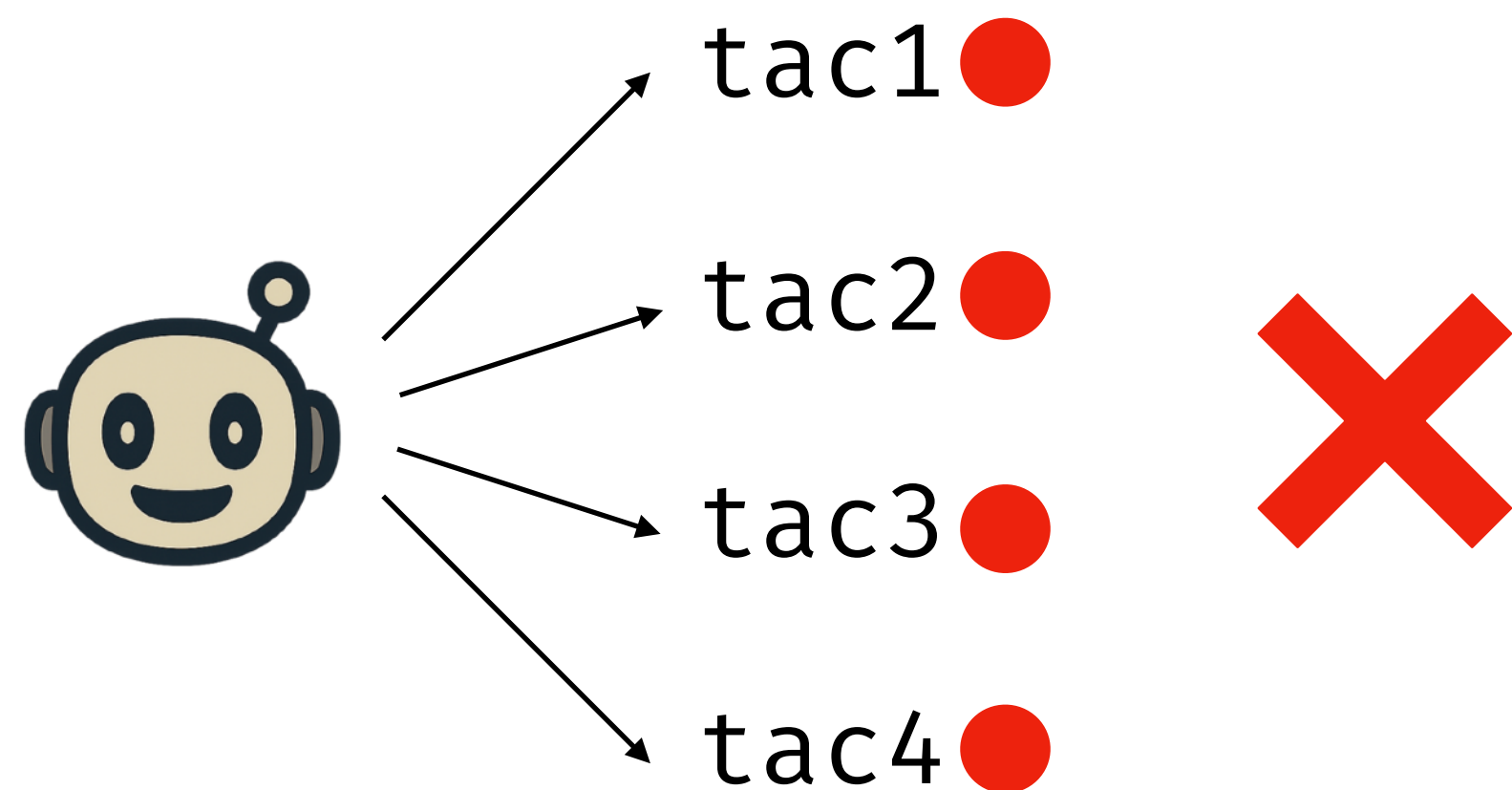
This notation represents the 'strictly less than' comparison ...

- **Configurations:** G — $G + D + L$ — $G + D \& N$ — $G + D \& N + L$

- **Pass@k:** $k \in \{1, 4, 8, 32\}$

probability of one success among k generated samples

Example: pass@4



Results

1. Bigger models benefit the most from additional context

Model	Configuration	pass@1	pass@4	pass@8	pass@32
GPT 4o	G	32.7	52.4	61.2	—
	G + D + L	35.2	55.2	64.3	—
	G + D & N	41.2	60.7	68.0	—
	G + D & N + L	42.1	61.7	70.4	—
CLAUDE SONNET 4	G	47.8	59.4	64.5	—
	G + D + L	43.5	57.7	62.8	—
	G + D & N	45.8	57.7	62.6	—
	G + D & N + L	50.1	62.9	67.8	—
QWEN3 32B	G	25.5	47.6	58.2	74.4
	G + D + L	27.8	50.3	60.7	77.2
	G + D & N	29.8	53.2	63.0	77.4
	G + D & N + L	29.7	53.0	63.1	77.8
QWEN3 14B	G	26.2	43.4	51.8	67.1
	G + D + L	23.8	40.9	47.5	65.1
	G + D & N	23.5	41.2	49.8	65.7
	G + D & N + L	24.1	41.8	50.4	65.7
QWEN3 4B	G	11.1	22.6	30.3	48.5
	G + D + L	10.9	22.7	30.2	47.6
	G + D & N	9.4	19.9	27.1	42.9
	G + D & N + L	11.0	22.5	29.8	46.8

Results

- 1. Bigger models benefit the most from additional context
- 2. Smaller models seem to lack sufficient ability to use the context

Model	Configuration	pass@1	pass@4	pass@8	pass@32
GPT 4o	G	32.7	52.4	61.2	—
	G + D + L	35.2	55.2	64.3	—
	G + D & N	41.2	60.7	68.0	—
	G + D & N + L	42.1	61.7	70.4	—
CLAUDE SONNET 4	G	47.8	59.4	64.5	—
	G + D + L	43.5	57.7	62.8	—
	G + D & N	45.8	57.7	62.6	—
	G + D & N + L	50.1	62.9	67.8	—
QWEN3 32B	G	25.5	47.6	58.2	74.4
	G + D + L	27.8	50.3	60.7	77.2
	G + D & N	29.8	53.2	63.0	77.4
	G + D & N + L	29.7	53.0	63.1	77.8
QWEN3 14B	G	26.2	43.4	51.8	67.1
	G + D + L	23.8	40.9	47.5	65.1
	G + D & N	23.5	41.2	49.8	65.7
	G + D & N + L	24.1	41.8	50.4	65.7
QWEN3 4B	G	11.1	22.6	30.3	48.5
	G + D + L	10.9	22.7	30.2	47.6
	G + D & N	9.4	19.9	27.1	42.9
	G + D & N + L	11.0	22.5	29.8	46.8

Results

1. Bigger models benefit the most from additional context

2. Smaller models seem to lack sufficient ability to use the context

3. Qwen3 models: trained with mathematical and coding tasks
→ may explain their overall good performance

Model	Configuration	pass@1	pass@4	pass@8	pass@32
GPT 4o	G	32.7	52.4	61.2	—
	G + D + L	35.2	55.2	64.3	—
	G + D & N	41.2	60.7	68.0	—
	G + D & N + L	42.1	61.7	70.4	—
CLAUDE SONNET 4	G	47.8	59.4	64.5	—
	G + D + L	43.5	57.7	62.8	—
	G + D & N	45.8	57.7	62.6	—
	G + D & N + L	50.1	62.9	67.8	—
QWEN3 32B	G	25.5	47.6	58.2	74.4
	G + D + L	27.8	50.3	60.7	77.2
	G + D & N	29.8	53.2	63.0	77.4
	G + D & N + L	29.7	53.0	63.1	77.8
QWEN3 14B	G	26.2	43.4	51.8	67.1
	G + D + L	23.8	40.9	47.5	65.1
	G + D & N	23.5	41.2	49.8	65.7
	G + D & N + L	24.1	41.8	50.4	65.7
QWEN3 4B	G	11.1	22.6	30.3	48.5
	G + D + L	10.9	22.7	30.2	47.6
	G + D & N	9.4	19.9	27.1	42.9
	G + D & N + L	11.0	22.5	29.8	46.8

Metrics:

✓ does the generated tactic type-checks ?

✗ *does the generated tactic produces a new goal ?* — Almost always true.

✗ *does the generated tactic is the same as the MathComp one ?* — Almost always false.

✗ *does the generated tactic produces the same MathComp goal ?* — Almost always false.

Motivations:

- ✓ Opaque models: what is important to them?
- ✓ Useful for Rocq users
- ✓ Not enough compute power

Tacq:

- **Experiments** on models and available information
- **Tools:**
 - *Pytanque*: context extraction
 - *LLM4Docq*: automatic natural language annotation
- **Next tactic prediction** pipeline

Futur works:

- Fine tuning models on next tactic prediction
- Explore other predictions (e.g., the target goal)
- LLM4Docq: annotate content defined implicitly by Hierarchy Builder

Pull requests:

 [petanque] New command petanque/list_notations_in_statement ✓ kind: enhancement needs: test-case 2 part: petanque #1017 by JulesViennotFranca was merged on Nov 27, 2025 0.2.5
 [intern] Allow to instrument internalization via open recursion. ✓ kind: feature kind: infrastructure kind: internal 10 kind: redesign part: elaboration part: implicit arguments part: notations part: vernac #21251 by ejgallego was merged on Nov 26, 2025 • Approved 9.2+rc1

Special **thanks** to MathComp experts that helped us review LLM4Docq annotations!

The list of contributors can be seen at <https://github.com/LLM4Rocq/LLM4Docq-MathComp>

If you also want to contribute, follow the instructions written in the GitHub repository.