Natural-Language Mathematical Statements to Formal Statements

Autoformalization as alignment, not translation

XMUM GE – Week 8

Ma Jiajun – May 26, 2026

Part I. Opening Case Study

One symbol, many formal objects

One Symbol, Many Formal Objects: Integral

On the board only this:

$$\int_a^b f(x)\,dx$$

> Question: to write this as a Lean theorem statement, what is missing?

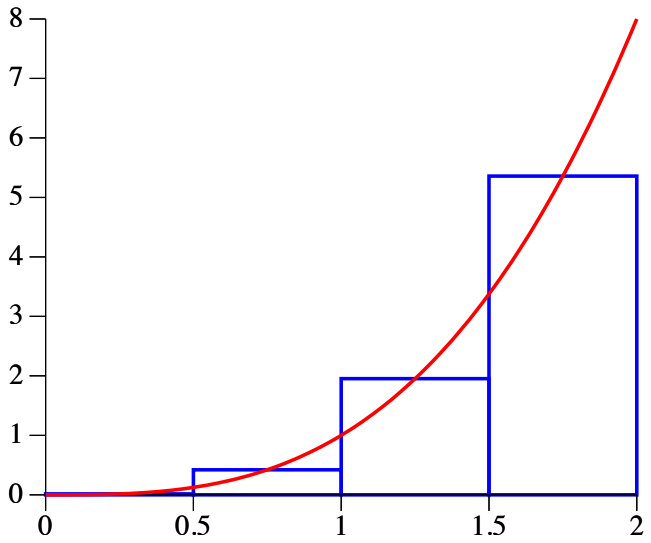
The missing piece is not Lean syntax. It is the mathematical object.

Possible readings

1. Riemann integral on a compact interval.
2. Lebesgue / Bochner integral wrt a measure.
3. Oriented interval integral $\int_a^b$.
4. Set integral over $[a, b]$, $(a, b]$, …
5. Variable upper limit $x \mapsto \int_a^x f(t)\,dt$.
6. Antiderivative relation $F' = f$.

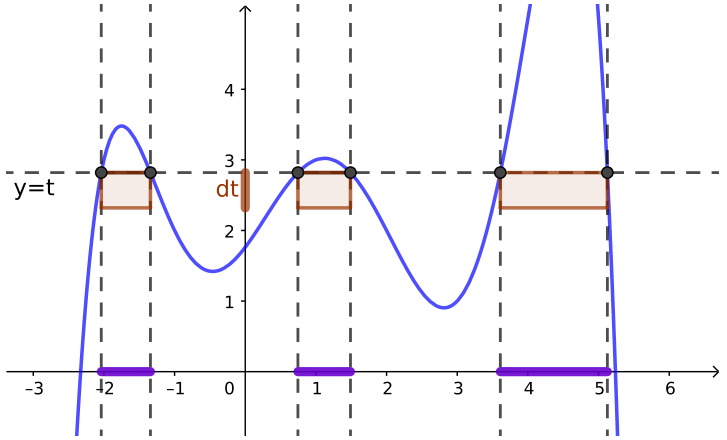
Riemann vs Lebesgue: Two Ways To Sum The Area

Riemann – slice the x-axis



$$\int f \approx \sum_i f(x_i)\,\Delta x_i$$

Lebesgue – slice the y-axis



$$\int f \approx \sum_j y_j\,\mu\big(\{x : f(x) \geq y_j\}\big)$$

Cut the **range** into thin horizontal layers. Sum **value × measure of the level set**.

Mathlib Says "As $R \to \infty$" With Filter and Tendsto

What the natural sentence says

$$\int_1^R \frac{\sin x}{x}\,dx \longrightarrow L \quad \text{as } R \to \infty$$

What Mathlib writes

```
Tendsto
  (fun R ⇒ ∫ x in 1..R, sin x / x)
  atTop
  (𝓝 L)
```

Three pieces — a function, a *source filter*, a *target filter*. No ε-δ on the surface.

A filter is "eventually"

A `Filter` on a type packages "what counts as *eventually*". The two we need:

> `atTop : Filter ℝ` — eventually, R is arbitrarily large.

> `𝓝 L : Filter ℝ` — eventually, the value lies in any neighborhood of L.

`Tendsto f F G` reads:

```
f pushes "eventually in F"
  to "eventually in G".
```

For our example: as R is eventually large, the partial integral is eventually near L. That is exactly

Running Example: Improper Riemann Exists, Lebesgue Does Not

$$\int_1^\infty \frac{\sin x}{x}\,dx \text{ exists.}$$

Reading A – improper Riemann integral: **true**

```
∃ L : ℝ,
  Tendsto
    (fun R : ℝ ⇒ ∫ x in (1:ℝ)..R, Real.sin x / x)
    atTop
    (𝓝 L)
```

Finite interval integrals have a limit as $R \to \infty$.

Reading B – Lebesgue integrable on $[1, \infty)$: **false**

```
¬ IntegrableOn
    (fun x : ℝ ⇒ Real.sin x / x)
    (Set.Ici 1)
```

$\int_1^\infty |\sin x/x|\,dx = \infty$. Lebesgue integrability needs absolute integrability; this is only conditional convergence.

Putnam 2025 B2: Geometry Becomes Integral Ratios

Let $f : [0, 1] \to [0, \infty)$ be strictly increasing and continuous. Let R be the region under $y = f(x)$ on $[0, 1]$. Let x_1 be the x-coordinate of the centroid of R. Let x_2 be the x-coordinate of the centroid of the solid generated by rotating R about the x-axis. Prove $x_1 < x_2$.

```
theorem putnam_2025_b2
    (f : ℝ → ℝ)
    (hf_cont : ContinuousOn f (Icc 0 1))
    (hf_mono : StrictMonoOn f (Icc 0 1))
    (hf_nonneg : ∀ x ∈ Icc (0 : ℝ) 1, 0 ≤ f x) :
    (∫ x in (0:ℝ)..1, x * f x) /
      (∫ x in (0:ℝ)..1, f x) <
    (∫ x in (0:ℝ)..1, x * (f x) ^ 2) /
      (∫ x in (0:ℝ)..1, (f x) ^ 2) := by
  sorry
```

The formal statement does not define R, centroid, or solid of revolution. It uses the analytic formulas directly.

Geometric narrative replaced by analytically equivalent integral ratios. The π cancels; denominator positivity is a proof obligation, not a hypothesis.

Gemini Draft: Correct Shape, Lean Syntax Fixes

Two layers on this slide

1. Apply Lean syntax fixes from earlier sessions: `import Mathlib`, `open Set Real MeasureTheory Interval`, `noncomputable` section, `Real.pi`, parameterize over `f`.
2. Once it typechecks, Gemini's **mathematical shape** is right.

```
noncomputable section

def area_R (f : ℝ → ℝ) : ℝ :=
  ∫ x in (0:ℝ)..1, f x

def x1 (f : ℝ → ℝ) : ℝ :=
  (∫ x in (0:ℝ)..1, x * f x) / area_R f

def vol_solid (f : ℝ → ℝ) : ℝ :=
  Real.pi * ∫ x in (0:ℝ)..1, (f x)^2

def x2 (f : ℝ → ℝ) : ℝ :=
  (Real.pi * ∫ x in (0:ℝ)..1, x * (f x)^2) /
    vol_solid f
```

Both versions express the same analytic comparison: AxiomMath inlines the integrals; Gemini names them. Presentation choice, not error.

"Indefinite Integral" Is Not A Defined Object

(A) Derivative side

```
HasDerivAt : (𝕜 → F) → F → 𝕜 → Prop
-- predicate: F' = f x at x
deriv : (𝕜 → F) → 𝕜 → F
-- total function; junk value 0 if no deriv
```

In `Mathlib.Analysis.Calculus.Deriv.Basic`. `HasDerivAt` is a `Prop`, `deriv` is a value.

(B) Definite integral side

```
intervalIntegral
  : (ℝ → E) → ℝ → ℝ → Measure ℝ → E
-- notation: ∫ x in a..b, f x ∂μ
```

In `Mathlib.MeasureTheory.Integral.IntervalIntegral.Ba` built on Bochner `MeasureTheory.integral`.

There is **no** `indefiniteIntegral` / `antiderivative` / `primitive` def in mathlib. Any "indefinite integral" must reduce to (A) or (B); FTC is the theorem connecting them, not a definition.

Three Informal Phrases, Two Primitives

Natural language	Real identity	Mathlib expression
"F is an antiderivative of f"	predicate on side (A)	`∀ x, HasDerivAt F (f x) x`
"the antiderivatives of f"	set carved out by that predicate	`{F | ∀ x, HasDerivAt F (f x) x}`
"$\int_a^x f(t)\,dt$"	a function on side (B)	`fun x ⇒ ∫ t in a..x, f t`

The first row uses **only** the derivative side. There is no integral in it.

The second row is a **set-builder**, not a `def`. mathlib has no `antiderivatives` definition.

The third row uses **only** the integral side. It is just a lambda over `intervalIntegral`.

Every alignment of "the indefinite integral of f" has to commit to one of three forms above – and therefore to one of the two primitives (A) or (B).

`deriv f x = 0` Is **Not** Evidence Of Differentiability

Core warning. `deriv f x = 0` does **not** imply that f is differentiable at x. The equation is **not** evidence of differentiability.

Why

`deriv` is a **total function**: Lean requires a value at every point.

If f is not differentiable at x, then `deriv f x` returns `0` by convention (junk value).

Two interpretations of `deriv f x = 0`

1. f *is* differentiable at x, derivative happens to be 0; **or**
2. f is *not* differentiable at x, `deriv` returned the default 0.

You cannot tell which from the equation alone.

Correct shape for "$f'(0) = 0$": use `HasDerivAt f 0 0` (packs existence + value together), or hold `DifferentiableAt ℝ f 0` as a separate hypothesis before reading off `deriv`. This is exactly why side (A) provides **both** `HasDerivAt` (assertion) and `deriv` (value).

Heaviside Step: The Junk-Value In Action

Setup

$$H(x) = \mathbf{1}_{[0,\infty)}(x)$$

```
def H : ℝ → ℝ :=
  Set.indicator (Set.Ici 0) (fun _ ⇒ 1)
```

H is **not continuous** at 0, so certainly not differentiable there.

Yet in Lean

```
example : deriv H 0 = 0 := by sorry
```

This holds – but via the **junk-value** path, not because the derivative equals 0.

`deriv H 0 = 0` ✓ (true)
`DifferentiableAt ℝ H 0` ✗ (false)
`HasDerivAt H 0 0` ✗ (false)

Reading `deriv f x = 0` as "the derivative at x is 0" is wrong whenever the function is not differentiable there. The compiler accepts the equation; the mathematical content has silently dropped out.

Fundamental Theorem Of Calculus As Statement Shapes

FTC-1

If $F(x) = \int_a^x f(t)\,dt$, then $F'(x) = f(x)$.

```
HasDerivAt
  (fun x ⇒ ∫ t in a..x, f t) (f x) x
```

FTC-2

If $F'(x) = f(x)$, then $\int_a^b f(x)\,dx = F(b) - F(a)$.

```
(∫ x in a..b, f' x) = F b - F a
```

Mathlib theorem names:

`intervalIntegral.integral_eq_sub_of_hasDerivAt`

`intervalIntegral.integral_deriv_eq_sub'`

The real content lives in the statement

- `HasDerivAt`, `HasDerivWithinAt`, or `deriv`?
- Where does the derivative hold?
- Integrability hypothesis?
- Oriented interval?
- Codomain structure?

"The fundamental theorem of calculus" $\Longrightarrow$ a family of theorem schemas.

Worked Example: $f' \equiv 0 \Longrightarrow f$ Constant (1/2)

Theorem (informal). If f is continuous on $[a, b]$ and $f' = 0$ on (a, b), then f is constant on $[a, b]$.

The "obvious" formal shape

```
theorem constant_of_deriv_zero
    {a b : ℝ} (hab : a < b) {f : ℝ → ℝ}
    (hcont : ContinuousOn f (Icc a b))
    (hdiff : DifferentiableOn ℝ f (Ioo a b))
    (hderiv : ∀ x ∈ Ioo a b, deriv f x = 0) :
    ∀ x ∈ Icc a b, ∀ y ∈ Icc a b, f x = f y :=
  sorry
```

Reads off the informal sentence directly: continuity on $[a, b]$, differentiability on (a, b), $f' = 0$ on (a, b), conclude constancy.

No Mathlib lemma has this exact shape

Closest closed-interval lemmas in `Mathlib.Analysis.Calculus.MeanValue`:

- `constant_of_derivWithin_zero`
- `constant_of_has_deriv_right_zero`
- `is_const_of_deriv_eq_zero` (no interval)

None accept "`DifferentiableOn` + `deriv = 0` on `Ioo a b`" as written. The alignment gap is in the **derivative notion**: at endpoints Mathlib uses `derivWithin` / `HasDerivWithinAt`, not `deriv` / `HasDerivAt`.

Worked Example: $f' \equiv 0 \Longrightarrow f$ Constant (2/2)

Real Mathlib API

```
-- closed interval ⟹ derivWithin, not deriv
example
    {a b : ℝ} {f : ℝ → ℝ}
    (hdiff : DifferentiableOn ℝ f (Icc a b))
    (hderiv : ∀ x ∈ Ico a b,
      derivWithin f (Icc a b) x = 0) :
    ∀ x ∈ Icc a b, f x = f a :=
  constant_of_derivWithin_zero
    hdiff hderiv
```

The lemma is `constant_of_derivWithin_zero`. Same lesson as the LeanArchitect / Multivariate Taylor case at the end of Part IV: at endpoints, use the `Within` variants.

Four alignment shifts vs. draft

- `DifferentiableOn` on `Icc` (continuity inherited)
- `derivWithin`, not `deriv`
- condition on `Ico a b`, not `Ioo a b`
- conclusion pinned to $f(a)$, not symmetric

The "deriv = 0 ⟹ constant" theorem has at least two real Mathlib shapes – both use the `Within` variants. **Neither** uses `deriv` or `HasDerivAt` directly.

Mathlib Does Not Start From Textbook Notation

Notation used by the Putnam B2 statement earlier:

```
∫ x in a..b, f x
```

comes from `MeasureTheory.Integral.IntervalIntegral`. Not Riemann sums as primary definition; an oriented interval integral built on the measure-theoretic integral.

```
∫ x in a..b, f x ∂μ
  = ∫ x in Ioc a b, f x ∂μ
  - ∫ x in Ioc b a, f x ∂μ
```

Orientation is part of the API:

```
∫ x in a..a, f x ∂μ = 0
∫ x in b..a, f x ∂μ
  = - ∫ x in a..b, f x ∂μ
∫ x in a..b, f x ∂μ
  + ∫ x in b..c, f x ∂μ
  = ∫ x in a..c, f x ∂μ
```

Formal libraries often choose definitions for theorem engineering, not for matching first-semester textbook wording.

Bochner Integral: Lebesgue For Vector-Valued Functions

Lebesgue you already saw

$$f : \mathbb{R} \to \mathbb{R}, \qquad \int_a^b f\,dx \in \mathbb{R}$$

The values are **real numbers**.

Concrete jump: a moving point in space

A particle's velocity is a vector that changes with time:

$$\mathbf{v} : [0, T] \to \mathbb{R}^3, \qquad \mathbf{v}(t) = (v_x(t), v_y(t), v_z(t))$$

Total displacement:

$$\int_0^T \mathbf{v}(t)\,dt \;\in\; \mathbb{R}^3$$

Bochner integral: same idea, any value type

$$f : \mathbb{R} \to E, \qquad \int f\,dx \;\in\; E$$

E can be $\mathbb{R}$, $\mathbb{R}^3$, $\mathbb{C}$, … – any space where "add vectors" and "measure size" make sense.

> When $E = \mathbb{R}$ the Bochner integral **equals** the Lebesgue integral. Nothing new.

Why does Mathlib write it like this?

So one definition covers every case:

```
E = ℝ    → real-valued     (sin x / x, FTC)
E = ℂ    → complex-valued  (Fourier)
E = ℝⁿ   → vector-valued   (velocity, force)
```

The Underlying Integral Is Bochner Integral

`MeasureTheory.integral` is the Bochner integral.

```
MeasureTheory.integral
  input:
    measure μ on domain α
    function f : α → E
    structure on E
      sufficient for integration
  output:
    element of E
```

Not only real-valued functions; takes values in suitable normed vector spaces.

Common notations:

```
∫ x, f x ∂μ
∫ x in s, f x ∂μ
∫ x in a..b, f x ∂μ
```

> Behind ordinary-looking notation sits a hierarchy:

```
measurable space / measure
  → integrable functions
  → Bochner integral
  → set integral
  → interval integral
```

Formalization step one: locate the concept in the library.

Mathlib Also Has Riemann-Style Integrals

Riemann-style integral lives in a separate module, `BoxIntegral`:

```
BoxIntegral.HasIntegral
BoxIntegral.Integrable
BoxIntegral.integral
BoxIntegral.IntegrationParams.Riemann
```

The same framework hosts Henstock–Kurzweil and McShane via different `IntegrationParams`.

Textbook phrase → Mathlib object:

```
"the Riemann integral"
  → BoxIntegral with
     IntegrationParams.Riemann

"the integral from a to b"
  → intervalIntegral built
     from MeasureTheory.integral
```

Bridge theorems connect box-style and measure-theoretic integrals under suitable hypotheses. The library keeps multiple related definitions; one is the more common proof interface.

What The Integral Case Teaches About Alignment

Template for the rest of the lecture:

```
informal phrase
  "the integral of f from a to b"

formal choices
  type of f
  codomain structure
  measure
  interval convention
  integrability condition
  endpoint behavior
  intended theorem shape
```

> *An informal mathematical statement is often a compressed reference to a whole formal design space.*

The integral example was concrete. Part II abstracts it into a general checklist.

Part II. General Mechanism

From informal sentence to formal statement

The Statement Is The Object

Week 8 builds:

> **a formal theorem statement**
> (not a proof)

Two checks on a formal statement:

1. It typechecks in the formal language.
2. It expresses the intended mathematical meaning of the informal statement.

> Dangerous autoformalization failures pass (1) but fail (2).

Example:

> Informal: *Every subgroup of a cyclic group is cyclic.*

A type-correct but wrong formalization could:

- state the ambient group is cyclic in the conclusion instead of the subgroup;
- use the wrong formal definition of "cyclic".

Alignment Checklist

1. **Objects.** Variables, functions, sets, structures, parameters.
2. **Types.** Each object's type.
3. **Structures.** Algebraic, order, topological, metric, measure, category.
4. **Quantifiers.** Universal, existential, order.
5. **Assumptions.** Explicit vs convention-hidden.
6. **Definitions.** Which library definition corresponds.
7. **Notation.** Overloaded symbols disambiguated.
8. **Statement strength.** Too weak, too strong, just right.
9. **Edge cases.** 0, empty set, equal endpoints, boundaries.
10. **Back-translation.** Translate the formal back; does it still read as the original theorem?

This is the human version of statement validation. Modern systems (Part IV) automate parts of it.

Typechecking Is Not Semantic Alignment

A Lean statement can be well-typed and still be the wrong theorem.

Common failure modes

- **Too weak.** Only a special case.
- **Too strong.** Extra hypotheses, or sharper conclusion.
- **Wrong object.** Nearby but different definition.
- **Wrong scope.** Quantifier order swapped.
- **Wrong ambient structure.** Specialized to $\mathbb{R}$ instead of generic.
- **Wrong convention.** Index from 0 vs 1; open vs closed interval; left derivative vs derivative.

> Formal statement checking is not enough. We also need statement alignment.

Part III. Worked Examples

Simple to research-level alignment problems

The First n Odd Numbers Sum To n^2

Informal:

> The sum of the first n odd numbers is n^2.

Hidden choices

- n is a natural number?
- "first n" means n terms, or up to n?
- Odd numbers as $2i + 1$ or $2i - 1$?
- Index $0..n - 1$ or $1..n$?
- Sum over `Finset.range`, list, recursion?
- Equality in `Nat`, `Int`, `Real`?

A possible Lean statement:

```
example (n : Nat) :
    (∑ i in Finset.range n, (2 * i + 1))
      = n ^ 2 := by
  ...
```

This statement has already made these choices:

- `n : Nat`;
- zero-based indexing;
- first n terms are $1, 3, \ldots, 2n - 1$;
- finite sum over `Finset.range`;
- equality in `Nat`.

Set Distributivity

Informal:

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

Hidden choices

- Universe type α?
- A, B, C are `Set α`?
- Extensional set equality?
- $\cap, \cup$ as set ops or lattice ops?

```
variable {α : Type*} (A B C : Set α)

example :
    A ∩ (B ∪ C)
      = (A ∩ B) ∪ (A ∩ C) := by
  ...
```

Even this simple example forces

- a universe `α`
- objects typed as `Set α`
- set lattice notation
- extensional equality

The Inverse Of A Product

Informal:

> The inverse of a product is the product of the inverses in reverse order.

Hidden choices

- Group, monoid-with-inverses, division ring, or category?
- Multiplication commutative or not?
- Conclusion $(ab)^{-1} = b^{-1}a^{-1}$?
- In a commutative setting, reverse order is hidden; otherwise substantive.

```
variable {G : Type*} [Group G]

example (a b : G) :
    (a * b)⁻¹ = b⁻¹ * a⁻¹ := by
  ...
```

> Natural language says "product" without naming the algebraic structure. Formal language cannot use multiplication until the structure is fixed.

Every Subgroup Of A Cyclic Group Is Cyclic

Informal:

> Every subgroup of a cyclic group is cyclic.

Hidden choices

- Ambient group G.
- Multiplicative vs additive notation.
- "Cyclic" as a predicate on groups, subgroups, or modules?
- Subgroup as bundled `Subgroup G`?
- Subgroup cyclicity uses the inherited group structure?

```
variable {G : Type*} [Group G]

-- sketch shape; final mathlib API
-- packages this differently:
∀ H : Subgroup G,
  IsCyclic G → IsCyclic H
```

> "Subgroup" in natural language refers both to a subset-like object and to a group in its own right. Formalization must choose the bundled representation that exposes the needed structure.

Continuous Functions On $[a, b]$ Attain A Maximum

Informal:

> A continuous function on $[a, b]$ attains a maximum.

Hidden choices

- $a \leq b$ assumed?
- $f : \mathbb{R} \to \mathbb{R}$, or $f : X \to \mathbb{R}$?
- `Continuous f` vs `ContinuousOn f (Set.Icc a b)`?
- "Attains a maximum" means $\exists x \in [a, b], \forall y \in [a, b], f(y) \leq f(x)$?
- Is this really compact-set extreme value?

```
∃ x ∈ Set.Icc a b,
  ∀ y ∈ Set.Icc a b, f y ≤ f x
```

A formal statement here specifies:

- domain set: `Set.Icc a b`;
- codomain order: $\leq$ on $\mathbb{R}$;
- continuity predicate: `ContinuousOn`;
- compactness source: closed bounded interval.

A common theorem name often hides compactness, order, and domain restriction.

$\mathrm{Gal}(E/\mathbb{Q}) \cong Q_8$: One Statement, Several Formal Shapes

Informal:

> Let $\alpha = \sqrt{(2+\sqrt{2})(3+\sqrt{3})}$ and $E = \mathbb{Q}(\alpha)$.
> Show $\mathbf{Gal}(E/\mathbb{Q}) \cong Q_8$.

Hidden choices

- α as `Real.sqrt …`;
- E as `IntermediateField.adjoin ℚ {α}`;
- $\mathbf{Gal}(E/\mathbb{Q})$ as `↥E ≃ₐ[ℚ] ↥E`;
- Q_8 as `QuaternionGroup 2`;
- $\cong$ as `≃*` (Type, not Prop).

```
-- Shape A: explicit MulEquiv (def, not theorem)
def gal_iso_Q8 :
  (↥E ≃ₐ[ℚ] ↥E) ≃* QuaternionGroup 2 := by
  sorry

-- Shape B: Galois-group predicate
theorem q8_is_galois_group
    [MulSemiringAction (QuaternionGroup 2) ↥E] :
    IsGaloisGroup
      (QuaternionGroup 2) ℚ ↥E := by
  sorry

-- Shape C: finite Galois group object
noncomputable def q8_as_finGaloisGroup_iso :
    Efin.finGaloisGroup
      ≅ FiniteGrp.of (QuaternionGroup 2) := by
  sorry
```

Three Shapes, Three API Paths

Shape A – MulEquiv

Direct group isomorphism.

```
(↥E ≃ₐ[ℚ] ↥E) ≃* QuaternionGroup 2
```

Closest to the informal $\cong$. Must use `def` (the type is `Type`, not `Prop`).

Shape B – IsGaloisGroup

Q_8 acts faithfully on E with fixed field $\mathbb{Q}$.

```
IsGaloisGroup (QuaternionGroup 2) ℚ ↥E
```

Mathlib bridges to `MulEquiv` via `IsGaloisGroup.mulEquivAlgEquiv`.

Shape C – FiniteGrp

Package E as `FiniteGaloisIntermediateField`.

```
Efin.finGaloisGroup ≅ FiniteGrp.of (QuaternionGroup 2)
```

"Finite Galois" lives inside the object, not as a separate hypothesis.

Three reasonable formal targets for one informal isomorphism. Choice of shape changes available API and downstream proof routes.

Part IV. Modern Autoformalization Systems

Mechanisms aimed at the alignment problem

What Modern Systems Are Trying To Fix

> **Central failure mode of every NL → FL system before 2024:** the formal statement compiles, and yet is **not** the intended theorem.

The gap nobody guards

```
NL input
   ↓
LLM produces Lean
   ↓
Lean compiler accepts
   ↓
????   ← typecheck pass ≠ semantic alignment
   ↓
Theorem statement matches NL?
```

Common failure shapes seen in the wild:

- **Free variable swap.** Lean sees `pi` as a free variable; the human meant `Real.pi`.

Each Part IV paper attacks one piece

Aspect	Paper
Stronger acceptance test	Rethinking – *BEq*
Library vocabulary selection	Rethinking – *RAutoformalizer*
Concept dependency synthesis	Aria
Semantic decomposition	LoC-Decomp
Validation + repair loop	ReForm
Validation without gold reference	Roundtrip
FL → NL infrastructure (Slides 40.5–48)	Herald / LeanSearch
Project-level NL ↔ formal	LeanArchitect

Rethinking: Typecheck Is Too Weak

> *Lean accepts the generated statement*
> *does **not** imply*
> *the generated statement is the intended theorem.*

Why every existing acceptance test fails

Metric	What it measures	Failure mode
Typecheck	Lean compiles	Wrong-but-legal Lean
BLEU	Token overlap	`Real.pi` vs `pi` indistinguishable
LLM judge	Surface plausibility	Misses subtle quantifier shifts
Def. equality	Lean kernel reduces	Equivalent statements with different

Empirical comparison

Benchmark: 200 expert-labeled (NL, formal-candidate, formal-reference) triples.

Metric	Precision	Recall	Accuracy
Typecheck	35.0	100.0	35.0
BLEU	63.0	24.3	68.5
LLM majority	70.6	85.7	82.5
Def. equality	100.0	11.4	69.0
BEq	**100.0**	**72.9**	**90.5**

Reading the row:

Rethinking: Dependency Retrieval Grounds Vocabulary

> Failure mode: *agnosia of context*. The model can **write** Lean syntax; it cannot **see** what formal objects already exist around the target theorem.

The dataset the authors needed to build

```
Mathlib 4 dependency dataset:
  243,797 formal objects
  139,933 theorems
  ~ 6 M edges in the use-graph
```

Construction pipeline:

```
1. Parse Mathlib4 source tree.
2. Extract every definition + theorem.
3. Build the dependency graph
     (X depends on Y iff X's body
      mentions Y).
4. Topological sort.
5. Informalize bottom-up:
   • leaves (Nat, Set, +, …) get
     hand-curated NL labels
   • each next layer uses NL of its
     dependencies as context
```

Why plain semantic search doesn't work

```
NL query:
  "A holomorphic function whose
   image is a single value is
   constant on its connected
   domain."

BM25 ranks documents by
  frequency of {holomorphic,
                image, single,
                value, constant,
                connected,
                domain}.
```

Most relevant Mathlib API is named `Set.image`, `IsConstant`, `Complex.differentiable_on`, `IsConnected`, `OpenSet`. **None of these surface**

Rethinking: Autoformalization Improves When Dependencies Are Present

RAutoformalizer = retrieval-augmented autoformalizer. The intervention is small in code, large in effect: *put the retrieved Mathlib dependencies inside the prompt or training context*.

The prompt restructured

Old prompt:

```
Translate the following theorem into
Lean 4 / Mathlib:

  "A holomorphic function on a
   connected open set ..."

[LLM guesses Lean syntax]
```

RAutoformalizer prompt:

```
You will write a Lean 4 statement.
The following Mathlib objects are
available and relevant:
```

Oracle setting tells you where the ceiling is

```
Con-NF BEq@8 with ORACLE dependencies
  (gold list, not retrieved):
                              ~ 35 %
```

Gap from baseline to oracle:

- baseline → retrieval: +12 pp
- retrieval → oracle: +18 pp

Diagnostic: the bottleneck is not "how good the LLM writes Lean"; it is "**which formal objects sit in**

Aria: Concept Dependency Graph Before Lean Code

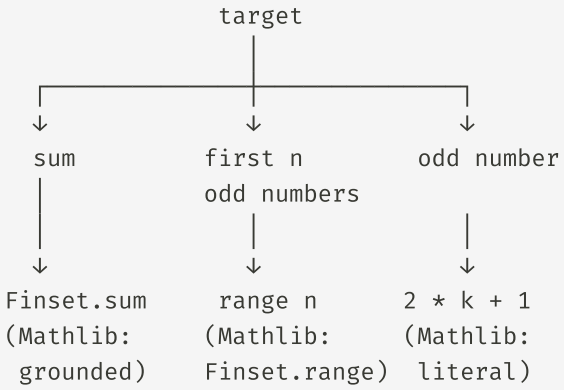
Aria's central move: never write Lean from natural language directly. First decompose the NL statement into a *graph of mathematical concepts*; ground the leaves in Mathlib; synthesize what's missing; only *then* write the theorem header.

Five failure modes Aria explicitly targets

1. **Hallucinated identifier.** Model writes `Mathlib.Foo.bar`; no such name exists.
2. **Outdated API.** `Finset.sup'` renamed; old name still memorised.
3. **Nearby-but-wrong object.** Picks `Continuous` when `ContinuousOn s` is needed.
4. **Concept missing from Mathlib.** Term like "balanced multiplicative system" has no Mathlib def — must be locally synthesized.

Walked-through example: "first n odd numbers sum to n^2"

Aria's graph (conceptual):



AriaScorer: Term-Level Grounding

> *If a generated local definition **compiles**, how do we know it is the intended **mathematical concept**?*

The problem AriaScorer solves

Aria stage [4] synthesizes a local def when Mathlib has no direct counterpart. Lean's typechecker confirms the def is *well-formed*; it cannot confirm the def captures the intended math. Stage [7] needs a different kind of check.

The mechanism

For every **term** in the generated Lean statement:

```
extract term  ──►  jixia static analysis
                     ↓
                   retrieve from
                   informalized Mathlib:
                     • name
                     • kind (def / theorem
                            / structure)
```

Three failure modes caught only by term-level grounding

1. UFD / IsDomain implicit inclusion. The phrase "*R is a UFD*" includes "R is an integral domain". A surface checker comparing string templates may flag a missing `[IsDomain R]`; AriaScorer sees that `UniqueFactorizationMonoid` in Mathlib already inherits `IsDomain`, no extra hypothesis needed.

2. Quaternion algebra: same name, different rules. Two papers can both write "quaternion algebra" but mean different multiplication tables. AriaScorer retrieves the actual definition of

LoC-Decomp: Formalization Template Exposes Commitments

> **Core move:** never ask the model for a one-shot theorem header. Enforce a *structured template* that splits the formalization into seven named sections, each forcing an explicit commitment that natural language conceals.

The seven-section template

```
1. environment imports
     (which Mathlib modules)

2. auxiliary types
     (problem-specific Type defs)

3. mathematical systems
     (the unknowns named as variables)

4. auxiliary functions
     (helper defs to talk about
      the systems)

5. constraints
     (relations the systems satisfy)
```

Worked example: "find all X such that …"

NL: *Find all positive integers n such that $n^2 + n + 1$ is prime.*

A one-shot LLM produces:

```
theorem foo (n : ℕ) :
  Nat.Prime (n^2 + n + 1) ↔ n = 1
```

– and is **wrong on at least 3 dimensions**: only $n = 1$? what about $n = 0$? what counts as "find all" – the set $\{1\}$, a predicate, or an iff?

LoC-Decomp: ASCC Back-Translation Is Structured

ASCC = Automatic Semantic Consistency Check. A structured back-translation that operates *per template section*, not on the whole file. This is what gives LoC-Decomp a feedback signal stronger than "Lean compiles".

Divide–conquer–merge pipeline

```
Lean formalization
   ↓  divide
   each template section becomes
   a semantic segment
   ↓  per-segment translate
   each segment → NL paragraph
   (uses Mathlib naming convention
    + jixia-style metadata, like
    Herald's translator)
   ↓  merge
   stitch segment paraphrases into
   one NL problem statement
   ↓  compare
   diff against original NL problem
   ↓  identify discrepancies
   per-segment list of mismatches
   ↓  classify severity
```

Two repair channels run in alternation

Semantic repair. ASCC discrepancy feedback → revise the relevant template section. Targets *wrong-theorem* failures.

Compiler repair. Lean error message → revise the relevant section. Targets *illegal-Lean* failures.

A single iteration runs Lean first; if compile fails, compiler-repair fires. If compile succeeds, ASCC fires; if ASCC flags a critical/major discrepancy, semantic-repair fires.

LoC-Decomp: ASCC Accuracy And End-To-End Pass Rates

Step 1: how good is the ASCC checker itself?

ASCC-3-MV (3-pass majority vote of the back-translation checker) vs human labels on **MATH-ASCC-Eval-150** (150 NL/Lean pairs hand-labeled by mathematicians):

Metric	Value
Precision	0.90
Recall	0.82
F1	0.86

Reading the numbers:

- **Precision 0.90** – when ASCC flags a mismatch, it is correct 9/10 times. Low false-alarm rate.

Step 2: does the template + loop improve formalization?

End-to-end pass rate with **DeepSeek-V3 + LoC-Decomp**:

	MATH-500	miniF2F
Template only, no loop	63.20 %	77.25 %
Template + iterative rectification	**84.00 %**	**90.16 %**
Δ	**+20.8 pp**	**+12.9 pp**

Two lessons from the delta:

1. **Template alone is not enough.** Even structured decomposition leaves ~20–35 % of MATH-500 problems with a mismatch.

ReForm: Reflection Loop For Semantic Fidelity

> **Core move:** the model does not produce one formal statement and stop. It produces a statement, *critiques itself*, revises, critiques again, until a stopping criterion. The entire reflection loop is **inside** the model – no external checker like ASCC.

Architecture

```
natural-language problem Q
   ↓
formal statement S₁
   ↓
semantic critique C₁
   (the model audits S₁ against Q)
   ↓
revised statement S₂
   (uses C₁ as guidance)
   ↓
semantic critique C₂
   ↓
    ...
   ↓
final answer Sₙ
```

Why a critique loop is hard

The model already wrote S_1. Asking *"is S_1 correct?"* in the same context tends to produce **shallow self-confirmation**:

> "The formal statement uses standard Lean notation. It correctly expresses the equality. The syntax is valid; the statement matches the problem." – useless. The reflection loop collapses.

A useful critique has to **cite Q's structure**, **name missing commitments**, and **make ambiguities explicit** for the next round.

ReForm: PBSO Rewards The Critique, Not Only The Answer

PBSO = Position-Based Step Optimization. A reward decomposition that pays the model separately for each turn of the reflection loop, not only for the final answer. This is what stops the loop from collapsing into shallow self-confirmation.

Why "reward only the final answer" fails

If the only signal is "does S_n pass Lean + semantic check?", the model learns to:

- generate any plausible S_1;
- emit a rubber-stamp critique ("looks correct, syntax valid");
- copy S_1 to S_n;
- collect reward when S_1 happened to be right;

Shallow vs specific critique

Penalized (shallow):

"The formal statement uses standard Lean notation. It correctly expresses the equality. The syntax is valid; the statement matches the problem."

→ no cite of Q's structure, no named missing piece, no ambiguity surfaced. $R_{\text{aux}} \approx 0.1$.

ReForm: Reported Numbers And Semantic-Error Rates In Existing Benchmarks

ReForm's main table

Semantic-consistency rate across four benchmarks (higher is better; means the model's S_n matches the NL Q after the reflection loop):

Model	miniF2F	ProofNet	Putnam	AIME
Baseline best	73.4	47.5	35.7	22.0
ReForm-8B	87.7	65.6	57.3	46.7
ReForm-32B	**91.4**	**70.4**	**62.3**	**66.7**
Δ (32B vs best)	+18.0	+22.9	+26.6	+44.7

Average lift across the four benchmarks: **+22.6 pp**.

Pattern: the harder the benchmark, the larger the lift. ReForm's reflection loop helps most where

The unexpected finding: human-written benchmarks have errors

ReForm built **ConsistencyCheck**, an 859-item expert-curated dataset for evaluating formalization quality. When they ran it on the human-written formal statements in existing benchmarks:

> **miniF2F** human-written formal statements: **16.4 %** contain semantic errors.
>
> **ProofNet** human-written formal statements: **38.5 %** contain semantic errors.

These are *gold references* the community has been training and evaluating against.

Roundtrip Verification: No Gold Formal Target

> **Core move:** drop the assumption that a "gold formal target" exists. Use a *roundtrip* NL → FL → NL → FL and check whether the two formalizations agree. Stability under roundtrip becomes the verification signal.

The setting most papers assume

```
informal statement x
gold formal target   y_gold
candidate output     y
compare              y  vs  y_gold
```

Works in benchmarks (miniF2F, ProofNet, FATE). **Does not work in real autoformalization**:

- production Mathlib developers *do not* hand-write the gold target;
- legal-code formalization *has no* Lean reference;
- novel mathematics *cannot have* a reference (the whole point is to produce one);

Why this works without ground truth

A wrong y_{orig} **usually** informalizes to a different x' than the original x, because the FL → NL step exposes the formal commitments the wrong formalization actually made. Then re-formalising the corrected x' produces a different Lean ($y_{\text{rt}} \neq y_{\text{orig}}$).

```
x = "all positive integers n such
     that n^2 + n + 1 is prime"

y_orig = "n = 1"          (wrong)
   ↓ T2
x'     = "n equals one"     (the FL only
                             asserts the
                             answer)
```

Roundtrip: Stage Diagnosis And Scoped Repair

> When $y_{\text{orig}} \not\equiv y_{\text{rt}}$, "the pipeline broke" is not actionable. Roundtrip introduces a **diagnoser** that names which of T1/T2/T3 first introduced the drift, and a **scoped repair** that fixes only the broken stage.

Three stages, three failure modes

```
T1:  informal x  →  formal y_orig
       failure: misread of x;
                missing hypothesis;
                wrong Mathlib object

T2:  formal y_orig →  informal x'
       failure: hallucinated extra
                conditions; lost
                type-class context;
                wrong quantifier order

T3:  informal x' →  formal y_rt
       failure: pipeline jitter (T3
                differs from T1 even
                on same NL); different
                model run, different
                rounding of choices
```

Scoped repair — fix only what's broken

Diagnosis	Repair scope	What gets regenerated
T1 broke	repair y_{orig}	T2 and T3 re-run from new y_{orig}
T2 broke	repair x'	only T3 re-runs
T3 broke	repair y_{rt}	nothing else regenerates

Why this matters: if you naïvely re-run the whole pipeline on every failure, you randomise T2 and T3 even when only T1 was wrong, and the loop becomes unstable. Scoping the repair gives the loop a chance to *converge*.

Mental model for classroom

Roundtrip: Reported Numbers And Wrong Fixed Points

Why the paper uses statute, not Lean

Roundtrip's evaluation is on **legal-code formalization**, not Lean math, because equivalence of two many-sorted first-order rules can be decided by an SMT solver (Z3). For Lean math, $y_{\text{orig}} \equiv y_{\text{rt}}$ may require Mathlib lemmas – sometimes a new theorem – so the methodology is cleaner in statute.

Reported numbers

Domains: **Texas Transportation Code** (150 rules), **Parks & Wildlife Code** (77 rules). Models: Claude Opus 4.6, GPT-5.2. Equivalence checker: **Z3**.

```
No repair (baseline):
    Trans.  44.7 %
    P&W     66.2 %

Claude + scoped repair:
```

The wrong fixed point pathology

Roundtrip consistency is **evidence, not proof**. The loop can stabilise on:

```
x  →  y_wrong  →  x_wrong  →  y_wrong
```

Both y_{orig} and y_{rt} misformalise *in the same way*, so the comparison succeeds; the candidate looks stable, but it is the wrong theorem.

Mitigation: *multiple paraphrase paths*. Run T2 with different prompts to produce x'_1, x'_2, x'_3; if all three re-formalise back to the same y_{orig}, confidence is higher than a single path. None of this *proves* correctness; it raises the bar a wrong fixed point has to clear.

Why The Reverse Direction Matters: Formal → Informal

> So far every paper in Part IV has worked on $\text{NL} \to \text{FL}$. The reverse direction — turning Mathlib's $\sim$200 K formal declarations back into natural-language statements — is **upstream infrastructure** for everything we have seen, not a symmetric afterthought.

1. Training data for autoformalization

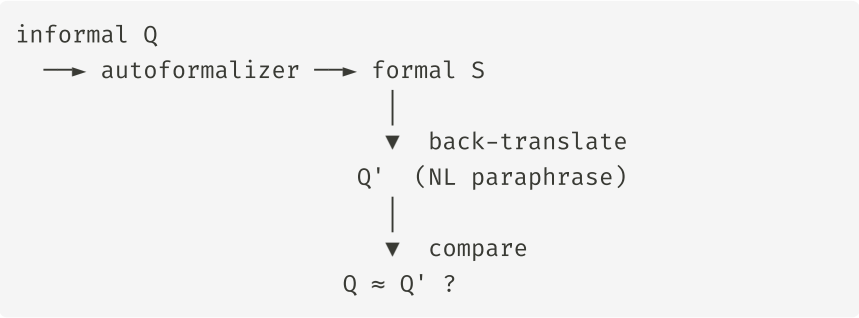
```
NL → FL model needs (informal, formal)
                    pairs to train on.

Mathlib4 ships:
  ~200 K formal declarations
  ~0     paired informal text
```

Without scalable FL → NL, the autoformalization model has no examples. Herald reports 580 k valid statements, 44 k NL-FL theorem pairs, 45 k NL-FL proof pairs — all generated by back-translation, not handwritten.

2. Retrieval and grounding

3. Verification (back-translation)



Used by ReForm (reflection critique), Roundtrip (Q vs Q_{rt}), Herald translator validation. No FL → NL → no back-translation check.

4. Library documentation

Herald: Mathlib4 To Natural-Language Data

Why naive token-reading fails

Mathlib identifiers like `Set.Ico`, `StrictMono`, `NormedAddCommGroup` are mathematician-readable only after you know the library's naming map. A bare LLM reading the Lean source reproduces the identifier; a mathematician needs the unfolded concept.

Pipeline

```
Mathlib4 declarations + proofs
  ↓
Lean-jixia static analysis:
  • names, namespaces
  • variables, types, instances
  • dependencies (used lemmas)
  • proof states between tactics
  • tactic-level call graph
  • surrounding section context
  ↓
dependency-aware informalization
  (per-term unfolding via metadata)
```

`Set.Ico` walkthrough — raw Lean:

```
theorem foo (a b c : ℝ) :
    c ∈ Set.Ico a b ↔ a ≤ c ∧ c < b
```

Naive LLM output (no metadata):

```
For real numbers a, b, c, c belongs
to the set Ico from a to b iff
a ≤ c and c < b.
```

"Ico" leaks. The natural-language reader doesn't know what `Ico` is and the translation transmits no math content.

Herald output (jixia + naming map):

```
For real numbers a, b, c, the element
c lies in the left-closed right-open
```

LeanSearch As Downstream Infrastructure

Search by natural-language query

```
NL query: "every continuous function
           on a compact interval
           attains its maximum"
  ↓
embedding model
  trained on Herald-style
  (informal, formal) pairs
  ↓
nearest-neighbour over
  informalized Mathlib
  ↓
formal declarations:
  IsCompact.exists_isMaxOn
  ContinuousOn.isCompact_image
  ...
```

LeanSearch is the deployed form of this stack — a public service at `leansearch.net`. It reappears as Aria's leaf-grounding step.

Three downstream consumers

The FL → NL chain

```
Herald
  dataset + translator
     ↓
LeanSearch
  informalized Mathlib
  as a query service
     ↓
Aria / ReProver / Rethinking
  LeanSearch-style retrieval
  inside NL → FL pipelines
     ↓
Autoformalization output
  grounded in real Mathlib
  identifiers, not hallucinated
```

Mathlib is now treated as **two parallel artifacts**: the formal source code (for compilation) and an informalized layer (for human and ML consumption). Both must be maintained as Mathlib evolves.

Herald Translator: Validation And Failure Modes

```
miniF2F-test                  96.7 %
internal graduate textbook    23.5 %
College CoT                   16.0 %
```

> Contest-style: very strong.
> Graduate / modern math: much harder.

Validation pipeline:

```
generated formal statement
  → Lean compiler check
  → back-translation to NL
  → NLI check against original
```

Failure modes with examples:

- **Subsequence.** Index map l must `Tendsto l atTop atTop`.
- **Closed subspace.** Missing "closed" $\longrightarrow$ NLI rejects; extra Y, f also rejected.
- **False negative.** Correct formal statement; back-translation flips implication direction.

What Makes Formal → Informal Hard: Three Tax

> Why does FL → NL not collapse into "ask an LLM to read Lean code"? Three independent taxes a back-translator must pay, even if the model is strong.

1. Naming-convention tax

Mathlib's identifiers are *compressed encyclopaedic labels*, not English.

```
Set.Ico   →  left-closed right-open interval [a, b)
Set.Ioo   →  open interval (a, b)
StrictMono  →  strictly increasing
IsCompact   →  compact
MeasureTheory.integral
            →  Lebesgue (Bochner) integral
Mono / Epi  →  monomorphism / epimorphism
```

A bare LLM reproduces the identifier and transmits no math. A back-translator needs a learned or extracted naming map (Herald's "jixia unfolding").

2. Implicit-argument tax

Which to surface, which to suppress:

Argument	In NL?
`[Abelian C]`	yes – "let C be an abelian category"
`[DecidableEq G]`	no – implementation detail
`{R : Type*} [CommRing R]`	yes – "let R be a commutative ring"
`{n : ℕ}` universe variables	no – invisible to the reader
`{u v : Universe}`	no – Lean-only

No fixed rule; the translator needs heuristics tied to Mathlib's design.

3. Surface-form tax

Quantifier Order: Continuous vs Uniformly Continuous

Informal: f *is continuous on* X. The natural-language phrase fixes the topic, not the relative order of the ε, δ, and base-point x quantifiers.

Shape A – pointwise: $\forall x \forall \varepsilon \exists \delta \forall y$

```
∀ x ∈ X, ∀ ε > 0, ∃ δ > 0,
  ∀ y ∈ X,
    |x - y| < δ → |f x - f y| < ε
```

δ may depend on x. This is *continuity* – the intended meaning of the natural-language phrase.

Shape B – uniform: $\forall \varepsilon \exists \delta \forall x \forall y$

```
∀ ε > 0, ∃ δ > 0,
  ∀ x ∈ X, ∀ y ∈ X,
    |x - y| < δ → |f x - f y| < ε
```

One δ works at every x. This is *uniform continuity* – strictly stronger.

Different theorems. B follows from A on compact X (Heine–Cantor), but is strictly stronger in general. Silently moving x past $\exists\delta$ rewrites the source.

Herald Case: Four Lemma, Mono Half

```
theorem
  CategoryTheory.Abelian.mono_of_epi_of_mono_of_mono
    {C : Type u_1} [Category.{v_1, u_1} C]
    [Abelian C]
    {R₁ R₂ : ComposableArrows C 3}
    (φ : R₁ ⟶ R₂)
    (hR₁ : R₁.Exact) (hR₂ : R₂.Exact)
    (h₀ : Epi (ComposableArrows.app' φ 0 ...))
    (h₁ : Mono (ComposableArrows.app' φ 1 ...))
    (h₃ : Mono (ComposableArrows.app' φ 3 ...)) :
    Mono (ComposableArrows.app' φ 2 ...)
```

The `...` are Lean-generated index proofs, not mathematical content.

A faithful informalization should restore the mathematical surface:

> **Four Lemma (monomorphism half).** Let C be abelian. Suppose a morphism between two exact composable chains of length 3, with the squares commuting. If φ_0 is epi, φ_1, φ_3 are mono, then φ_2 is mono.

Do not unfold Lean implementation details (e.g. `ComposableArrows.app'` index proofs) into the informal version.

Beyond Theorem Headers: Three Levels Of Formal → Informal

> Herald and LeanSearch target **theorem statements**. That is the easy level. FL → NL splits into three levels of increasing difficulty; each has different tooling and different open problems.

Level 1 – Statement translation (covered)

```
theorem ContinuousOn.isCompact_image
  {f : α → β} {s : Set α}
  (hf : ContinuousOn f s)
  (hs : IsCompact s) :
  IsCompact (f '' s)
```

→ "*A continuous function on a compact set has compact image.*"

Solved well by Herald-class translators on contest-style theorems (miniF2F-test 96.7 %), still hard on graduate-level statements (College CoT 16 %).

Level 2 – Proof translation

Level 3 – Definition translation

Definitions are the hardest direction.

```
class CompleteLattice (α : Type*)
  extends Lattice α, OrderTop α,
          OrderBot α where
  sSup : Set α → α
  le_sSup : ∀ s a, a ∈ s → a ≤ sSup s
  sSup_le : ∀ s a,
    (∀ b ∈ s, b ≤ a) → sSup s ≤ a
  sInf : Set α → α
  ...
```

→ "*A complete lattice is a lattice with arbitrary suprema and infima.*"

LeanArchitect: Blueprint Drift Is Alignment Drift

Earlier slides treated alignment as a property of **one theorem statement**. Big Lean projects make alignment **project-scale**: prose, declarations, dependencies, and proof status all have to stay synchronized as the project evolves.

What a blueprint looks like

Major Lean formalization projects (Polynomial Freiman–Ruzsa, Liquid Tensor Experiment, Carleson, FLT) keep a parallel **LaTeX blueprint** alongside the Lean source:

```
LaTeX blueprint (informal):
  • theorem statements in prose
  • proof sketches / outlines
  • dependency labels (def, lem,
    thm, cor cross-references)
  • Lean cross-links: \lean{ ... }
  • progress markers:
      \leanok      = formalized
      \uses{ ... }   = required deps
      \mathlibok   = upstreamed
      \notready    = needs work
```

Five drift modes between the two artifacts

When the blueprint and the Lean source drift, proof agents make mistakes:

1. **Working on a stale node.** The blueprint's NL statement was edited last week; the Lean header was not. The agent's "intended theorem" is a draft that no longer matches the prose.

LeanArchitect: `@[blueprint]` Moves Metadata Into Lean

> **Core move:** add a Lean attribute `@[blueprint ... ]` that carries the NL statement, the proof sketch, and the LaTeX label *inside the Lean file*. Lean is the single source of truth; the LaTeX blueprint is **derived**, not co-edited.

The `@[blueprint]` attribute

```
@[blueprint "thm:add-comm"
  (statement := /-- Addition in $N$
                   is commutative. -/)
  (uses      := [def:nat, lem:zero-add,
                 lem:succ-add])
  (status    := WIP)]
theorem MyNat.add_comm (a b : MyNat) :
    a + b = b + a := by
  /-- By induction on `a`, then
      \cref{lem:zero-add, lem:succ-add}. -/
  induction a with
  | zero ⇒ rw [zero_add, add_zero]
  | succ n ih ⇒
    rw [succ_add, ih, add_succ]
```

The attribute holds:

- a stable LaTeX label `"thm:add-comm"`;

Extracted automatically by the tooling

From the Lean source alone, LeanArchitect's extractor produces:

```
Lean identifier:    MyNat.add_comm
LaTeX label:        thm:add-comm
NL statement:       (the docstring)
NL proof sketch:    (the in-proof docstring)
constants used:     {Nat, HAdd, ... ,
                     MyNat.zero_add,
                     MyNat.succ_add}
proof status:       leanok | sorry-free |
                    has-sorry
mathlib status:     mathlibok |
                    pending-upstream
```

LaTeX fragments emitted

LeanArchitect Case: Multivariate Taylor And `derivWithin`

A worked end-to-end example showing `@[blueprint]` + proof-agent + human review catching a statement-level alignment bug that pure typecheck would miss. The bug shape echoes what Part I taught about `deriv` vs `derivWithin`.

The target

Multivariate Taylor theorem in integral form on a domain $D \subset \mathbb{R}^n$:

$$f(x+h) = \sum_{|\alpha|<k} \frac{D^\alpha f(x)}{\alpha!} h^\alpha + R_k(x,h)$$

with R_k written as an integral of higher derivatives along the segment from x to $x+h$.

The workflow

```
1. GPT-5 Pro
     drafts NL proof outline
```

The alignment bug exposed

When the human inspected the 3 failures, the diagnosis was *not* "the agent couldn't find a proof". The diagnosis was:

Initial formalization used `deriv` for a function on an interval $[a,b]$.

The correct formal object is `derivWithin (Icc a b)` – the derivative *restricted to* the segment. Outside the segment, `deriv` can either be 0 (junk value) or take a value from analytic continuation

Part IV Synthesis

Seven papers. **One diagnosis: typecheck is too weak.** Each paper picks one cog of the alignment machine; together they form a coherent stack from acceptance test all the way up to project-scale documentation.

Layer 1 — Better acceptance test

Going from "Lean compiles" to "statements are semantically equivalent".

- **Rethinking — BEq.** Bidirectional bounded transformation; precision 1.00.
- **Aria — AriaScorer.** Term-level grounding via jixia +

Layer 2 — Library grounding

The model cannot use Mathlib it cannot see.

- **Rethinking — Dependency retrieval.** Bottom-up informalized Mathlib + retrieval; baseline → RAutoformalizer +5–12 pp.
- **Aria — Concept dependency graph + bottom-up synthesis.** GoT

Layer 3 — Feedback loops + project scale

A single shot is rarely right; the system needs targeted feedback signals and an architecture that survives project lifetimes.

- **LoC-Decomp — Template + ASCC.** Per-segment back-translation; +20 pp from rectification loop.

Part V. Alignment Template

Fields a formal statement must pin down

Fields A Formal Statement Must Pin Down

```
Informal statement:

Objects:

Types:

Structures / typeclasses:

Definitions from library:

Quantifiers:
```

```
Assumptions:

Conclusion:

Edge cases:

Back-translation:
```

> The deliverable is not a proof. It is a formal statement whose every field above has been committed to deliberately.

Sample Informal Claims For Alignment

A. The inverse of a product is the product of inverses in reverse order.

B. A continuous function on $[a, b]$ attains a maximum.

C. Every subgroup of a cyclic group is cyclic.

D. The first n odd numbers sum to n^2.

E. The indefinite integral of f is F.

Three possible formal targets: derivative-side predicate, derivative-side set, or a specific definite-integral-with-variable-upper-limit. Each gives a different theorem.

Alignment Audit: Six Questions For Any Formal Statement