LIFT: LIBRARY INTEGRATION OF FORMALIZED THEOREMS FOR REUSABLE MATHEMATICAL KNOWLEDGE

Anonymous authors
Paper under double-blind review

ABSTRACT

Automatic formalization produces growing collections of definitions and proofs in Lean, but differences in assumptions and representations impede reuse across projects. We present LIFT, a method for jointly constructing public interfaces and adaptations that recover the original results and prescribed constructions. Existing results are aligned by their full types, common arguments are exposed through general interfaces, and concrete constructions are connected by representation maps. An asynchronous multiple-producer, single-consumer (MPSC) algorithm constructs candidates in parallel and reconciles each change with the current library. Revised candidates can use newly incorporated content; verified releases preserve earlier source recoveries as public interfaces evolve. Integration determines shared statements and adaptations before proof search addresses the remaining obligations. We organize ReasLib from the public components of 20 previously formalized developments, whose records contain 5,061 accepted integration items. Three checked cases exhibit canonical reuse, general interfaces, and explicit structural data. A promoted normal-cone interface supports a later research proof, while a recorded online revision reuses a newly accepted theorem in its source proof.

1 INTRODUCTION

Automatic formalization now reaches chapters, textbooks, and research papers (Wang et al., 2026b; Gloeckle et al., 2026; Rammal et al., 2026). Long-horizon systems maintain proof dependencies and construct definitions beyond existing libraries (Zhang et al., 2026b; Moakhar et al., 2026). As these developments accumulate, reuse depends on shared definitions, structural hierarchies, and general results, as illustrated by Mathlib (The mathlib Community, 2020). Independently formalized projects can express related mathematics under different assumptions and representations, while their proofs repeat common arguments. Building a reusable library requires finding this shared content and preserving its connections to the original mathematics.

Existing approaches address several parts of this reuse problem. Retrieval connects formalization tasks to existing premises and definitions (Yang et al., 2023; Pu et al., 2026; Wang et al., 2025). LEGO-Prover, MathlibLemma, and DreamProver construct reusable lemma collections, while LeanAgent maintains a dynamic database of premises and proofs (Wang et al., 2024; Liu et al., 2026; Zhang et al., 2026a; Kumarappan et al., 2025). Work on reusable mathematical interfaces includes abstraction learning and refactoring in Divide and Abstract, concrete realizations of abstract structures in SITA, and relation-based transport in Isabelle's Transfer and Lifting (Min et al., 2026a; Li et al., 2026; Huffman & Kunčar, n.d.; Huffman et al., n.d.). Lean Refactor optimizes existing proofs (Lu et al., 2026). Theory-level autoformalization and LCS-Bench extend the object of study to interdependent definitions and theorems (Min et al., 2026b; Feng et al., 2026), while FormalRx examines their correspondence to the source mathematics (Wang et al., 2026a). These advances motivate a joint question: which shared interface should replace project-specific content, and how does each original result follow from it? Library growth alone does not answer this question, as the LEGO-Prover case study illustrates (Berlot-Attwell et al., 2026).

Building on M2F's project-level formalization (Wang et al., 2026b), we study library integration with fixed source specifications. Its existing integration implementation supplies candidate construction and MPSC coordination. The mathematical decisions concern which public interface to

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

expose and how to recover each source through it. Independently proposed interfaces can overlap, and later arguments can reveal a more useful shared structure. An update therefore changes both the public mathematics and its connections to earlier sources. We formulate the joint construction and the recovery conditions governing its evolution through three contributions:

1. Recovery-guided integration. We formulate integration as the joint construction of public interfaces and recoveries at fixed source specifications. Typed alignment, common abstraction, and representation transport guide the choice of statements, concrete data, and connecting maps. Shared derivations must supply the original conclusions or prescribed constructions, including their defining equations and operation laws.
2. Asynchronous refinement with source recovery. Using the existing MPSC mechanism, producers refine interfaces and recoveries concurrently. One consumer reconciles proposals with the current library and checks affected source uses. Conflicts or invalid adaptations return candidates for revision; revised candidates can use newly incorporated mathematics while other eligible proposals proceed under the queue's scope constraints.
3. Online evolution with historical recovery. We specify how successive project contributions extend a shared library under accumulated source obligations. Public interface changes require re-establishing affected recoveries at their fixed specifications. Working content can inform later candidates; a shared release requires completed public and recovery proofs, dependency closure under permitted axioms, and checks of unchanged registered clients.

2 From Formalized Projects to Shared Libraries

2.1 Problem Formulation

The inputs are formalized projects $P_1, \ldots, P_n$, $n \geq 1$, with fixed versions and permitted background declarations $\mathcal{B}$. Their accepted definitions, constructions, structures, and proved theorems form D. Each source unit $u \in D$ has a fixed formal specification U_u: the original parameters, hypotheses, and conclusion for a theorem; prescribed data, defining equations, and operation laws under a declared correspondence for a construction. An arbitrary value of the same type does not preserve the prescribed data. Previously unproved statements remain separate obligations. For a theorem, U_u is its Lean statement; N_u denotes its natural-language source proposition when provided. Source specifications stay fixed throughout library construction.

The output (R, M, Π) contains a shared library, source correspondences, and recovery evidence (for u_i, we abbreviate $U_{u_i}, \rho_{u_i}, \pi_{u_i}$ as U_i, ρ_i, π_i):

$$\Pi = \{\pi_u\}_{u \in D}, \qquad \pi_u : U_u \quad \text{for every } u \in D. \tag{1}$$

The library contains public and necessary source-specific declarations; M records parameter assignments, representation maps, and adaptations. Each π_u supplies the proof or prescribed construction required by U_u. Lean checks the public content and recoveries with their dependency closure under permitted axioms (Lean, n.d.). The source-recovery requirement (1) fixes validity. Sharing is expressed by a public derivation that participates in one or more source recoveries independently of the old answers; merely retaining those answers establishes preservation.

2.2 Mathematical Challenges

Type alignment and structure selection. *(i) Full types and conversion direction.* Related statements differ in dependent parameters, universe levels, assumptions, and conclusions. Source hypotheses must justify the public assumptions, and the public conclusion must yield U_u. Restricted proof-based conversions also underlie BEq (Liu et al., 2025). *(ii) Instance inference and structure selection.* Combining projects can introduce competing implicit structures or coercions on one type. The correspondence must select the intended operations and laws through explicit parameters, bundled structures, or proved connections.

Shared arguments and usable interfaces. *(iii) Deduplication and common abstraction.* A public theorem $g \in R$ and source adaptations $\rho_i \in M$ must jointly satisfy

$$\pi_i = \rho_i(g) : U_i, \qquad i = 1, \ldots, k. \tag{2}$$

An existing general theorem may supply g; otherwise the common argument needs a suitable abstraction. *(iv) Usable interfaces and dependencies.* The shared result must expose the parameters and laws that its recoveries need. Definitions and constructions also require constructors, application equations, and rewriting laws, with their prerequisites available for composition.

Preserved data and completed derivations. *(v) Structure and concrete data.* A representation map $e : X \to Y$ must preserve the operations required by U_u. For corresponding binary operations, this includes

$$e\big(\mu_X(x_1, x_2)\big) = \mu_Y\big(e(x_1), e(x_2)\big). \tag{3}$$

Defining equations impose further laws for prescribed data. *(vi) Proof completion and actual replacement.* The resulting connections must supply complete evidence $\pi_u : U_u$, including any laws needed to return to the source representation. In the shared derivation (2), g must be independent of the old answer and its conclusion must participate in ρ_i.

3 LIFT: Asynchronous Library Integration

3.1 Joint Construction of Interfaces and Recoveries

Figure 1 follows project contributions into an evolving ReasLib. Successive materials extend each local library in the lower panel. Above, P_A and P_B refine the same result g, while P_C adds h; joint refinement, queueing, and reconciliation incorporate their proposals with source recoveries. The return arrow supplies a new release for later formalization. Language-model agents construct interfaces and recoveries together, guided by typed alignment, common abstraction, and representation transport. Input preparation preserves specifications and provenance while aligning Lean versions.

A construction state is $L = (\widetilde{R}, \widetilde{M}, \widetilde{\Pi}, W)$, where W records unresolved propositions. Integration first chooses statements, definitions, maps, and recovery expressions. If public assumptions cannot be supplied, we revise the interface or adaptation; missing data and laws enter the same construction. Terms and declared targets are checked against this joint plan. Existing proofs are reused; proof search processes W in dependency order, and validated proofs discharge obligations. A changed public statement returns to integration to recheck its source adaptations. Verified outputs lie in

$$\mathfrak{F}(D) = \{(R, M, \Pi) : \mathrm{Trust}(R, M, \Pi) \ \wedge \ \forall u \in D,\ \pi_u : U_u\}. \tag{4}$$

Here Trust requires completed proofs and dependency closure under the permitted axioms. Intermediate checks validate terms and declared obligations; a released ReasLib version satisfies (4) and supports subsequent formalization.

3.2 MPSC Integration and Online Library Evolution

Asynchronous construction coordinates mathematical prerequisites and interference between library changes. For a source dependency graph G_D, let A contain units whose required content is available. An unclaimed unit u becomes ready when $\mathrm{Pred}_{G_D}(u) \subseteq A$. Thus u_2 and u_3 in Figure 2 can be constructed concurrently after u_1, while u_4 waits for both. From a common baseline, the illustrated producers refine g_0 for u_2 and reuse it for u_3, each constructing the corresponding recovery on a private copy L_c^0. Availability follows construction; proof completion remains a release requirement.

A candidate $c = (L_c^0, \Delta_c, F_c, V_c, s_c, r_c)$ records its changes, protected scope, validation targets, queue sequence, and revision. The scope F_c covers changed files and declared source, manifest, and validation targets. Relevant drift before handoff returns construction to a fresh context; unrelated changes leave the proposal usable. With $c' \prec c$ denoting earlier queue order, incorporation is permitted for

$$\mathcal{E} = \{c \in \mathcal{Q}_{\mathrm{ready}} : \nexists c' \in \mathcal{Q}_{\mathrm{blocking}},\ c' \prec c,\ F_{c'} \cap F_c \neq \varnothing\}. \tag{5}$$

Ready candidates are queued or awaiting infrastructure retry; blocking candidates also include an ongoing merge. In Figure 2, c_2 waits behind c_1 because their protected scopes overlap. Proposals needing mathematical revision leave the blocking set, allowing other work to proceed. Repair uses the updated library, retains s_c, and increments r_c before requeueing. Several candidates can be eligible, but one consumer incorporates them sequentially.

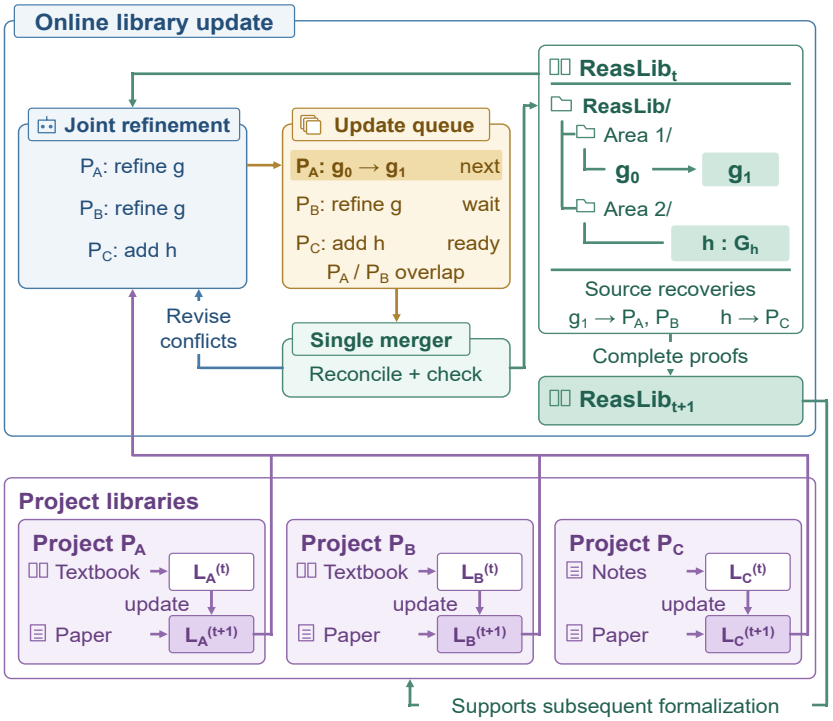

Figure 1: Online library update. Projects extend $L_p^{(t)}$ to $L_p^{(t+1)}$. The upper ReasLib panel shows working content evolving from release t. Proofs, source recoveries, and registered-client checks permit release $t+1$; updated content supports later formalization.

Reconciliation compares the copied baseline, the current state L, and the candidate version:

$$L^\star = \mathrm{Merge}_3\big(L_c^0, L, L_c^0 \oplus \Delta_c\big), \qquad \mathrm{Check}_{V_c}(L^\star). \tag{6}$$

Here $\oplus$ applies the changes. In the lower panel of Figure 2, the current library has acquired h since the baseline was copied. Reconciliation incorporates $g_0 \mapsto g_1$ while preserving h; identical edits are retained once. Type, dependency, and source-use checks precede acceptance. Failed proposals restore the pre-merge state and follow the return path for revision and requeueing; infrastructure failures retain the candidate for retry. Recorded acceptance determines whether an interrupted attempt completes or restores the preceding state.

Online construction applies this rule to successive project-library contributions. Each project imports a fixed ReasLib release while new materials extend its local library L_p. Concurrent proposals from these libraries are refined against the evolving shared context. A changed interface $g_0 : G_0 \mapsto g_1 : G_1$ also changes the recovery problem for earlier sources. If it admits a proved adaptation $\kappa : G_1 \to G_0$, those recoveries compose as

$$\rho'_u = \rho_u \circ \kappa, \qquad \rho'_u(g_1) : U_u. \tag{7}$$

Dependent interfaces require the corresponding parameter and data transports. Otherwise the proposal must re-establish the affected recoveries directly at their fixed specifications. For accumulated obligations D_t and new obligations D_p, release requires

$$\mathrm{ReasLib}_{t+1} \in \mathfrak{F}(D_t \cup D_p) \quad \wedge \quad \mathrm{ClientsOK}(R_{t+1}, M_{t+1}), \tag{8}$$

where $\mathrm{ReasLib}_{t+1} = (R_{t+1}, M_{t+1}, \Pi_{t+1})$ and ClientsOK checks registered original calls. Starting from a verified release, this condition preserves historical recovery by induction. Failed attempts leave the preceding release available. Updated working content informs later candidates; completed releases provide reusable mathematics for subsequent formalization.

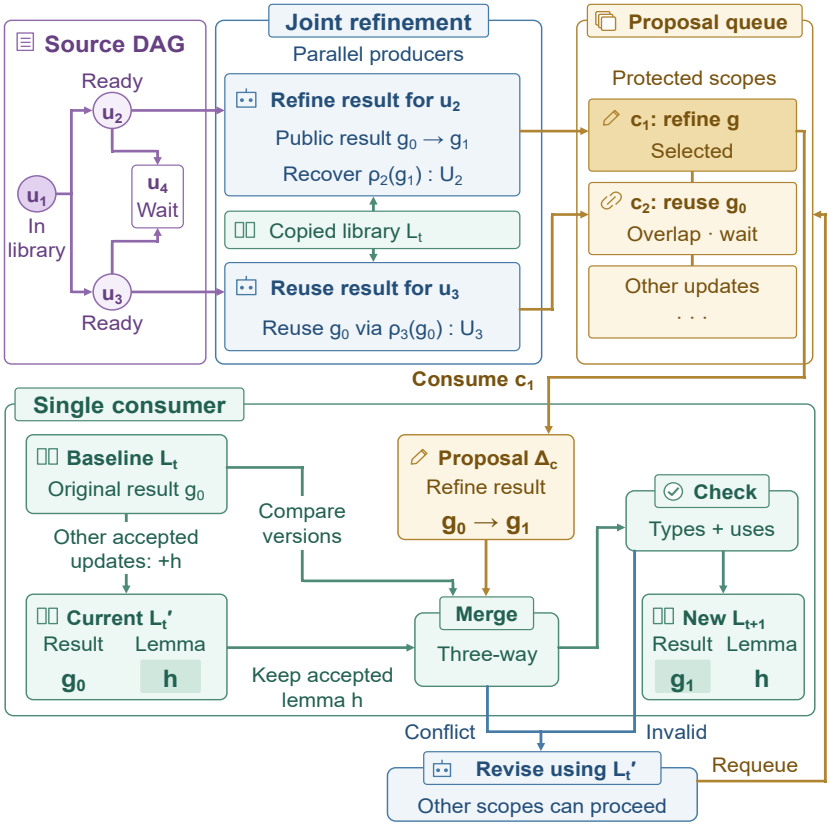

Figure 2: MPSC integration of a library fragment. Queue order constrains overlapping scopes. The consumer reconciles baseline L_t, current library L'_t, and proposal Δ_c, then checks the merged content before incorporation into L_{t+1}. The compatible refinement $g_0 \mapsto g_1$ preserves h; conflicts and invalid content return for revision.

3.3 Evidence-Guided Canonical Alignment

Figure 3 expands joint refinement in Figures 1 and 2. Its upper loop inspects declarations, jointly plans $g : G$ and recoveries ρ_i, and revises the plan using construction checks. The lower panels are coupled choices within that plan. Alignment in panel (a) adapts source assumptions to a library theorem and returns its conclusion to the source. Embedding similarity can miss mathematical equivalence (Ye et al., 2026); retrieval inspects full types, structures, defining equations, imports, and affected uses. Compiler probes test parameter assignments; recovery terms establish the link.

For $U_i = \forall a,\ H_i(a) \to Q_i(a)$ and public $g : \forall b,\ H(b) \to Q(b)$, parameter assignments θ_i and directed adaptations yield

$$\alpha_i(a) : H_i(a) \to H(\theta_i a),$$
$$\beta_i(a,h) : Q(\theta_i a) \to Q_i(a), \qquad h : H_i(a), \tag{9}$$
$$\pi_i(a,h) = \beta_i(a,h)\big(g(\theta_i a)(\alpha_i(a)(h))\big) : Q_i(a). \tag{10}$$

The assignment θ_i fixes structures and coercions, α_i supplies public assumptions, and β_i recovers the source conclusion. Definitional conversions compute directly; remaining laws enter W. These maps determine reuse, promotion, and source-specific content. Abstraction and transport complete the joint plan when the retrieved theorem supplies only part of the derivation.

3.4 Common Abstraction and Dependency Closure

Panel (b) in Figure 3 connects U_2, U_3 through shared laws $H(b)$, closes dependencies, and recovers both sources from g. Inspected derivations identify the laws; an existing general theorem can supply

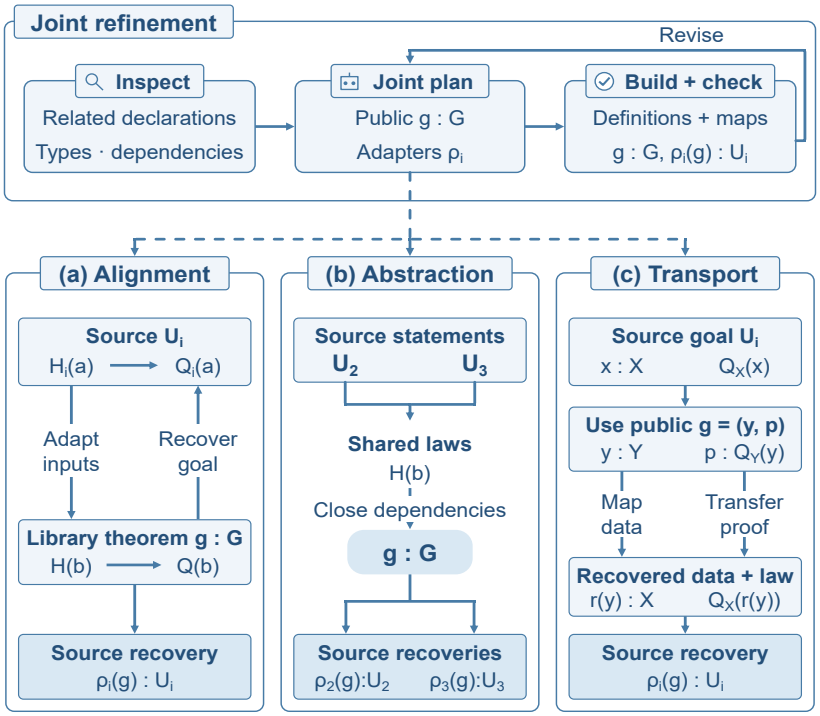

Figure 3: Three coupled refinement patterns for a public result $g : G$ and source adapters ρ_i. (a) Reuse connects a library theorem to the source obligation. (b) One abstraction recovers multiple sources. (c) A data-bearing result $g = (y, p)$ returns through $\rho_i(g) = (r(y), \beta_y(p))$. Checks guide revision; unfinished laws remain in W.

the interface, or proposed assumptions must support a common argument and follow from each source context:

$$H(b) = \bigwedge_{\ell} L_\ell(b), \qquad g : \forall b,\ H(b) \to Q(b), \qquad \rho_i(g) : U_i. \tag{11}$$

The common derivation is independent of the old source answers. Its public closure $\mathcal{C}$ includes missing constructors, application equations, and supporting lemmas, with $\mathrm{Dep}(\mathcal{C}) \subseteq \mathcal{B} \cup \widetilde{R} \cup \mathcal{C}$; Dep denotes declaration prerequisites. Available content is reused, source-specific hypotheses remain in bridges, and one-use helpers remain local. Import and consumer checks exclude cycles and reverse dependencies on the replaced source. Missing prerequisites revise the interface and its protected scope together.

3.5 Representation Synthesis and Transport

Panel (c) in Figure 3 transports $g = (y, p)$ along two branches: one returns the data, the other transfers its law. Representation synthesis determines what the public interface exposes. A property of a supplied map can remain a predicate; a construction returning a map must supply its data. Competing structures become explicit parameters, while coherent existing classes support inference.

For source data $x : X$ satisfying $Q_X(x)$, let $e : X \to Y$ map into the public representation and $r : Y \to X$ return a public construction. An object y and proof $p : Q_Y(y)$ recover the source through

$$\beta_y : Q_Y(y) \to Q_X(r(y)), \qquad \pi_u = \big(r(y), \beta_y(p)\big) : \Sigma_{x:X} Q_X(x). \tag{12}$$

The transport preserves the defining behavior and operation laws fixed by U_u in Q_X. Prescribed data x require $r(y) = x$; the round-trip law $r(e(x)) = x$ gives this equality when $y = e(x)$. Maps and returned objects are actual terms; their remaining laws enter W. Construction checks

concrete data and structure interfaces, while semantic review tests agreement with the source. A failed correspondence revises the representation and recovery within the joint plan.

4 EXPERIMENTAL EVALUATION

The evaluation connects library construction to three kinds of evidence. Historical M2F outputs describe project-level content and proof handoffs; nine named declarations and five compiled files check public interfaces and source applications; a completed Beck revision traces source-local reuse of newly accepted mathematics. We use ReasLib for the public components organized from the historical developments. Their aggregate counts measure construction outcomes, while the declaration checks and revision trace examine specific recovery mechanisms.

4.1 REASLIB CONSTRUCTION ACROSS MATHEMATICAL DOMAINS

The construction corpus comprises 20 selected mathematical developments spanning textbooks, research developments, and problem collections. Each has a retained source export and a recorded integration-stage overview. Table 1 measures coverage by accepted source items: 5,061 of 10,922 catalogue items have successful integration outcomes. The libraries contain 15,716 public-module occurrences, including inherited content and interfaces with pending proofs. These counts describe the mathematical content available within each project. Integration organizes this content through reuse, publication, local retention, and removal. The records contain 8,854 reuse decisions, 19,253 publication decisions, 2,568 local-retention decisions, and 3,283 removals, counting repeated treatment of a declaration. Publication is the largest action category in 16 of the 20 developments; reuse is largest in the number theory script, L0/L1, Riemann surfaces, and Stacks developments. Stacks records nearly equal reuse and publication counts: 4,953 and 4,915. Figure 4 shows the different profiles: L0/L1 records 112 reuse decisions and four publications, whereas Bauschke records seven and 247. These decisions determine whether source recovery uses an available result or a newly exposed public interface.

Table 1: Completed integration outputs across the construction corpus.

Development	Items	Integrated	Development	Items	Integrated
Linear/nonlinear programming	1,464	104	Monotone-operator sum	361	209
Convex optimization	1,137	243	L0/L1 optimization	18	14
Sun–Yuan optimization	661	494	BFGS convergence	21	10
Linear optimization	751	152	Symmetric pairs	547	3
ADMM development	55	15	DFP with Wolfe conditions	262	176
Nocedal optimization	808	245	Mathematical question pairs	507	82
Number theory: script	384	375	Riemann surfaces	142	70
Bauschke convex analysis	902	23	Matrices and linear algebra	83	60
Elementary number theory	431	419	Stacks development	1,730	1,722
Primes and congruences	652	641	Analysis/algebra sample	6	4
Total: 20 developments				10,922	5,061

Items counts source entries; Integrated counts successful source-item integrations.

4.2 REUSABLE INTERFACES AND SOURCE RECOVERY

Three completed cases examine which mathematical interfaces can supply their source requirements. We inspect the public declarations, their proof dependencies, and their applications at the source types. Table 2 separates the original mathematical setting from the resulting interface and its recovery. The cases cover reuse of an existing theorem, removal of a fixed coordinate representation, and exposure of concrete structure maps. For smooth first-order optimality, `IsLocalMin.fderiv_eq_zero` supplies the conclusion without a duplicate public theorem. Taylor's first interface gives $f(x+p) = f(x) + Df(x+tp)[p]$ for some $t \in (0,1)$ when $f : E \to \mathbb{R}$ is continuously differentiable on a real normed space. Setting $E = \mathbb{R}^n$ recovers the source; the integral interface retains codomain completeness. In topology, the maps act on based loop classes,

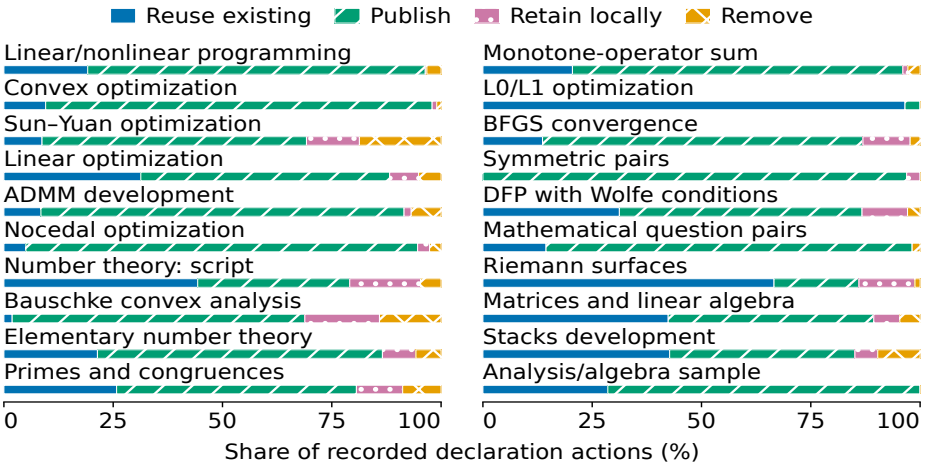

Figure 4: Integration decisions across 20 developments. Bars normalize recorded actions within each project; source and item-local retention are combined.

preserve multiplication, and are mutual inverses. At fixed basepoints, the interface removes the source's path-connectedness assumption.

The monotone-operator development shows reuse in a later research proof. Integration promotes an existing theorem on maximal monotonicity of the normal cone of a positive-radius closed ball centered at zero. Its public interface allows real normed spaces with an explicit dual pairing, provided the map to the continuous dual is surjective. The project supplies this condition for c_0, the space of sequences converging to zero, using absolutely summable sequences ℓ^1 as its dual representation. A later operator-sum proof applies the same public theorem through this pairing. The interface remains unchanged while the surrounding construction changes.

Table 2: Completed interfaces and their mathematical recovery mechanisms.

Source setting	Reusable interface	Source recovery	Checks
Smooth optimality in $\mathbb{R}^n$	Existing Fréchet derivative theorem	Instantiate the Euclidean type and local-minimum hypothesis	1
Three coordinate Taylor formulas	First- and second-order calculus on real normed spaces	Select $E = \mathbb{R}^n$ and the required scalar codomain	3
Product fundamental group	Projection P, pairing Q, and a multiplicative equivalence	Apply the maps with $QP = \mathrm{id}$ and $PQ = \mathrm{id}$	5

Checks counts the inspected declarations. All nine pass the axiom check without `sorryAx`; the associated five library and source files compile in Lean 4.32.

4.3 Proof Completion after Interface Construction

Seven linked batches trace obligations from integration to proof completion. Integration fixes public declarations and pending obligations; proof tasks can group related obligations around a shared derivation. The batches contain 118 successful integration items and 135 successful proof tasks. Identifiers or declaration/module pairs link all 249 pending obligations to successful proof-task records (Table 3). In the Taylor case, integration records three public obligations: the directional mean-value formula, integral variation of the derivative, and second-order remainder. One successful proof task lists all three obligation identifiers. The exported proofs are precisely the three Taylor declarations in Table 2, whose public module and source specializations pass the independent Lean checks. The topology handoff carries concrete maps, application equations, and inverse laws.

Table 3: Proof obligations followed from integration into successful proof work.

Development	Integration items	Proof tasks	Linked obligations
Nocedal optimization	3	3	4
BFGS convergence	10	15	56
L0/L1 optimization	14	18	6
Riemann surfaces	23	27	44
ADMM development	4	7	77
Analysis/algebra sample	4	4	4
Matrices and linear algebra	60	61	58
Total	118	135	249

Items and tasks are batch totals; linked obligations count individual handoffs to successful proof-task records. Table 2 gives independent Lean checks.

4.4 Online Reuse through Candidate Revision

We examine a completed revision in Beck's strong-convexity development to determine how an accepted interface changes a subsequent source proof (ReasBook contributors, n.d.; Beck, 2017). The input contains four modules, 20 source declarations, 15 source theorems, and 22 fixed clients. Four MPSC producer slots construct candidates with Grok-4.7 under Lean 4.30. Candidate records establish when new content becomes available; the revised proof shows how it is used. Table 4 traces the revision and its source checks. Lemma 5.20 and Proposition 5.13 begin from the same library context. The Lemma 5.20 candidate introduces a congruence theorem: pointwise equal functions have the same strong-convexity property on the source set. Proposition 5.13 returns for revision and resumes with this theorem available. The revision adds a public result for $q_m(x) = m\|x\|^2/2$ on a convex subset C of a real inner-product space. Its real-valued source companion sets $m = 1$ and uses congruence to recover the expression $\|x\|^2/2$. With `hC` proving convexity of C, the saved proof combines the interfaces as `(StrongConvexOn.mul_sq_norm hC 1).congr`; ring normalization supplies pointwise equality. The intervening update thus participates directly in the later proof.

Table 4: Completed online revision and its Beck source checkpoint.

Event	Evidence	Completed outcome
3–4	Concurrent starts	Lemma 5.20 and Proposition 5.13 use the same context
15	Interface accepted	`StrongConvexOn.congr` from Lemma 5.20
18–20	Candidate revision	Proposition 5.13 restarts with the accepted interface
24	Added by revision	`StrongConvexOn.mul_sq_norm` and its source application
24	Recovered source	Real-valued companion to Proposition 5.13
25	Original declaration types	20/20 preserved
25	Trusted source theorems	15/15 retained
25	Fixed clients	22/22 pass

Events index recorded actions. The checks cover the accumulated Beck source after the accepted revision.

5 Conclusion

Reusable library construction couples public mathematics with recoveries of the original results and constructions. We formulate them as a joint problem: aligning a theorem, exposing a common argument, or changing a representation also determines how the source requirements are recovered. MPSC coordination reconciles concurrent proposals against the evolving library; online release conditions preserve earlier recoveries through subsequent interface changes. The ReasLib collection draws on public components from 20 previously formalized developments, with 5,061 accepted integration items. Checked calculus and topology interfaces expose general results and concrete maps that supply their source requirements. In the Beck revision, a newly accepted interface becomes part of a later source proof through an explicit adaptation.

Statement on AI Usage

AI tools assisted research planning, method and experiment design, mathematical formulation and proof exposition, literature searches, evaluation scripts, extraction and aggregation of existing records, interpretation of results, and manuscript and figure preparation. Model-generated formalizations are the research outputs studied in this paper. Validation uses the stated Lean checks, original bibliographic sources, and retained records for the reported totals. The authors retain responsibility for the final content and conclusions.

Ethics Statement

The study uses formal mathematical materials. Source records identify provenance, licenses, and prior borrowing. Materials are shared under the applicable licenses.

Reproducibility Statement

Appendix A gives the algorithms, recovery conditions, worked examples, and illustrative prompts. Appendix B defines corpus selection, proof-obligation links, and the environments used for each evaluation component; Appendix C reports project-level outcomes, checked interfaces, and the recorded revision. Retained project exports and stage records support the construction counts; compiler diagnostics and declaration dependencies support the interface checks; ordered events and saved source proofs support the revision case. The anonymized repository for experimental data, the ReasLib demo, and the Lean-to-JSON and toolchain-upgrade tools is anonymous.4open.science/r/LIFT-C5F7.

References

Amir Beck. *First-Order Methods in Optimization*, volume 25 of *MOS-SIAM Series on Optimization*. Society for Industrial and Applied Mathematics, 2017. doi: 10.1137/1.9781611974997. URL https://epubs.siam.org/doi/book/10.1137/1.9781611974997.

Ian Berlot-Attwell, Tobias Sesterhenn, Frank Rudzicz, and Xujie Si. Is this LLM library learning? evaluation must account for compute and behaviour. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3534–3568. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.eacl-long.163. URL https://aclanthology.org/2026.eacl-long.163/.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. URL https://web.stanford.edu/~boyd/cvxbook/.

Yuming Feng, Frederick Pu, One An, Osbert Bastani, Li Zhang, Jiani Huang, Xujie Si, and Ziyang Li. Theory-scale auto-formalization of logics for computer science. arXiv:2606.26525v1, 2026. URL https://arxiv.org/abs/2606.26525v1.

Fabian Gloeckle, Ahmad Rammal, Charles Arnal, Rémi Munos, Vivien Cabannes, Gabriel Synnaeve, and Amaury Hayat. Automatic textbook formalization. arXiv:2604.03071v1, 2026. URL https://arxiv.org/abs/2604.03071v1.

Brian Huffman and Ondřej Kunčar. HOL theory Transfer, n.d. URL https://isabelle.in.tum.de/library/HOL/HOL/Transfer.html. Online theory documentation, accessed September 21, 2026.

Brian Huffman, Ondřej Kunčar, Cezary Kaliszyk, and Christian Urban. HOL theory Lifting, n.d. URL https://isabelle.in.tum.de/library/HOL/HOL/Lifting.html. Online theory documentation, accessed September 21, 2026.

Adarsh Kumarappan, Mohit Tiwari, Peiyang Song, Robert Joseph George, Chaowei Xiao, and Anima Anandkumar. LeanAgent: Lifelong learning for formal theorem proving. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/b67c77f8db8b991d73d6f2e16f491840-Abstract-Conference.html.

Lean. Validating a Lean proof, n.d. URL https://lean-lang.org/doc/reference/latest/ValidatingProofs/. Lean language reference, accessed September 21, 2026.

Chenyi Li, Wanli Ma, Zichen Wang, and Zaiwen Wen. SITA: A framework for structure-to-instance theorem autoformalization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40 (23):19224–19232, 2026. doi: 10.1609/aaai.v40i23.38997. URL https://ojs.aaai.org/index.php/AAAI/article/view/38997.

Qi Liu, Xinhao Zheng, Xudong Lu, Qinxiang Cao, and Junchi Yan. Rethinking and improving autoformalization: Towards a faithful metric and a dependency retrieval-based approach. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/d630537fc4402cfa3ebbc7450a0cac91-Abstract-Conference.html.

Xinyu Liu, Zixuan Xie, Amir Moeini, Claire Chen, Shuze Daniel Liu, Yu Meng, Aidong Zhang, and Shangtong Zhang. MathlibLemma: Folklore lemma generation and benchmark for formal mathematics. arXiv preprint arXiv:2602.02561v3, 2026. URL https://arxiv.org/abs/2602.02561v3.

Jialin Lu, Soonho Kong, Rodrigo Stehling, Kaiyu Yang, Zhangyang Wang, Weiran Sun, and Wuyang Chen. Lean Refactor: Multi-objective controllable proof optimization via agentic strategy search. arXiv:2605.20244v2, 2026. URL https://arxiv.org/abs/2605.20244v2.

Marcus Min, Yeqi Gao, Wilson Sy, Zhaoyu Li, Xujie Si, and Osbert Bastani. Divide and abstract: Autoformalization via decomposition and abstraction learning. In *International Conference on Learning Representations*, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/9f94298bac4668db4dc77ddb0a244301-Abstract-Conference.html.

Marcus J. Min, Mike He, Zhaoyu Li, Zixuan Yi, Sharad Malik, Aarti Gupta, Xujie Si, and Osbert Bastani. Position: The case for theory-level autoformalization. In *International Conference on Machine Learning, Position Paper Track*, 2026b. URL https://icml.cc/virtual/2026/poster/67212. Spotlight.

Arshia Soltani Moakhar, Iman Gholami, Max Springer, Mahdi JafariRaviz, and MohammadTaghi Hajiaghayi. Beyond the library: An agentic framework for autoformalizing research mathematics. arXiv:2606.31134v3, 2026. URL https://arxiv.org/abs/2606.31134v3.

Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1 (3):127–239, 2014. doi: 10.1561/2400000003. URL https://web.stanford.edu/~boyd/papers/prox_algs.html.

Lushi Pu, Weiming Zhang, Xinheng Xie, Zixuan Fu, Bingxiang He, Hengyu Zhao, Hongya Lyu, Xin Li, Jie Zhou, and Yudong Wang. MathForm: Scaling mathematical autoformalization with knowledge retrieval and verification-guided refinement. arXiv:2608.14221v1, 2026. URL https://arxiv.org/abs/2608.14221v1.

Ahmad Rammal, Niket Patel, Fabian Gloeckle, Amaury Hayat, Julia Kempe, Remi Munos, Charles Arnal, and Vivien Cabannes. Formalizing mathematics at scale. arXiv:2605.29955v1, 2026. URL https://arxiv.org/abs/2605.29955v1.

ReasBook contributors. ReasBook: Project repository, n.d. URL https://github.com/optpku/ReasBook. Software repository, accessed September 21, 2026.

The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, 2020. doi: 10.1145/3372885.3373824. URL https://leanprover-community.github.io/papers/mathlib-paper.pdf.

Lieven Vandenberghe. Conjugate functions. ECE236C lecture notes, University of California, Los Angeles, 2022. URL https://www.seas.ucla.edu/~vandenbe/236C/lectures/conj.pdf.

Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, and Xiaodan Liang. LEGO-Prover: Neural theorem proving with growing libraries. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/85dca46374dc0f27b4bb5f265b3d17f0-Abstract-Conference.html.

Haocheng Wang, Baiyu Huang, Yingjia Wan, Xiao Zhu, Xiaoyang Liu, Yinya Huang, and Zhijiang Guo. FormalRx: Rectify and eXamine semantic failures in autoformalization. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026a. URL https://arxiv.org/abs/2607.04655v1.

Zichen Wang, Anjie Dong, and Zaiwen Wen. Tree-based premise selection for Lean4. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1094. URL https://papers.neurips.cc/paper_files/paper/2025/hash/2ee86e3e9315027920ff2f205f4c3b3a-Abstract-Conference.html.

Zichen Wang, Wanli Ma, Zhenyu Ming, Gong Zhang, Kun Yuan, and Zaiwen Wen. M2F: Automated formalization of mathematical literature at scale. arXiv:2602.17016v1, 2026b. URL https://arxiv.org/abs/2602.17016v1.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 21573–21612, 2023. doi: 10.52202/075280-0944. URL https://papers.nips.cc/paper/2023/hash/4441469427094f8873d0fecb0c4e1cee-Abstract-Datasets_and_Benchmarks.html.

Jiaying Ye, Samarth Rao, Leo Carlin, Kedar Chintalapati, Saharsh Bhargava, Rachit Jaiswal, Michael Zhou, Jared Darlington, Jiahe Lu, Jarod Alper, Vasily Ilin, and Henry Kvinge. Does my embedding reflect that $A = B$? evaluating mathematical equivalence in embedding models. arXiv:2606.23959v2, 2026. URL https://arxiv.org/abs/2606.23959v2.

Youyuan Zhang, Jialiang Sun, Hangrui Bi, Chuqin Geng, Wenjie Ma, Zhaoyu Li, and Xujie Si. DreamProver: Evolving transferable lemma libraries via a wake-sleep theorem-proving agent. arXiv:2604.26311v1, 2026a. URL https://arxiv.org/abs/2604.26311v1.

Yuanhe Zhang, Yuekai Sun, Taiji Suzuki, Jason D. Lee, and Fanghui Liu. LeanMarathon: Toward reliable AI co-mathematicians through long-horizon Lean autoformalization. arXiv:2606.05400v1, 2026b. URL https://arxiv.org/abs/2606.05400v1.

A Method Details and Worked Examples

A.1 Source Specifications and Construction States

The mathematical unit of integration is a declaration together with the context that gives it meaning. A source theorem includes its universe parameters, dependent arguments, implicit structures, hypotheses, and conclusion. A source construction additionally fixes the data to be returned and the laws by which those data are observed. Declarations are grouped when their definitions are mutual or when a source item requires several declarations to state its result. Grouping changes the unit of scheduling; each original obligation keeps its own identity.

For a theorem, a useful schematic form is

$$U_u = \forall a : A_u,\ H_u(a) \to Q_u(a). \tag{13}$$

Here A_u may itself be dependent: later arguments can refer to earlier types, structures, or values. The notation H_u collects the explicit assumptions for exposition; the actual Lean type retains their individual binders and inference behavior. The source context also fixes the definitions appearing in H_u and Q_u. Replacing a definition by an unrelated one with the same result type would therefore change the specification even if the new theorem elaborated.

For a prescribed construction, write

$$U_u = \Sigma_{x:X_u} \Phi_u(x), \qquad \Phi_u(x) = \text{required equations and structure laws for } x. \tag{14}$$

Data and evidence can be returned as a dependent pair, a structure, a definition with companion theorems, or a bundled map. If the source specifies a particular map or operation, its defining behavior belongs to Φ_u. If it specifies any object satisfying a property, that property determines the permissible choice. The source fixes this distinction before integration.

Table A1 collects the objects used in the construction and coordination algorithms. The accepted output (R, M, Π) and the working state L serve different purposes. The working library can contain explicit unfinished propositions, while a release satisfies the completed-output condition (4). In both states, the source specifications remain fixed. A change to the proposed public statement changes the recovery problem, not the source target.

Table A1: Mathematical objects in construction and coordination.

Object	Meaning and role
P_p, L_p	Formalized project p and its project library; L_p^0 is its initial library.
$\mathcal{B}$	Fixed permitted background declarations and their dependencies.
u, N_u, U_u	Source unit, natural-language source when provided, and fixed formal specification; U_u is a Lean statement for a theorem.
U_i, ρ_i, π_i	Abbreviations for $U_{u_i}, \rho_{u_i}, \pi_{u_i}$ when the source units are numbered.
R, M, Π	Released library content, source correspondences, and completed recoveries.
g, ρ_u	A public result and an adaptation recovering source u from it.
$L = (\widetilde{R}, \widetilde{M}, \widetilde{\Pi}, W)$	Working content, correspondences, recovery expressions, and unfinished propositions.
$\mathcal{C}$	Proposed public declarations together with their required prerequisites.
L_c^0, Δ_c	Exact producer baseline and the candidate's immutable changes. Figure 2 writes the baseline as L_t, the current state as L'_t, and the merged state as L_{t+1}.
$c, \mathcal{Q}$	A candidate library update and the queue of such updates.
F_c, V_c	Protected file paths and ordered validation targets of candidate c; candidate subscripts c_i are abbreviated to i.
s_c, r_c	Queue sequence and candidate revision; repair preserves the former.
D_t	Source obligations accumulated by shared-library release t.

Three dependency relations appear in this setting. Source-content dependencies determine whether the mathematical prerequisites of a unit are available. Module imports determine which compiled environments are needed to elaborate that unit. References in elaborated types and terms determine the final declaration and axiom closure. A source edge can group several declaration edges, and an imported module can contain many unrelated declarations. The algorithms use the appropriate relation at each stage instead of treating all three as one exact graph.

The proposed public closure has a mathematical boundary as well as an import boundary. Its declarations may depend on permitted background libraries and other members of $\mathcal{C}$. A claimed replacement must also have a recovery that does not obtain its answer from the independent implementation being replaced. A source-specific definition needed to express the problem can remain available, while the old proof or answer-specific construction is excluded from the replacement check. This preserves the original question while testing whether the public result actually supplies the answer.

A.2 One Refinement Mechanism, Three Principles

Algorithm 1 develops a public interface and its source recoveries jointly. Its input includes the source unit, a copied library state, and a bounded construction budget. The proposal records inspected related declarations, the intended public statement or representation, the action assigned to each source declaration, and the source adaptations. Term construction and review operate on that same proposal. Alignment, abstraction, and transport can revise one another: a retrieved theorem may determine a representation, and an explicit representation may expose a common interface. Returned candidates pass construction checks; unresolved proofs stay in W.

The dashed branches from the joint plan in Figure 3 identify the mathematical choices used by this construction. In panel (a), the downward adaptation supplies the public assumptions and the upward adaptation recovers the source conclusion. Panel (b) gathers shared laws and their prerequisites before branching into source recoveries. Panel (c) separates the map returning data from the proof of its required law. All three feed the same construction-and-review loop.

Algorithm 1 Joint construction of a library extension

Require: Source unit u with fixed U_u, copied construction state L^0, budget
Retrieve related public content and inspect the available source derivations
Jointly plan g and ρ_u using alignment, abstraction, and transport
while construction budget remains **do**
    Construct definitions, maps, public statements, and source-interface adaptations
    Reuse established proofs; record remaining propositions and dependencies in W
    Check terms, scope, and declared targets; review against the plan and U_u
    if all construction checks pass **then**
        return changes Δ, protected scope F, validation targets V, and W
    end if
    Repair terms or revise the joint plan using the failed checks
end while
return incomplete status, checkpoint, diagnostics, and remaining obligations

A.2.1 Canonical Evidence and Declaration Actions

Canonical search inspects full declarations in the target, its imports, affected consumers, project libraries, and Mathlib. An inspected candidate is recorded as equivalent, related but requiring adaptation, or absent from the inspected search. Compiler probes expose elaborated types and available instances; the proposed recovery establishes the mathematical relation.

For the source and public theorem in Section 3.3, θ_i assigns the public parameters, α_i establishes its assumptions from the source hypotheses, and β_i recovers the source conclusion. Their directions are determined by use: a theorem with stronger assumptions is useful only when the source can supply them, whereas a stronger public conclusion can be weakened by β_i. If a public structure is inferred implicitly, its selected operations still have to agree with the source operations used in these maps. An explicit structure argument often makes that agreement visible.

These relations support the declaration actions in Table A2. Promotion can apply to a source theorem that is already general; abstraction does not require starting from a collection of concrete examples. Conversely, a source-specific fact may be mathematically correct and useful without belonging in the public interface. Its retention then contributes to source preservation, while later use determines whether it counts as shared content.

Table A2: Declaration actions within a joint integration proposal.

Action	Mathematical choice	Required connection
Reuse existing	Select an available public result.	Parameter and conclusion adaptations recover the source.
Promote	Give suitable source content a public interface and owner.	Public prerequisites and original uses remain available.
Retain source	Keep a fact tied to its source development.	Preserve its original specification and dependencies.
Keep local	Retain a helper used by the current construction.	Its scope follows the derivation that needs it.
Remove	Eliminate content made redundant by accepted changes.	All required source results and registered uses remain recoverable.

An equivalent declaration can often be reused by direct conversion. A related theorem can instead require a representation map, a change of parameters, or additional connecting lemmas. The proposal records which of these is intended before code repair begins. Stronger public assumptions, changed structures, or a new representation require a revised proposal and rechecked recoveries at the unchanged U_u. Ordinary elaboration repairs, such as supplying an implicit argument already fixed by the proposal, preserve its meaning.

A.2.2 Shared Laws and Public Dependency Closure

Common abstraction starts from the inspected mathematical relation, not from a prescribed number of sources. A public interface is useful when its parameters and laws support a derivation and its source adaptations can provide those laws. In (11), the conjunction $H(b)$ identifies the assumptions used by the common argument. The planner can retain an existing general theorem, weaken unnecessary assumptions, or introduce a new common statement. These are model-guided proposals subsequently checked through their terms and recoveries.

The interface includes what a caller needs to use the result. For an abstract optimizer, this may mean a constructor, an optimality law, and a stability theorem. For a quotient representation, it may mean a lifting map, a well-definedness theorem, and an application equation. The proposed closure $\mathcal{C}$ includes missing prerequisites for these operations and reuses existing public prerequisites. The condition

$$\mathrm{Dep}(\mathcal{C}) \subseteq \mathcal{B} \cup \widetilde{R} \cup \mathcal{C} \tag{15}$$

is checked in the permitted background. A valid closure also needs an acyclic declaration and import order and completed proofs.

Source bridges carry the assignments and identifications specific to a development. One-use proof helpers can remain local to a derivation. This placement keeps the public interface close to the common mathematical argument. The proposal's granularity is assessed through source recovery and actual consumers: missing application laws expand the closure, while unused public helpers can return to local scope.

A.2.3 Representations, Observable Data, and Transport

The representation decision distinguishes a property from data satisfying that property. A proposition about an already supplied function can remain a predicate. A construction that returns a function, inverse, witness, or algebraic operation must supply the corresponding term. It also decides which structure is an explicit parameter, which existing class can be inferred coherently, and whether callers need a bundled object. These choices affect both the public theorem's type and the terms available to its source bridges.

For a source representation X and public representation Y, the laws required of $e : X \to Y$ and $r : Y \to X$ are determined by U_u. An equivalence of carriers supplies inverse laws but may require further operation-preservation lemmas; a one-way map can suffice when the source asks only for a property transported in that direction. For constructed data, the recovery in (12) returns $r(y)$ with $\beta_y(p)$, and defining equations identify the result with the prescribed source behavior. For a theorem over structures, recovery applies the public result to transported parameters and converts its

conclusion. Integration supplies the concrete maps and constructors; their remaining propositional laws enter W with their owning declarations and proof prerequisites.

A.3 Validation, Proof Completion, and Recovery

Validation separates the candidate's mathematical plan, its elaborated implementation, and its completed evidence. The plan fixes declaration actions, public ownership, representation choices, source bridges, and semantic invariants. Construction implements those choices. Deterministic checks then inspect the changed scope, import direction, structure interfaces, concrete data, and declared compilation targets. Semantic review compares the actual types, terms, and changes with the plan and the source specifications. Table A3 records the evidence produced at each layer.

Table A3: Evidence used to accept a construction and a release.

Layer	Checked object and acceptance evidence
Proposal consistency	Declaration actions, public ownership, representation, source bridges, and mathematical invariants.
Candidate structure	Changed paths, permitted imports, concrete definitions, companion interfaces, and declared consumers.
Lean elaboration	Actual terms and types in the candidate's declared validation targets.
Proof handoff	Agreement between the unfinished declarations found by the scan and the recorded proof units.
Source recovery	A term at each fixed original specification, including defining laws for prescribed data.
Release trust	Completed public and recovery proofs, transitive dependency audit, and registered original-client checks.

The proof handoff records obligations by declaration rather than by placeholder occurrence: one declaration may contain several unfinished fragments, and one source item may produce several public and recovery declarations. It records their owners and dependencies, using references from both the types and bodies of elaborated constants. Proof search follows this dependency order through the constructed public interface and reuses completed proofs whose statements and dependencies remain valid. If proof work requires a change to a public statement or representation, the proposal returns to integration to reconsider its source bridges.

Release acceptance audits public outputs, recovery terms, and their transitive dependencies across the whole required output, including reused and unchanged branches. It requires completed evidence under the foundational axioms fixed in the run configuration. Intermediate checks allow declared obligations elsewhere in W.

Source recovery and original-client use are checked separately. Let $\ell : T_\ell$ be an original call in context Γ_ℓ. The client text, its assumptions, and its expected behavior remain fixed. Library-side compatibility declarations derived from M provide the original interface, giving the checks

$$\mathcal{B} + R + \mathrm{Compat}(M) + \Gamma_\ell \vdash \ell : T_\ell, \qquad \mathrm{Laws}_\ell(\ell). \tag{16}$$

Here $\mathrm{Compat}(M)$ denotes checked declarations and import interfaces implementing the source correspondences. The client laws cover applications, structure inference, conversions, computation, and rewriting. Renamed or reorganized content is exposed through this compatibility layer; editing a registered client does not establish compatibility. Candidate checks cover affected uses, and a release checks the complete frozen set.

Composition of recoveries. Suppose $\rho_u : G_0 \to U_u$ is a completed recovery whose other dependencies remain valid in the updated environment. If a revised public result $g_1 : G_1$ comes with a proved adaptation $\kappa : G_1 \to G_0$, the new recovery is

$$\rho'_u = \rho_u \circ \kappa, \qquad \rho'_u(g_1) : U_u. \tag{17}$$

The typing follows by applying κ and then the existing recovery. For dependent interfaces, the adaptation also supplies the required parameter and equality transports. For data, it preserves the defining laws in (14). This composition explains how the original obligation can stay fixed while the public organization changes. Whether a particular proposed κ exists and satisfies these laws is itself a proof obligation.

A.4 Worked Examples in Convex Analysis

Convex optimization provides a common setting for the three principles in Figure 3: choosing feasible points and minimizing convex costs. Projection returns the feasible point closest to an input; a proximal point balances a cost against staying close to that input. The following worked examples ask three different questions: which existing theorem explains a distance penalty, which argument controls two different optimizers, and how a construction changes when distance is measured differently. In each example, the public interface and the mathematical connection recovering the source requirement are determined together.

We work in $\mathbb{R}^n$, $n \geq 1$, with the Euclidean norm. The set C is nonempty, closed, and convex, and $P_C(x)$ denotes its unique nearest point to x. We take a convex cost $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ that is finite somewhere (proper) and has no upward jumps at limit points (lower semicontinuous), with $\lambda > 0$. These are the standard conditions under which adding a positive quadratic distance penalty gives a unique proximal minimizer (Parikh & Boyd, 2014; Boyd & Vandenberghe, 2004). The value $+\infty$ excludes infeasible choices: the indicator δ_C is zero inside C and $+\infty$ outside it.

A.4.1 Canonical Alignment for a Squared-Distance Penalty

A constrained optimization problem may penalize how far an input lies from its feasible set. The source theorem concerns the half squared distance $\varphi_C(x) = \frac{1}{2}\|x - P_C(x)\|^2$. It asks for its gradient and a bound on how much that gradient changes with x, even when the set has corners. The required conclusions are $\nabla\varphi_C(x) = x - P_C(x)$ and a gradient Lipschitz constant of at most one: perturbing the input cannot amplify the change in this gradient.

Suppose the public library contains a theorem about the *Moreau envelope*, the minimum cost of a regularized optimization problem. Its associated proximal map returns the minimizing point. The theorem supplies

$$\begin{aligned} e_\lambda f(x) &= \min_u \left\{ f(u) + \frac{\|x-u\|^2}{2\lambda} \right\}, \\ \nabla e_\lambda f(x) &= \lambda^{-1}(x - \operatorname{prox}_{\lambda f}(x)), \qquad \operatorname{Lip}(\nabla e_\lambda f) \leq \lambda^{-1}. \end{aligned} \tag{18}$$

The source is stated using a set and its nearest-point map; the public theorem uses a function and its minimizer. Alignment represents feasibility by $f = \delta_C$ and chooses $\lambda = 1$. Minimizing then means finding the nearest point inside C, which establishes

$$e_1\delta_C = \varphi_C, \qquad \operatorname{prox}_{\delta_C} = P_C.$$

Substituting these identities in (18) yields both source conclusions, including

$$\|(x - P_C(x)) - (y - P_C(y))\| \leq \|x - y\|.$$

In Figure 3(a), *Adapt inputs* is the assignment $\theta(C) = (\delta_C, 1)$ and its hypothesis evidence α: nonemptiness makes the indicator proper, closedness makes it lower semicontinuous, and convexity is preserved. *Recover goal* is the adaptation β through the two identities above. Together they form the source recovery ρ_C from the public theorem g. Canonical alignment reuses the analytic proof already in the library; its essential work is establishing that the public input and output describe the original distance problem.

A.4.2 Common Abstraction for Stable Optimizers

The second question concerns sensitivity: how much can an output change when its input changes? Projection maps an input to a feasible point. Softmax maps a vector of scores to a probability distribution, with temperature $\tau > 0$. Their separate stability results look different, but both constructions choose an optimizer of a linear score minus a curved cost:

$$x_h(p) = \arg\max_x \{\langle p, x\rangle - h(x)\}.$$

For projection, choose $h_Q(x) = \frac{1}{2}\|x\|^2 + \delta_C(x)$. Completing the square identifies $x_{h_Q}(p) = P_C(p)$. For softmax, let $\Delta_n = \{x \geq 0 : \sum_j x_j = 1\}$ be the probability simplex and use the entropy cost,

with $0 \log 0 = 0$:

$$h_E(x) = \begin{cases} \tau \sum_j x_j \log x_j, & x \in \Delta_n, \\ +\infty, & x \notin \Delta_n, \end{cases} \qquad [x_{h_E}(p)]_j = \frac{\exp(p_j/\tau)}{\sum_k \exp(p_k/\tau)}.$$

The shared argument uses *strong convexity*, a uniform lower bound on curvature. For a proper lower semicontinuous μ-strongly convex cost h, $\mu > 0$, the optimizer exists and is unique in this finite-dimensional setting (Vandenberghe, 2022). Its optimality condition is $p \in \partial h(x_h(p))$, where the subgradient extends the derivative to nonsmooth convex costs. Strong convexity and the Cauchy–Schwarz inequality give

$$\begin{aligned} \mu \|x_h(p) - x_h(q)\|^2 &\le \langle p - q, x_h(p) - x_h(q) \rangle \\ &\le \|p - q\| \, \|x_h(p) - x_h(q)\|. \end{aligned} \tag{19}$$

Thus $\|x_h(p) - x_h(q)\| \le \mu^{-1} \|p - q\|$: the curvature controls sensitivity to changing scores. The projection recovery supplies h_Q and $\mu = 1$. The softmax recovery supplies h_E and the valid Euclidean modulus $\mu = \tau$, yielding a bound of $1/\tau$. For the latter, the entropy Hessian is $\tau \operatorname{diag}(1/x_j) \succeq \tau I$ on the positive simplex; continuity extends the curvature inequality to its boundary, and the constrained optimality equations give the softmax formula.

Figure 3(b) expresses this abstraction decision. The *Shared laws* are optimality and strong convexity, and the public result g is the optimizer-stability theorem obtained from (19). The public interface must expose the optimizer as well as these laws so that callers can use the bound. Each source recovery ρ_i identifies its own cost and output and supplies the curvature constant. The common derivation uses neither original stability theorem; it turns their shared argument into one interface while retaining their distinct mathematical constructions.

A.4.3 Representation Transport in Weighted Geometry

The third question asks for a minimizing point when movement has different costs in different directions, as when coordinates have different scales. The source uses a weighted distance, whereas the public proximal construction uses ordinary Euclidean distance. Let E be an invertible linear map and $Q = E^\top E$, so Q is positive definite and $\|v\|_Q^2 = v^\top Q v$. The required point, denoted $\operatorname{prox}_{\lambda f}^Q(x)$, minimizes

$$f(u) + \frac{\|x - u\|_Q^2}{2\lambda}.$$

The library must supply this point together with its minimizing property.

Changing coordinates by $e(u) = Eu$ turns the weighted distance into Euclidean distance. The objective must change with the coordinates: set $\widetilde{f} = f \circ E^{-1}$ and return points through $r(v) = E^{-1} v$. For every source candidate u, the two costs agree:

$$f(u) + \frac{\|x - u\|_Q^2}{2\lambda} = \widetilde{f}(e(u)) + \frac{\|e(x) - e(u)\|^2}{2\lambda}. \tag{20}$$

Applying the public construction to $\widetilde{f}$ at Ex returns a Euclidean minimizer y. Mapping it back gives the required weighted point:

$$\operatorname{prox}_{\lambda f}^Q(x) = E^{-1} \operatorname{prox}_{\lambda \widetilde{f}}(Ex). \tag{21}$$

Choosing $f = \delta_C$ recovers weighted projection, $P_C^Q(x) = E^{-1} P_{EC}(Ex)$, connecting this construction to the nearest-point problem of the first example.

In Figure 3(c), *Map data* returns $r(y)$ and *Transfer proof* transports its optimality. To see why both are needed, write J_x for the source cost and $\widetilde{J}_{Ex}$ for the transformed cost. For every source competitor u,

$$J_x(r(y)) = \widetilde{J}_{Ex}(y) \le \widetilde{J}_{Ex}(e(u)) = J_x(u). \tag{22}$$

The middle inequality is the public minimizer's guarantee; the equalities use (20) and $e(r(y)) = y$. Invertibility preserves the assumptions on the transformed cost, and positive definiteness gives uniqueness in the source geometry. The recovery therefore returns the specified point and its characterizing law. This is the pair of data and evidence in (12), with β_y carrying the minimizing inequality back to the source.

A.5 MPSC Coordination and Candidate Revisions

Algorithm 2 separates concurrent mathematical proposal construction from incorporation into one destination library. The repository-level mechanism is a multiple-producer, single-consumer queue. Each producer works with an exact copied baseline, while the consumer reconciles a candidate with the current state before accepting it. Figure 2 illustrates both source readiness and candidate interference. They determine different parts of the algorithm.

A.5.1 Source Readiness and Protected Scopes

Let G_D be the source dependency graph and A the units whose required content is available. A unit u can be proposed when its prerequisite units are available and it has no active owner:

$$\mathrm{Ready}(u; A) \quad \Longleftrightarrow \quad \mathrm{Pred}_{G_D}(u) \subseteq A \ \wedge \ \mathrm{Unclaimed}(u). \tag{23}$$

Mutually defined declarations form one unit before this test. Availability is stage-specific: constructing a statement can use an admitted statement prerequisite, whereas a completed release needs the required proofs. Missing compiled prerequisites are restored in import order at the recorded source version. A deferred unit then retries its stage checks in that environment.

A candidate $c = (L_c^0, \Delta_c, F_c, V_c, s_c, r_c)$ records its copied baseline, changes, protected paths, validation targets, sequence, and revision. Its write set is part of F_c; source, manifest, and validation paths also contribute to the protected scope. The implementation coordinates paths, not an exact mathematical read/write set over individual declarations. Disjoint protected scopes remove one queue-order restriction, while validation still checks whether the merged library supplies the required mathematical dependencies.

Private construction does not hold the consumer lock. Checkpoint reuse requires compatibility with the source, baseline, and candidate identity. Before handoff, the changed payload is frozen and its declared scope is checked. Under a short snapshot guard, relevant current content is compared with the copied baseline. Relevant drift returns the producer to a fresh context within its budget. Unrelated library changes leave the scoped proposal usable; later reconciliation still occurs against the then-current library.

A.5.2 Eligibility and a Concrete Queue Trace

Queue order is attached to candidates, with revised successors retaining their original sequence. Table A4 distinguishes readiness from blocking. The eligibility rule (5) excludes a ready candidate whenever an earlier blocking candidate has intersecting protected paths. A consumer with no requested candidate selects the earliest eligible candidate for its validation stage. A requested candidate must pass the same eligibility test. There remains one consumer, even when several candidates are eligible simultaneously.

Table A4: Queue states and their effect on later overlapping candidates.

State	Ready	Blocks	Next action
Queued	Yes	Yes	Reconcile and validate when eligible.
Merging	No	Yes	Complete publication or restore its prepared state.
Infrastructure retry	Yes	Yes	Retry after restoring the failed attempt.
Repair required	No	No	Revise the proposal on the producer side.

The illustrative trace in Table A5 uses three proposals. Candidates c_1 and c_2 affect a common convex-analysis module; c_3 has a disjoint protected scope. During a merge, eligibility describes what could be selected after the current consumer action completes. A mathematical failure moves the candidate to repair and releases its blocking effect. This permits progress on another candidate without treating the failed proposal as accepted.

Preserving the sequence leaves intact any later candidate already accepted while repair was underway. The repaired proposal is constructed and checked against updated content. Its revision identifies the new immutable payload. Infrastructure retry remains in the ready and blocking sets

Table A5: An illustrative queue trace with overlapping candidates.

Step	Queue event	Eligible candidates
1	c_1, c_2, c_3 arrive with $s_1 < s_2 < s_3$; $F_1 \cap F_2 \neq \varnothing$.	c_1, c_3
2	The consumer begins merging c_1; it remains blocking.	c_3
3	c_1 requires source repair; the pre-merge state is restored.	c_2, c_3
4	c_2 is validated and incorporated into the library.	c_3
5	A revised c_1 arrives from the updated library, preserving s_1.	c_1^{rev}, c_3

so that an earlier interrupted operation can be retried under the original scope ordering. The rule controls proposal interference; progress depends on recoverable failures and the assigned budgets.

A.5.3 Reconciliation, Validation, and the Commit Point

For one changed path, let b be the copied content, u the current content, and a the candidate content. The reconciliation rule has the form

$$\mathrm{Reconcile}(b,u,a) = \begin{cases} a, & u = b, \\ u, & u = a, \\ \mathrm{Merge}_3(b,u,a), & \text{a compatible merge is obtained,} \\ \text{repair}, & \text{otherwise.} \end{cases} \tag{24}$$

Creation and deletion use the corresponding absent-file cases. The implementation uses a three-way text merge and a restricted append-only fallback. Mathematical validation follows this content reconciliation: the merged manifest is checked against actual paths and digests, and the declared targets are validated in order under the responsible stage.

Before applying changes, the consumer saves the current protected state, denoted L^-, and a durable preparation record. This is the state immediately before incorporation, not the producer's older baseline L_c^0. Restoring L^- preserves updates already accepted from other producers. A conflict or source-validation failure requests producer repair after restoration. A retryable infrastructure failure requests another queue attempt. This recovery rule assumes intact preparation records and successful file restoration. A restoration failure leaves the working repository unresolved and requires external recovery before that state can be reused.

Successful validation is followed by a durable receipt containing the candidate identity, revision, resulting digests, validation evidence, and next generation. The receipt is the commit point. An interruption before it requires restoration before retry; an interruption after it completes the generation and status updates forward. Repeated requests use the receipt or existing active candidate instead of creating an independent duplicate publication. Algorithm 2 includes these branches because they determine which library state later producers may observe.

A.6 Online ReasLib Releases and Historical Recovery

The repository queue advances a working state; online library construction also determines when that state becomes an accepted shared release. A project imports a fixed ReasLib version while building its local library L_i and proposes public content for incorporation against the current shared state using the same candidate rule. Public modules follow mathematical subject matter, while correspondence records retain project and source identities across releases (Figure 1). Subsequent projects can build on accepted definitions, theorems, and source connections.

Let D_t be the accumulated source obligations and D_i those contributed by the new project. An attempted update constructs a proposed interface, the new recoveries, and any adaptations needed by historical recoveries. A release is accepted when

$$(R_{t+1}, M_{t+1}, \Pi_{t+1}) \in \mathfrak{F}(D_t \cup D_i) \quad \wedge \quad \mathrm{ClientsOK}(R_{t+1}, M_{t+1}). \tag{25}$$

The client condition checks unchanged registered calls through the compatibility declarations in (16). Rejected attempts retain their diagnostics and unresolved obligations, while the preceding accepted release remains available. Working publications and accepted releases have separate identities and counts.

Algorithm 2 MPSC construction and validated incorporation

Require: Repository state L, durable queue $\mathcal{Q}$, item and recovery budgets

procedure PRODUCE(u) ▷ concurrent producers; one owner per source item
  If a receipt or active candidate exists, return its recorded status
  Defer unless $\mathrm{Ready}(u; A)$ holds; otherwise claim u
  Copy L privately; record exact baseline L^0 and compatible checkpoints
  $(\Delta, F, V, W) \leftarrow$ CONSTRUCT(u, L^0) using Algorithm 1
  If incomplete, retain the checkpoint and return failure or deferred status
  Freeze changed bytes; reject undeclared edits or a changed handoff payload
  Under a short snapshot guard, compare current $L|_F$ with $L^0|_F$
  if relevant drift is detected **then**
    Retain diagnostics; restart on a fresh copy within budget
  else
    Enqueue $(L^0, \Delta, F, V, s, r)$; repaired successors retain s and increment r
procedure CONSUME ▷ one writer for this repository
  Acquire guards with unconditional release on every exit; recover interruptions
  Complete receipted updates forward; restore and retry unreceipted updates
  Select the requested or earliest eligible $c \in \mathcal{E}$; if none, release and return idle
  Check validation-stage responsibility; mark c merging and protect F_c
  If protection fails, retain c for retry, release guards, and return
  Save the current protected state and its durable preparation record
  Reconcile (L_c^0, L, Δ_c); preserve identical changes as no-ops
  if a merge conflict occurs **then**
    Restore the saved state; mark c for producer repair, outside the blocking set
  else
    Reconcile the manifest with merged bytes; validate declared targets in order
    if source validation fails **then**
      Restore; return diagnostics for producer repair
    else if infrastructure validation fails **then**
      Restore; retain c for queue retry
    else
      Persist the receipt with published digests and the next generation
      Advance L and mark c merged; resolve later interruptions from the receipt
  Record any restoration failure; release all guards on completion or exception

If $g_0 : G_0$ is replaced by $g_1 : G_1$, each affected historical recovery can use the composition in (17). A sequence of such updates composes adaptations in reverse interface order. For maps $\kappa_j : G_{j+1} \to G_j$, an original source supported by g_0 is recovered from g_k by

$$\rho_u\big(\kappa_0(\kappa_1(\cdots \kappa_{k-1}(g_k)\cdots))\big) : U_u. \tag{26}$$

The composed chain can be replaced by a proved equivalent recovery. Dependent representations also require transport of parameters, data, and laws, with acceptance at the fixed original specification. Historical preservation then follows by induction over accepted releases: the initial release has a checked recovery for every member of D_0, and each subsequent accepted release recovers the old obligations through checked adaptations or directly re-established recoveries while providing completed evidence for the new obligations. Each accepted release thus recovers every obligation in $D_t \cup D_i$.

A.7 INPUT PREPARATION AND VERSION ALIGNMENT

Existing Lean projects enter integration through declaration extraction and, when required, toolchain alignment. Both preserve the source identity used by U_u and its recoveries. The preparation procedures make the mathematical inputs explicit before public interfaces are changed.

Lean-to-JSON conversion. Conversion starts from a project compiled under fixed Lean and dependency versions. Extraction records declaration identities, complete types, owning modules, source locations, and references from elaborated types and bodies. Related declarations form source

units with the definitions and assumptions needed to interpret them. Import relationships provide module context, while declaration references identify mathematical prerequisites, as distinguished in Appendix A.1. A schematic unit record is

$$J_u = \big(\mathrm{id}(u), U_u, \Gamma_u, \mathrm{loc}(u), \mathrm{Dep}(u), \widehat{N}_u\big), \tag{27}$$

where Γ_u is the explicit formal context and $\widehat{N}_u$ is a generated mathematical description. A language model uses the extracted facts and surrounding source context to describe the unit; mathematical review compares the description with the parameters, assumptions, conclusion, and prescribed data in U_u. An available original text N_u remains a separate source reference. Revision may change the description or grouping while retaining the declaration identity and formal specification.

The resulting units are serialized as JSON and structurally checked for required fields, consistent identities, and resolved references. Integration uses U_u for recovery and the descriptive context for retrieval and planning. For example, a Taylor unit retains its quantified space and differentiability hypotheses in U_u; its description explains the directional expansion, and its references identify the derivative definitions and supporting line-restriction results. Generality is read from the formal type.

Toolchain alignment. Toolchain alignment fixes the original environment, the target Lean/Mathlib versions, and each original specification's encoding and declaration correspondences in the target environment. A working copy is built there, and diagnostics guide repairs to imports, names, notation, structure inference, and APIs. Repeated builds check elaboration, while the original source specifications, prescribed definitions, and registered calls remain the preservation targets.

Interface preservation is checked separately: the adapted declarations must recover the fixed source requirements under explicit API and representation correspondences, and original applications are checked in the target environment. Representation changes must carry the operation's defining equations and required laws. The preparation record retains the original source, target environment, adaptations, checks, and any human intervention. These preparation changes are recorded separately, and mathematical integration starts from the resulting state.

A.8 Illustrative Role Instructions

These schematic instructions summarize the mathematical roles using the source and library contexts in Appendix B; they are illustrative examples rather than verbatim prompts from the recorded runs.

> **Plan.** "Inspect [sources] and [visible library]. Propose a public interface, source recoveries, and their required assumptions and laws."
> **Construct.** "Implement [proposal] with concrete definitions and maps. Reuse proofs and register the remaining proof obligations."
> **Review and repair.** "Check [changes] against [source specifications]. Use [Lean diagnostics] to repair terms; revise the joint plan when mathematical choices change."
> **Reuse.** "Prove [fixed target] from [visible library]. Preserve its full type and identify the public results and adaptations used."

B Corpus and Evaluation Definitions

B.1 Mathematical Developments and Evaluation Units

The inventory dated 26 September 2026 contains 61 M2F repositories: 20 representative mathematical developments, 17 related versions or historical batches, four mathematical projects without retained integration manifests, and 20 interface or coordination tests. Corpus selection follows mathematical content and artifact availability: the 20 developments have exported Lean projects and recorded integration outcomes. One representative export is retained for each selected development; related versions are recorded separately, and their outcomes are not pooled. The ReasLib collection consists of these project-level public components and their source associations, retaining their recorded environments. The three number-theory developments use their recorded project identities.

Table B1 gives the mathematical scope of each development. The optimization group includes broad textbooks and focused manuscripts: the former supply definitions and results across many chapters, while the latter supply concentrated dependencies around a particular algorithm or convergence

argument. The topology and linear-algebra projects include structured mathematical objects, such as group homomorphisms and operator representations. The Stacks development contributes a large algebraic-geometry interface collection. Together these sources exercise both theorem reuse and the placement of definitions, constructions, and auxiliary results in a public hierarchy.

Table B1: Source materials and mathematical scope of the construction corpus.

Development	Source title or material	Mathematical content
Linear/nonlinear programming	Linear and nonlinear programming material	Mathematical programming and constructions.
Convex optimization	*Convex Optimization*	Convex sets, functions, and optimality.
Sun–Yuan optimization	*Optimization Theory and Methods: Nonlinear Programming*	Optimization algorithms and analysis.
Linear optimization	*Introduction to Linear Optimization*	Linear programs and prerequisites.
ADMM development	ADMM counterexample manuscript	Operator splitting and supporting lemmas.
Nocedal optimization	*Numerical Optimization*	Differential calculus and optimality.
Number theory: script	Elementary number theory script	Arithmetic definitions and theorem dependencies.
Bauschke convex analysis	*Convex Analysis and Monotone Operator Theory in Hilbert Spaces*	Convex analysis and monotone operators.
Elementary number theory	Elementary number theory material	Arithmetic and number-theoretic results.
Primes and congruences	Primes and congruences material	Primes, divisibility, and congruences.
Monotone-operator sum	Monotone-operator sum manuscript	Monotone operators and analytic constructions.
L0/L1 optimization	*Sharp Composite Localization and Geometry-Driven Accelerated Proximal Optimization under Gradient-Dependent Smoothness*	Composite optimization and local smoothness.
BFGS convergence	BFGS convergence-order manuscript	Quasi-Newton convergence arguments.
Symmetric pairs	*Regularity and Gelfand Property for Complex Symmetric Pairs*	Regularity and representation theory.
DFP with Wolfe conditions	*A Counterexample to Global Convergence of Classical DFP under the Standard Strong Wolfe Conditions*	Quasi-Newton updates and line search.
Mathematical question pairs	Mathematical question–answer pairs	Formal questions and dependencies.
Riemann surfaces	Riemann surfaces: covering spaces	Paths, coverings, and fundamental groups.
Matrices and linear algebra	Matrices and linear algebra material	Matrix representations and operator norms.
Stacks development	*The Stacks Project*	Algebraic geometry and chapter dependencies.
Analysis/algebra sample	Analysis and algebra sample	Analytic and algebraic statements.

A *source item* is one entry in a project's formalization catalogue. It may be a definition, a theorem, or a construction, and can produce several Lean declarations. An *integration item* is a source item

with a recorded integration-stage outcome. A *public module* is a Lean file in the exported public library. A *declaration action* records how an integration decision treats one declaration. Finally, a *proof obligation* identifies a proposition that remains to be proved after its interface and ownership have been determined. These units correspond to different stages of the method: one source theorem may introduce several public results, and one proof task may complete several related obligations.

For a project P, let $\mathcal{C}_P$ be its source-item catalogue and $\mathcal{I}_P \subseteq \mathcal{C}_P$ the items recorded as successfully integrated. The construction count is $|\mathcal{I}_P|$; its catalogue-relative coverage is $|\mathcal{I}_P|/|\mathcal{C}_P|$. Public-module counts describe the exported project libraries, including inherited dependencies. Summation across projects counts occurrences, whereas declaration-level mathematical checks operate on the explicitly named interfaces.

Table B2 lists the historical exports and the two focused checks. The exports declare Lean 4.32.0 in 18 projects, 4.31.0 in Sun–Yuan, and 4.29.0 in Bauschke; 18 retain a Mathlib manifest, with the Nocedal and matrix exports lacking one. Online proposals are model-generated. Compilation, axiom inspection, source-type comparison, and registered-client checks provide the validation evidence.

Table B2: Environments and evidence for the evaluation components.

Component	Model and environment	Evidence and checks
Historical construction	Models not uniformly recorded; retained exports declare Lean 4.29/4.31/4.32.	Project exports, integration manifests, and recorded stage outcomes.
Offline interfaces	No model calls; Lean 4.32.0 and Mathlib v4.32.0.	Five compiled files and axiom dependencies of nine named declarations.
Online Beck revision	Grok-4.7; Lean 4.30.0 and pinned Mathlib; four producers.	Type, axiom, and client checks on saved proposals, proofs, and event records.

Historical environment entries describe retained exports; the corpus does not record uniform per-run model and toolchain identities. Mathlib links identify the two check environments.

B.2 Integration Outcomes and Declaration Decisions

The integration stage determines which declarations become public, which existing declarations recover a source, and which auxiliary results remain local. Its recorded success means that the interface construction and associated handoff have been accepted; propositional obligations may pass to the prover. Table 1 reports catalogue sizes and successful integrations for each development.

The five declaration decisions in Table B3 express ownership and reuse. *Reuse existing* selects an available declaration as the public mathematical result. *Publish* places a declaration in a public module. *Remove* eliminates a declaration from the proposed representation. *Retain in source* keeps it in the source development, while *Retain locally* keeps it within the item's local context. Declarations can recur across records, and publication may leave proof obligations.

Table B3: Declaration decisions measured in the retained integration records.

Decision	Recorded mathematical role	Occurrences
Reuse existing	Recover the source through an available interface	8,854
Publish	Assign a declaration to a public module	19,253
Remove	Eliminate a declaration during refinement	3,283
Retain in source	Keep a source-specific declaration with its source	2,001
Retain locally	Keep an item-specific helper within its local context	567
Total		33,958

For a project P and decision a, write $N_{P,a}$ for its number of recorded occurrences. Figure 4 uses $N_{P,a}/\sum_b N_{P,b}$ to compare decision profiles across projects of different sizes. The two local-retention labels are combined in the plot and kept separate in Table C3. Table C2 records the available public-module context.

B.3 Following Obligations into Proof Work

The proof-handoff study selects seven previously identified batches with linkable statement, integration, and subsequent proof records. We start from obligations marked pending at integration and locate their later proof tasks. A persistent identifier gives the primary match. When an identifier changes, both the declaration name and its owning module must agree. Matching identifies individual obligations; completion status belongs to the proof task.

Let $\mathcal{W}_b$ denote the pending obligations in integration batch b, and let $\mathcal{J}_b^{\mathrm{succ}}$ denote its successful proof tasks. Each task j supplies a set $\mathcal{O}_j$ of obligations. The reported linkage count is

$$H_b = \left|\{w \in \mathcal{W}_b : \exists j \in \mathcal{J}_b^{\mathrm{succ}},\ w \text{ matches an obligation in } \mathcal{O}_j\}\right|. \tag{28}$$

We count each integration obligation once, even when more than one later record mentions it. This definition lets a source result expand into several proof obligations and lets a proof task solve related statements together. The selected batches span numerical optimization, quasi-Newton methods, composite optimization, topology, and linear algebra. Selection requires linked integration and subsequent proof records. Persistent identifiers match 246 obligations; declaration/module pairs match three further obligations concerning matrix representation, a positive-definite matrix norm, and an operator-norm characterization. Table C4 gives the batch-level counts and matching routes.

B.4 Checking Mathematical Interfaces in Lean

The mathematical case study checks public declarations and their source applications. We select numerical optimization for canonical theorem alignment and general calculus interfaces, and topology for a construction with explicit structure-preserving maps. The checks use the exported Lean files and their declared mathematical dependencies. We compile the two public modules and three associated source files under Lean 4.32, and inspect the axiom dependencies of the nine named declarations in Table C5.

A declaration passes the dependency check when its proof or definition closes without `sorryAx` and uses only the recorded standard Lean axioms. All nine inspected declarations depend only on `propext`, `Classical.choice`, and `Quot.sound`. The source files separately check that the public results elaborate at the intended source types. For the optimization examples, this includes specialization to real Euclidean space; for the topology example, it includes the forward map, inverse map, and multiplicative equivalence at the chosen basepoints.

The type interpretation matters for the first-order optimality case. The source concerns a smooth function near a local minimum, and the public theorem is expressed with Lean's totalized Fréchet derivative. Under the source's differentiability assumptions, this derivative is the ordinary derivative and vanishes at the minimum. We evaluate recovery of that smooth source statement. For Taylor's integral identity, the derivative takes values in a space of continuous linear maps, so the codomain and completeness assumptions are retained when interpreting the formula.

The normal-cone case follows a public interface across retained versions of the monotone-operator development. Manifest `INF-044` records its promotion to `ReasLib/Analysis/Convex/NormalCone/ClosedBall.lean`; the source specialization and the later proof `Lorentz.exists_seedCounterexample` establish its uses. Four core modules are unchanged between these versions. The later development has a successful Lean 4.32 build, with targets specified in `.github/workflows/lean_action_ci.yml`. Its `docs/comparator-verification.md` reports checks of three exported interfaces and their transitive proof dependencies under the permitted standard axioms.

B.5 Online Revision and Source Checks

The online case selects the completed Beck revision at event 25 of a recorded integration run. The ReasBook source concerns strong convexity and squared norms (ReasBook contributors, n.d.; Beck, 2017). Its four modules contain 20 original declarations, 15 proved theorems, and 22 registered clients; background declarations supply the initial public context. Preservation checks compare the original declaration types with the updated source, inspect theorem dependencies under the permitted axioms, and compile the registered original calls. Their denominators are the three corresponding source counts. These measurements cover the Beck source checkpoint after the accepted revision.

The run uses Grok-4.7, Lean 4.30, and a fixed Mathlib revision, with four producers and one consumer. The ceilings are two million tokens, 200 model calls, and 90 minutes per update, with four design rounds, three Lean-repair rounds per design, and two conflict revisions. We follow Lemma 5.20 and Proposition 5.13 from their common initial context through the accepted revision and Beck checkpoint.

B.6 Evidence of Reuse in a Revision

Reuse is established by three observations: the public interface is accepted, it is available when revision begins, and it occurs in the revised source proof. Candidate records give the event order; the saved proof identifies the specialization and expression conversion that recover the source requirement. Appendix C.5 shows this chain for `StrongConvexOn.congr`. The surrounding Beck checkpoint checks source preservation. Event indices measure execution order, not elapsed time.

C Supplementary Experimental Results

C.1 Construction Coverage and Public Organization

Table C1 groups the 20 developments in Table 1 by mathematical domain. The source catalogues locate accepted integration outputs within textbooks, research developments, and focused collections.

Table C1: Project-level construction of ReasLib across mathematical domains.

Domain	Projects	Source items	Integrated	Modules
Optimization	9	5,177	1,453	2,187
Number theory	3	1,467	1,435	556
Convex analysis	2	1,263	232	181
Representation theory	1	547	3	5
Mixed mathematics	2	513	86	61
Topology	1	142	70	57
Linear algebra	1	83	60	21
Algebraic geometry	1	1,730	1,722	12,648
Total	20	10,922	5,061	15,716

Integrated counts accepted source items. Modules are public files retained in project libraries, including inherited content; counts are summed over projects.

Public modules combine inherited context with newly organized content (Table C2). The BFGS snapshot retains 638 public modules alongside ten integrated source items; the Stacks snapshot contributes 12,648 of the 15,716 module occurrences.

C.2 How Integration Redistributes Declarations

Table C3 gives the five declaration decisions for every development. Publishing is the largest category, with 19,253 occurrences, followed by reuse of existing declarations, with 8,854. Local retention contributes 2,568 occurrences across source-level and item-level contexts. The 3,283 removal decisions record declarations eliminated during interface refinement. The outcomes therefore include selecting existing public results, introducing public interfaces, and retaining supporting material where it is needed.

The topology project records 281 reuse decisions and 82 publications. The product-group case below examines one of its interfaces, where explicit maps combine existing constructions. The Stacks development contributes 4,953 reuse decisions and 4,915 publications, combining both actions at a larger scale in Figure 4.

C.3 Detailed Proof Handoffs

Table C4 expands the seven linked batches used in the main text. They contain 118 successful statement-stage items, 118 successful integration items, and 135 successful proof tasks. All 249

Table C2: Retained integration records and project-level public libraries.

Development	Integration records	Public modules
Linear/nonlinear programming	104	69
Convex optimization	243	247
Sun–Yuan optimization	494	385
Linear optimization	152	135
ADMM development	15	26
Nocedal optimization	245	213
Number theory: script	375	220
Bauschke convex analysis	23	29
Elementary number theory	419	109
Primes and congruences	641	227
Monotone-operator sum	209	152
L0/L1 optimization	14	42
BFGS convergence	17	638
Symmetric pairs	3	5
DFP with Wolfe conditions	176	432
Mathematical question pairs	82	59
Riemann surfaces	64	57
Matrices and linear algebra	60	21
Stacks development	1,791	12,648
Analysis/algebra sample	4	2
Total	5,131	15,716

Integration records count manifest files in the exports; successful integration counts source items in the recorded stage overviews. Public modules include inherited mathematical content.

Table C3: Declaration decisions by mathematical development.

Development	Reuse	Publish	Remove	Source	Local
Linear/nonlinear programming	106	427	18	2	0
Convex optimization	191	1,769	20	21	0
Sun–Yuan optimization	721	5,000	1,544	556	445
Linear optimization	274	498	45	51	7
ADMM development	31	304	25	6	0
Nocedal optimization	101	1,801	53	55	0
Number theory: script	594	466	63	216	1
Bauschke convex analysis	7	247	52	21	42
Elementary number theory	260	792	70	92	1
Primes and congruences	563	1,198	192	228	3
Monotone-operator sum	161	594	22	8	1
L0/L1 optimization	112	4	0	0	0
BFGS convergence	24	129	4	19	0
Symmetric pairs	0	63	0	2	0
DFP with Wolfe conditions	369	656	34	106	17
Mathematical question pairs	40	232	5	0	0
Riemann surfaces	281	82	5	22	32
Matrices and linear algebra	64	71	7	9	0
Stacks development	4,953	4,915	1,124	587	18
Analysis/algebra sample	2	5	0	0	0
Total	8,854	19,253	3,283	2,001	567

Source and Local denote the two retention decisions. Counts are decision occurrences in the retained records, including repeated treatment of a declaration.

obligations recorded for proof at integration are linked to successful proof-task records. The totals distinguish batch-level stage items and tasks from individual obligation links.

The batches differ in how proof obligations are distributed across source items. ADMM and BFGS account for 133 of the 249 linked obligations (53.4%), with 14 of the 118 integrated items (11.9%). The matrix batch contains 60 integrated items and 58 linked obligations. An integrated source requirement can give rise to several separately stated propositions.

Table C4: Obligation-level links between integration and successful proof work.

Development	Statement items	Integrated items	Proof tasks	Linked obligations	ID links	Name/module links
Nocedal optimization	3	3	3	4	4	0
BFGS convergence	10	10	15	56	56	0
L0/L1 optimization	14	14	18	6	6	0
Riemann surfaces	23	23	27	44	44	0
ADMM development	4	4	7	77	77	0
Analysis/algebra sample	4	4	4	4	4	0
Matrices and linear algebra	60	60	61	58	55	3
Total	118	118	135	249	246	3

Statement, integration, and proof counts cover successful items at their respective stages. The final two columns identify the matching route for each linked obligation.

The three declaration/module matches are `LinearMap.exists_matrix_mulVec` in the matrix-representation module, `Matrix.PosDef.norm_eq_sqrt_dotProduct_mulVec` in the positive-definite-matrix module, and `ContinuousLinearMap.norm_isGreatest_unitBall` in the finite-dimensional-operator module. Their names and owners coincide across the integration and proof records. The remaining 246 obligations retain their identifiers directly.

Two batches show why the method fixes interfaces before delegating proofs. In the Taylor case, one source item generates three public calculus obligations, which are completed by one proof task. The shared development of line restrictions and derivative identities can therefore support all three public results. In the product fundamental-group case, integration constructs the concrete maps and records their application equations and inverse laws as proof obligations. Grouping these obligations around the intended public object connects the resulting laws to the data that clients use.

C.4 Completed Mathematical Interfaces

The calculus and topology cases come from the exported project libraries. They complement the illustrative convex-analysis examples in Appendix A with completed interfaces and checked source applications. Table C5 lists the nine declarations whose dependencies were inspected. Both public modules and all three associated source files compile; each listed declaration closes using only the standard Lean axioms recorded in Appendix B.4. The final case follows a normal-cone interface into a later research proof.

Table C5: Public declarations checked in the mathematical case study.

Declaration	Mathematical role
`IsLocalMin.fderiv_eq_zero`	Canonical first-order condition
`Taylor.existsFirstOrder`	Directional mean-value formula
`Taylor.fderivAddEqIntegral`	Integral variation of the derivative
`Taylor.existsSecondOrder`	Second-order Taylor remainder
`FundamentalGroup.prodProjectionHom`	Projection homomorphism
`FundamentalGroup.prodPathHom`	Pairing homomorphism
`FundamentalGroup.prodPathHom_leftInverse`	Pairing after projection
`FundamentalGroup.prodPathHom_rightInverse`	Projection after pairing
`FundamentalGroup.prodMulEquiv`	Multiplicative equivalence

C.4.1 Canonical Alignment in First-Order Optimality

In smooth optimization, a local minimum cannot be improved by a sufficiently small movement along any direction. Its first-order change must therefore vanish in every direction. The source states this condition in real Euclidean space with n coordinates. The library already provides `IsLocalMin.fderiv_eq_zero`, expressed using a local-minimum predicate and the Fréchet derivative. Canonical alignment reuses this theorem at the Euclidean source type.

Writing U_u for the source statement and G for the public theorem type, the recovery specializes the type parameters of G to $\mathbb{R}^n$ and uses the source's local-minimum hypothesis. Under the source smoothness assumptions, the recovered conclusion is

$$x_\star \text{ is a local minimum of } f \quad \Longrightarrow \quad Df(x_\star) = 0, \qquad f : \mathbb{R}^n \to \mathbb{R}. \tag{29}$$

The checked source application recovers the original optimality statement from this Mathlib proof.

C.4.2 General Interfaces for Taylor Formulas

In differential calculus, Taylor formulas describe how a function and its derivative change when the input moves from x to $x + p$. The numerical-optimization source contains three such formulas. The public module `ReasLib.Analysis.Calculus.Taylor` expresses them using continuous linear maps on real normed spaces. This removes the fixed coordinate dimension while retaining the differentiability and completeness conditions needed by each result. Choosing the source's Euclidean space recovers its original formulas from these general interfaces.

For a continuously differentiable scalar-valued function $f : E \to \mathbb{R}$ on a real normed space, the first interface provides

$$\exists t \in (0, 1), \qquad f(x + p) = f(x) + Df(x + tp)[p]. \tag{30}$$

The declaration `Taylor.existsFirstOrder` exposes this result independently of the source dimension. Its proof uses the one-variable restriction of f to the line through x in direction p. For this public result $g : G$, the recovery $\rho_u(g) : U_u$ specializes the space parameter to Lean's `EuclideanSpace` over $\mathbb{R}$ indexed by `Finn`. The saved source checks this application at its full original type, including the `ContDiff` hypothesis and quantified x, p. The axiom audit inspects the public proof; source elaboration checks its specialization at U_u.

The second interface considers a twice continuously differentiable map $f : E \to F$, where F is a complete real normed space. It describes how the derivative changes along the same segment:

$$Df(x + p) = Df(x) + \int_0^1 D(Df)(x + tp)[p]\, dt. \tag{31}$$

Here $Df(x)$ is a continuous linear map from E to F, and the integrand in (31) has that same type. The public declaration `Taylor.fderivAddEqIntegral` therefore represents an operator-valued identity. Its interface preserves the codomain completeness required for integration. The source recovers the scalar-valued finite-dimensional formula by setting $E = \mathbb{R}^n$ and $F = \mathbb{R}$.

The third interface gives a second-order Taylor identity for a twice continuously differentiable scalar-valued f:

$$\exists t \in (0, 1), \qquad f(x + p) = f(x) + Df(x)[p] + \tfrac{1}{2} D^2 f(x + tp)[p, p]. \tag{32}$$

The declaration `Taylor.existsSecondOrder` uses the same general differential structure. Its source specialization restores the coordinate setting without changing the public theorem. The integration record assigns a distinct pending-obligation identifier to each of the three declarations. A single later successful proof task lists those same three identifiers, connecting the planned interfaces to the exported proofs. These are also the three Taylor declarations in Table 2: their proof dependencies contain no `sorryAx`, and both the public module and the source specializations compile.

C.4.3 Explicit Transport for Product Fundamental Groups

In topology, the fundamental group describes closed paths from a fixed point back to itself. Paths represent the same element when one can be continuously deformed into the other while keeping the endpoints fixed; the group operation follows one path after another. The source asks how this

structure behaves for a product space. Let X and Y have fixed basepoints x and y. Projection sends a product loop to its two component loops, and pointwise pairing reverses this construction. Integration supplies the corresponding maps and their algebraic laws through the group homomorphisms

$$P : \pi_1(X \times Y, (x, y)) \longrightarrow \pi_1(X, x) \times \pi_1(Y, y), \qquad Q : \pi_1(X, x) \times \pi_1(Y, y) \longrightarrow \pi_1(X \times Y, (x, y)). \tag{33}$$

The forward map is `prodProjectionHom`; the inverse construction is `prodPathHom`. Their public inverse laws establish

$$Q(P(\gamma)) = \gamma, \qquad P(Q(\alpha, \beta)) = (\alpha, \beta). \tag{34}$$

Together with preservation of multiplication, these laws are packaged as `prodMulEquiv`, giving

$$\pi_1(X \times Y, (x, y)) \simeq_* \pi_1(X, x) \times \pi_1(Y, y). \tag{35}$$

The interface provides the maps themselves, so a client can transport elements and use their algebraic laws. The source file checks the projection, pairing, inverse identities, and final equivalence. The public formulation requires fixed basepoints but does not require path connectedness, making it directly applicable beyond the source's path-connected setting. The five checked declarations verify the construction and its laws as a single usable mathematical interface.

C.4.4 Normal-Cone Interfaces in a Research Proof

This convex-analysis example concerns an operator-sum proof on an infinite-dimensional space. A normal cone describes the supporting functionals of a convex constraint. For real normed spaces X, Y, the public pairing supplies a continuous linear injection $J : Y \to X^*$ into the continuous dual. At $x \in C$, its normal cone consists of $y \in Y$ such that $J(y)(z - x) \leq 0$ for every $z \in C$; it is empty outside C. Monotonicity requires $J(y - y')(x - x') \geq 0$ between graph points. A monotone graph is maximal when no larger graph has this property. The public theorem gives

$$J \text{ is surjective}, \quad r > 0 \quad \Longrightarrow \quad N^J_{\overline{B}_X(0,r)} \text{ is maximally monotone.} \tag{36}$$

The public theorem is `DualPairing.maximalMonotone_normalConeGraph_closedBall`. The integration record promotes it to the normal-cone module and records its proof obligations. Its prerequisites provide the pairing, monotonicity, and normal-cone definitions. The interface requires Y to represent every continuous linear functional on X.

The concrete development takes $X = c_0$, the real sequences converging to zero, and $Y = \ell^1$, the absolutely summable real sequences. The coordinate pairing $J(a)(x) = \sum_n x_n a_n$ supplies the continuous dual map. The representation theorem `C0Seq.coordinateDualPairing_surjective` proves surjectivity, allowing the general normal-cone result to apply to these sequence spaces. Table C6 separates the public parameters from their concrete realization. The retained source checks the original conclusion at $r = 1/2$. The later proof `Lorentz.exists_seedCounterexample`, in `SeedCounterexample.lean`, applies the same public theorem at $r = 12$ to establish maximal monotonicity of its normal-cone component:

```
exact C0Seq.coordinateDualPairing.maximalMonotone_normalCone
Graph_closedBall
  C0Seq.coordinateDualPairing_surjective 12 hr
```

Here `hr` proves positivity of the radius. The dual-pairing, monotonicity, coordinate-pairing, and closed-ball modules are unchanged between the two retained versions.

Table C6: A public normal-cone theorem supplies a research proof.

Public requirement	Concrete realization	Role in the proof
Real normed spaces X, Y	c_0 and ℓ^1	Supply the primal and dual representations
Continuous dual map J	$J(a)(x) = \sum_n x_n a_n$	Connect sequence data to functionals
Surjectivity of J	Every functional has an ℓ^1 representation	Discharge the public theorem's hypothesis
Positive radius r	$1/2$ in the source; 12 in the later proof	Recover the required closed-ball result

C.5 A Revision that Reuses Newly Accepted Mathematics

The completed Beck revision shows how library context changes a source proof. Two candidates, Lemma 5.20 and Proposition 5.13, start from the same library context. Lemma 5.20 is accepted first and introduces `StrongConvexOn.congr`. Proposition 5.13's initial candidate returns for revision. Its next proposal starts from the library version that already contains the congruence theorem. Figure C1 and Table C7 show the recorded event order.

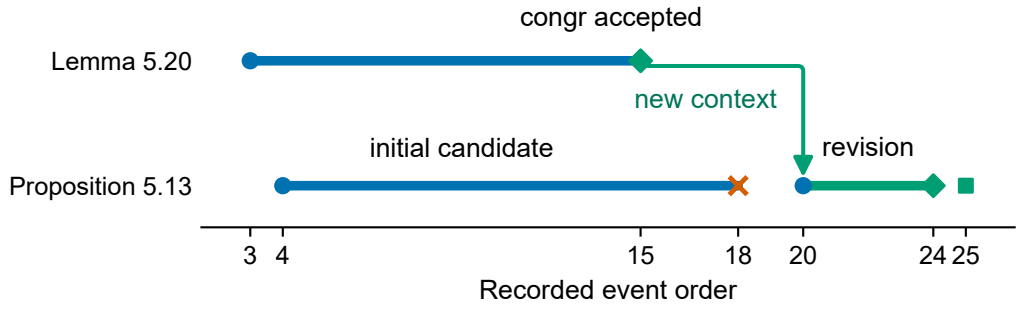

Figure C1: A congruence interface enters Beck's revised squared-norm proof. Diamonds mark acceptance, the cross a return for revision, and the square a passing source check.

Table C7: Recorded events linking an accepted interface to a later revision.

Event	Recorded action	Mathematical effect
3	Start Lemma 5.20	Construct a candidate from the initial library context.
4	Start Proposition 5.13	Construct a second candidate from the same context.
15	Accept Lemma 5.20	Add the strong-convexity congruence interface.
18	Return initial Proposition 5.13	Keep its proposed change outside the accepted library.
19	Request revision	Return the candidate for refinement.
20	Start revised Proposition 5.13	Use the library containing the accepted congruence theorem.
24	Accept revised Proposition 5.13	Add a general squared-norm interface and its source application.
25	Check the Beck source	20 source types, 15 theorems, and 22 clients: all pass.

The revised source lemma is the real-valued companion for the half squared norm on a convex set C in a real inner-product space. The new theorem `StrongConvexOn.mul_sq_norm` gives m-strong convexity of q_m for $m > 0$:

$$q_m(x) = \frac{m}{2}\|x\|^2, \qquad x \in C. \tag{37}$$

The source takes $m = 1$ and writes the same function as $\|x\|^2/2$. The congruence theorem transports strong convexity along this pointwise equality. The resulting source proof is the following application, with the final arithmetic identity discharged by normalization:

```
exact (StrongConvexOn.mul_sq_norm hC 1).congr
  fun x _ => by ring
```

The earlier proof established convexity after subtracting the quadratic term. The accepted revision places that argument in the general public result and recovers the source by specialization and congruence. The initially concurrent proposals thus acquire a mathematical dependency through the updated library. The accumulated Beck checkpoint passes all 20 declaration-type checks, 15 source-theorem checks, and 22 fixed clients, preserving the source specifications and applications.