OPERATION KOMOREBI

REALIZATION CALCULUS



RESEARCH LIBRARY

Sufficient Resolution

Exact reduction, hidden memory, and certified reconstruction

Lean and evidence guide

Komorebi P.

Standardized publication edition / 8 October 2026

REALIZATION CALCULUS

Reading scope

An exact source map for split representation, behavioral reduction, hidden memory, and certified Hilbert-space reconstruction.

Contents

1 Reading the proof evidence 3
2 The source contract and reproducible identities 3
3 How to read a formal contract 4
4 Block 1: split representation, compression, and behavior 4
5 Block 2: exact discrete memory and finite certificates 6
6 Block 3: finite observable models and minimality 7
7 Block 4: quantitative approximate sufficiency 8
8 Block 5: bounded continuous-time memory 8
9 Block 6: continuous defect transport and stability 9
10 Block 7: orthogonal obstruction and compatibility conditioning 10
11 Block 8: actual minimum-norm reconstruction 11
12 Block 9: the intrinsic obstruction metric 11
13 Block 10: constructed regularized recovery 12
14 Block 11: strong limits and the uniform boundary 13
15 Block 12: data-selected discrepancy 13
16 Block 13: protected inexact certificates 14
17 Block 14: changing available operators 14
18 Block 15: source-bound-free generalized discrepancy 15
19 Block 16: calibration generates whole-operator uncertainty 15
20 Classical support and the proof trust boundary 17
21 Inspecting a statement locally 17
22 A realistic low-memory replay 17
23 Executable companions and their evidence 18
24 Coverage boundaries to retain 19

25 Formula-to-Lean and module appendices **20**

26 Production module inventory **21**

27 Development block census **24**

28 Traditional claim concordance **25**

1 Reading the proof evidence

This guide maps the mathematical contracts to the saved Sufficient Resolution v0.16.0 sources. It explains what the formal objects contain, which hypotheses the theorems require, and how to inspect or replay the proof evidence. The authoritative theorem statement is the elaborated Lean type; the source files and their section variables are the available specification when no compiler is present.

The source inventory contains **93 production modules**, **998 theorem/lemma declarations**, **144 definitions**, **9 abbreviations**, and **8 structures**, giving **1,159 explicitly named declarations**. These counts include supporting lemmas, examples, classical specializations, and definitions. They measure the formal source corpus, not independent mathematical discoveries or publication completion.

The final archived v0.16 checks, dated September 14, 2026, record two separate source-tree runs. Each compiled the **31-module Block 16 dependency closure plus its block root**, inspected the transitive axioms of **52 new named declarations**, and replayed that closure with `leanchecker`. The archived summary explicitly says that a fresh full 93-module corpus replay was **not run**. Earlier blocks have their own archived checks; the latest closure does not replace a full-corpus run.

For this final edition, source inventories, named mappings, preservation of production files, publication builds, and finite Python checks are current review activities. **No fresh Lean compilation was performed.** The portable proof runtime and imported `.lake` cache are absent from the recovered source archive. Recovery of the first portable-runtime chunk and a bounded independent retry both returned HTTP 502, preventing restoration of the five-piece runtime. The repaired full-corpus driver is included for a future replay with the pinned runtime.

The root-level current verification register distinguishes these categories. A plan-only driver result or a textual absence of placeholders is not kernel checking. A passing finite Python test is not a proof of an infinite-dimensional or all-time statement.

2 The source contract and reproducible identities

The project lives in `formal/`. Production statements are in `formal/Komorebi/SufficientResolution/`; `formal/Komorebi.lean` imports all 93 modules. The historical Block 1 guide is `THEOREM_GUIDE.md`; Blocks 2 through 16 are `BLOCK_02_THEOREMS.md` through `BLOCK_16_THEOREMS.md`.

Item	Exact pin or location
Lean toolchain	`leanprover/lean4:v4.34.0-rc2`
Lean commit	`6a10ac8c22beadecabdbb0919c2b50214762f91d`
Mathlib revision	`aeb72f1e9981b6d35b426b16dca7759be3e7ba06`
Toolchain selection	`formal/lean-toolchain`
Dependency configuration	`formal/lakefile.toml`, `formal/lake-manifest.json`
Current formal metadata	`formal/AUDIT.json`, `formal/LEAN_RUNTIME_METADATA.json`
Historical final runs	`formal/verification/release_v0.16.0/report.json`, `formal/verification/roundtrip_v0.16.0/report.json`
Regenerated source inventories	`lean/Declaration_Inventory.csv`, `lean/Module_Inventory.csv`, matching JSON files

The regenerated inventories scan the present files after masking comments and strings. They track namespace and section scopes, record each declaration's source line and source-file SHA-256 digest, and agree exactly with the saved v0.16 declaration index. This is a textual inventory. It neither elaborates a type nor verifies a proof body. The eight source structures do not include automatically generated structure-field declarations in the 1,159 count.

The production proof files remain byte-for-byte unchanged in this edition. Stale top-level metadata and checksums are retained under `formal/inherited_provenance/`. The release copy repairs the old general runner, whose static module list stopped at 81 modules; the new driver discovers the full production import graph and records its scope explicitly. The source audit wrapper `formal/AxiomAudit.lean` was regenerated for all 1,159 unique registry names; its original 1,010-command version is preserved under inherited provenance. This wrapper's source correspondence was checked, but its new audit commands were not executed without the proof runtime.

3 How to read a formal contract

Lean's `LinearMap` type means a linear map over a declared scalar type; `ContinuousLinearMap` means a continuous linear map over the declared normed field. Continuity is part of the latter object and yields an operator norm. A formula involving that norm concerns the complete map, not just selected input vectors.

Lean's section variables are implicit theorem hypotheses. A displayed theorem may depend on `CompleteSpace`, `FiniteDimensional`, or projection instances declared above it. Reading only the theorem's final line can erase these conditions. Conversely, some statements intentionally use less structure than a convenient Hilbert-space specialization.

The source sets `autoImplicit false`. In the Lake configuration, warnings are errors. `noncomputable` marks definitions using classical existence or selection; it does not mean the proof is admitted. A noncomputable parameter selector with a proved specification is distinct from an executable numerical root finder.

Unless a paragraph says otherwise, declaration names below have the common prefix

`Komorebi.SufficientResolution.`

Operator multiplication applies the right factor first. A chronological word $w = [i_1, \dots, i_n]$ applies T_{i_1} first, so $T_w = T_{i_n} \cdots T_{i_1}$. The empty word acts as the identity. This convention controls the recurrence and defect formulas.

4 Block 1: split representation, compression, and behavior

4.1 Exact representation with redundant coordinates

`Resolution R X V` contains linear analysis $a : X \to V$, synthesis $s : V \to X$, and the reconstruction identity $sa = I_X$. The coordinate space may be redundant: as need not equal I_V. Its idempotent $E = as$ selects realizable coordinates.

For resolutions on X and Y, the encoded operator and its decoding are

$$K(T) = a_Y T s_X, \qquad D(K) = s_Y K a_X.$$

The source proves

$$D(K(T)) = T, \qquad K(D(K)) = E_Y K E_X.$$

Thus a proposed coordinate kernel comes from a unique global map exactly when $E_Y K E_X = K$. The actual `Resolution.kernelEquiv` bundles the correspondence with the compatible-kernel subtype. `Resolution.encode_comp` proves composition and `Resolution.encode_id` identifies the encoded identity with E, rather than with the unrestricted coordinate identity.

Principal declarations: `Resolution.projector_idempotent`, `Resolution.fixed_iff_realizable`, `Resolution.decode_encode`, `Resolution.encode_decode`, `Resolution.compatible_iff_existsUnique`, `Resolution.kernelEquiv`, `Resolution.encode_comp`.

Hypotheses: semiring modules with additive commutative monoids; the split maps are supplied. There is no finite-dimensional, metric, orthogonal, or arithmetic-existence assumption.

4.2 Compression records hidden return

`SplitCompression` contains $C : X \to Z$, $R : Z \to X$, and $CR = I_Z$. Over commutative-ring modules, set $P = RC$, $Q = I - P$, and $\Phi(T) = CTR$. Then

$$\Delta(S, T) = \Phi(ST) - \Phi(S)\Phi(T) = CSQTR.$$

This is an exact hidden-return factorization. The source also proves its three-factor coherence identity and two-stage compression formula. Multiplicativity for one pair does not imply multiplicativity for all powers.

Principal declarations: `SplitCompression.defect_factorization`, `SplitCompression.multiplicative_iff_hidden_return_zero`, `SplitCompression.defect_coherence`, `SplitCompression.power_defect`, `SplitCompression.defect_then`.

4.3 Actual globalization obstruction

Over commutative-ring modules, the chain $G \xrightarrow{A} V \xrightarrow{\delta} W$ has compatible space $K = \ker \delta$. The source constructs the actual quotient of K by realizations lying in K. Under the chain condition $\delta A = 0$, this is $K/\operatorname{Ran} A$. For a compatible v, its class vanishes exactly when there exists an actual g with $Ag = v$.

All classes vanish exactly when $\operatorname{Ran} A = \ker \delta$, with the chain condition retained. Source uniqueness additionally requires injectivity of A. The quotient definition by a pulled-back realization submodule and its pointwise zero-class test do not require silently assuming universal realizability.

A `Globalization.PresentationMap` includes both realization and compatibility commuting squares. It induces the actual quotient map, preserves zero obstruction, and reflects zero obstruction only when that induced map is injective. Forward commutation alone does not discharge a global existence or reflection obligation.

Principal declarations: `Globalization.obstructionClass_zero_iff`, `Globalization.all_obstructions_zero_iff_exact`, `Globalization.unique_realization_of_zero`, `Globalization.PresentationMap.obstructionMap`, `Globalization.PresentationMap.reflects_zero_obstruction`.

4.4 The behaviorally sufficient quotient

Over commutative-ring modules with additive groups, for specified operators T_i and linear observation O, define

$$N = \{x : OT_w x = 0 \text{ for every finite word } w\}.$$

`Behavioral.invisible` is an actual submodule. It is invariant under all specified operations and is the largest invariant submodule contained in $\ker O$. The quotient X/N has constructed operations and readout, with every future observation preserved.

For any exact competing model $q : X \to Z$ with $qT_i = U_i q$ and $Vq = O$, the source proves $\ker q \subseteq N$ and constructs a comparison toward the quotient. The operator alphabet need not be finite. These hypotheses quantify over actual maps and all words; a sampled list of trajectories does not satisfy them by itself.

Principal declarations: `Behavioral.invisible_invariant`, `Behavioral.invariant_submodule_le_invisible`, `Behavioral.all_future_observations_preserved`, `Behavioral.quotient_eq_iff_same_future`, `Behavioral.exact_model_kernel_le`, `Behavioral.modelQuotientComparison`.

5 Block 2: exact discrete memory and finite certificates

The general recurrence and prepared-closure identities use commutative-ring modules. The annihilating-polynomial and characteristic-polynomial layer uses a field; finite hidden dimension enters only the dimension-controlled certificate.

`BlockDynamics` constructs the operator

$$T(y, z) = (Ay + Bz, \Gamma y + Dz), \qquad M_k = BD^k\Gamma.$$

Here Γ denotes the source's block `G`, avoiding confusion with the source space used in the inverse-problem chapters. The recurrence comes from powers of the actual full operator. For arbitrary hidden initial state,

$$z_n = D^n z_0 + \sum_{j=0}^{n-1} D^{n-1-j}\Gamma y_j,$$

$$y_{n+1} = Ay_n + BD^n z_0 + \sum_{j=0}^{n-1} M_{n-1-j} y_j.$$

Prepared evolution uses $z_0 = 0$ and $E_n = CT^n R$. Exact agreement $E_n = A^n$ for all n, vanishing of every M_k, the prepared semigroup law, and vanishing of all power defects are equivalent. The first nonzero M_k appears at full depth $k + 2$:

$$E_{k+2} - A^{k+2} = M_k \quad \text{if } M_j = 0 \text{ for } j < k.$$

A supplied monic annihilating polynomial $p(D) = 0$ makes the first $\deg p$ moments an all-depth certificate. Over a field with hidden dimension $d < \infty$, Mathlib's Cayley–Hamilton theorem supplies the characteristic polynomial, yielding

$$[E_n = A^n \text{ for every } n] \iff [M_k = 0 \text{ for } k < d] \iff [E_n = A^n \text{ for } n \le d+1].$$

The visible space may remain infinite-dimensional and $d = 0$ is included. These are equalities of whole linear maps. The canonical split decomposition constructs the actual hidden space $\ker C$ and applies the same certificate to arbitrary split compression. All-state closure is stronger: independence from every initial hidden state holds exactly when $B = 0$.

Principal declarations: `BlockDynamics.discrete_memory_equation`, `BlockDynamics.prepared_powers_iff_memory_zero`, `BlockDynamics.first_return_discrepancy`, `BlockDynamics.memory_zero_of_annihilator`, `BlockDynamics.prepared_powers_iff_finite_horizon`, `BlockDynamics.all_state_closure_iff_B_zero`, `SplitCompression.exact_powers_iff_canonical_finite_horizon`.

6 Block 3: finite observable models and minimality

Over a field with finite-dimensional X, define

$$N_r = \bigcap_{|w| \le r} \ker(OT_w), \qquad k = \dim \ker O.$$

`FiniteHorizon.visibleKernel_succ` establishes

$$N_{r+1} = \ker O \cap \bigcap_i T_i^{-1} N_r.$$

A plateau $N_{r+1} = N_r$ certifies $N_r = N$. The dimension budget proves $N_k = N$ and a distinguishing word of length at most k whenever future behaviors differ. Noncommuting operators are included. Finite enumeration of the bounded word family requires a finite alphabet; the abstract horizon theorem does not.

Minimality concerns exact linear competitors related to the original state by q. The canonical quotient has

$$\dim(X/N) \le \dim \operatorname{Ran} q \le \dim Z,$$

with the last bound under finite-dimensional Z. Its image is minimal exactly when $\ker q = N$. Unused ambient coordinates of Z do not become minimal by this statement.

For finite alphabet, the coordinate map $H_k x = (OT_w x)_{|w| \le k}$ has $\ker H_k = N$. The actual model is $\operatorname{Ran} H_k$, not the full product of arbitrary observation tables. Quotient equivalence, reduced dynamics, and readout are constructed on this realizable range.

For a finite alphabet, a `FiniteCertificate.Certificate` supplies finite-observation generation $q = GH_r$, exact intertwining, and exact output factorization. These identities imply $\ker q = N$ without a finite-dimensional assumption on X. Surjectivity of q is required only to identify the entire target Z with the quotient.

Principal declarations: `FiniteHorizon.visibleKernel_stabilizes`, `FiniteHorizon.same_future_iff_bounded`, `BehavioralMinimality.image_dimension_minimal_iff`, `BehavioralMinimality.minimalImageEquiv`, `FiniteObservableModel.coordinate_all_future_fidelity`, `FiniteCertificate.certificate_kernel_eq`, `FiniteCertificate.certificateEquiv`.

The rational minimizer uses row-space saturation and exact certificate checks. The abstract Lean construction is classical linear algebra, not an extracted Gaussian-elimination algorithm. The Python parser, minimizer, and arbitrary JSON certificate checks are tested implementations.

7 Block 4: quantitative approximate sufficiency

For continuous linear maps on normed spaces over a nontrivially normed field, let

$$\delta_i = qT_i - U_i q, \qquad E_w = qT_w - U_w q.$$

The algebraic transport identity is

$$E_{i::w} = E_w T_i + U_w \delta_i, \qquad E_{u++v} = E_v T_u + U_v E_u.$$

It yields whole-map error budgets. Under nonnegative letter-specific bounds $\|T_i\| \le a_i$, $\|U_i\| \le b_i$, $\|\delta_i\| \le \varepsilon_i$, the source defines product bounds A_w, B_w and a recursively transported allowance D_w, proving $\|E_w\| \le D_w$.

For uniform nonnegative bounds a, b, ε, the response is

$$H_0 = 0, \qquad H_n(a,b) = \sum_{j=0}^{n-1} a^j b^{n-1-j} \quad (n > 0),$$

and $\|E_w\| \le \varepsilon H_{|w|}(a,b)$. The distinct-rate formula is $(a^n - b^n)/(a-b)$; the equal-rate formula is $H_{n+1}(a,a) = (n+1)a^n$. No division by zero is used for equal rates.

If $\|O - Vq\| \le \eta$, both mismatch sources remain:

$$\|OT_w - VU_w q\| \le \eta a^n + \|V\| \varepsilon H_n(a,b).$$

With true operations nonexpansive and reduced operations bounded by $0 \le b < 1$, the all-word bound is $\varepsilon/(1-b)$. Initial reduced-state error retains $b^n \|qx - z\|$. Common operator-norm bounds are assumptions; individual spectral radii do not supply them.

Principal declarations: `ApproximateAlgebra.wordDefect_eq_sum`, `ApproximateBounds.norm_wordError_le_budget`, `ScalarBudgets.response_of_ne`, `ScalarBudgets.response_same_succ`, `StableApproximation.norm_wordError_contractive`, `StableApproximation.norm_output_initial_contractive`, `ApproximateComposition.reencode_outputError`.

No approximate minimum-dimension theorem or automatically selected stable norm is established. The analytic interface does not automatically give every earlier algebraic behavioral quotient a canonical norm or bounded realization. Re-encoding can preserve actual outputs while worsening norm bounds through conditioning. The formal scalar counterexample shows arbitrarily small one-step error can grow without bound.

8 Block 5: bounded continuous-time memory

The analytic flow `ContinuousFlow.flow L t` is the actual operator exponential e^{tL} on a real Banach space. The source proves its derivative, composition law, and uniqueness properties. All real times are

allowed with oriented interval integrals.

For bounded block generator $L(y,z) = (Ay + Bz, \Gamma y + Dz)$, the trajectory $e^{tL}(y_0, z_0)$ gives

$$\begin{aligned} z(t) &= e^{tD}z_0 + \int_0^t e^{(t-s)D}\Gamma y(s)\,ds, \\ y'(t) &= Ay(t) + Be^{tD}z_0 + \int_0^t M(t-s)y(s)\,ds, \\ M(t) &= Be^{tD}\Gamma. \end{aligned}$$

The memory equation is a `HasDerivAt` theorem for the constructed full trajectory. Variation of constants derives the integral identity from an actual ODE and continuous forcing. A converse constructing hidden state from every arbitrary Volterra solution is not included.

The analytic bridge proves that all-time equality of bounded mapped exponentials is equivalent to equality of every mapped generator power. Consequently,

$$[Ce^{tL}R = e^{tA} \text{ for all real } t] \iff [BD^k\Gamma = 0 \text{ for every } k].$$

Prepared semigroup closure is equivalent as well. If hidden dimension is d, the first d feedback moments suffice. Actual iterated derivatives satisfy

$$E^{(n)}(0) = CL^nR, \qquad M^{(n)}(0) = BD^n\Gamma.$$

The first return has derivative discrepancy $BD^k\Gamma$ at order $k+2$. Its Taylor coefficient includes division by $(k+2)!$. Finite sampled times or approximate trajectories are not these operator identities.

Principal declarations: `ContinuousFlow.variation_of_constants`, `ContinuousBlockDynamics.continuous_memory_equation`, `ContinuousFlow.mapped_flow_eq_iff_powers_eq`, `ContinuousBlockDynamics.prepared_exact_iff_finite_memory`, `ContinuousBlockDynamics.prepared_semigroup_iff_memory_zero`, `ContinuousBlockDynamics.prepared_exact_iff_finite_derivatives`, `ContinuousFlow.compressed_flow_iff_finite_generator`.

Unbounded generators, domain issues, time-varying generators, and nonlinear forcing are outside this bounded autonomous contract.

9 Block 6: continuous defect transport and stability

On real Banach spaces, for bounded generators L, A and bounded q, define

$$\delta = qL - Aq, \qquad E(t) = qe^{tL} - e^{tA}q.$$

The source proves the actual backward derivative and Duhamel identity

$$E(t) = \int_0^t e^{(t-s)A}\delta e^{sL}\,ds.$$

Vanishing generator defect is equivalent to vanishing evolution defect at every real time. For nonnegative time, supply whole-flow envelopes valid for every $s \geq 0$, with nonnegative amplitudes M_L, M_A and defect allowance ε:

$$\|e^{sL}\| \le M_L e^{as}, \qquad \|e^{sA}\| \le M_A e^{bs}, \qquad \|\delta\| \le \varepsilon$$

give

$$\|E(t)\| \le M_L M_A \varepsilon R(a,b,t), \qquad R(a,b,t) = \int_0^t e^{b(t-s)} e^{as}\, ds.$$

The exact response is $(e^{at} - e^{bt})/(a-b)$ when $a \neq b$, and te^{at} when $a = b$. If the true flow is bounded and the reduced flow decays at rate $r > 0$, the uniform allowance is $M_L M_A \varepsilon / r$. Readout mismatch and reduced initialization have their own terms, including the original state's norm.

Principal declarations: `ContinuousDefect.duhamel`, `ContinuousDefect.generatorDefect_zero_iff`, `ExponentialBudgets.response_of_ne`, `ContinuousApproximation.norm_evolution_le_response`, `ContinuousStability.norm_evolution_uniform`, `ContinuousStability.norm_initialization_uniform`, `ContinuousApproxExamples.unstable_error_unbounded`.

The growth envelopes are external whole-map premises. A small generator discrepancy alone supplies no uniform future-error control; the scalar boundary example proves this failure for actual exponentials.

10 Block 7: orthogonal obstruction and compatibility conditioning

Let $R \subseteq K \subseteq V$ be real inner-product subspaces admitting orthogonal projections. The actual components are

$$r = P_R v, \qquad o = (P_K - P_R)v, \qquad i = (I - P_K)v.$$

Their unique, pairwise orthogonal decomposition satisfies

$$v = r + o + i, \qquad \operatorname{dist}(v,R)^2 = \|o\|^2 + \|i\|^2.$$

For a presentation $G \xrightarrow{A} V \xrightarrow{\delta} W$ with $\delta A = 0$, use $R = \operatorname{Ran} A$ and $K = \ker \delta$. The original algebraic obstruction quotient is linearly identified with $K \cap R^{\perp}$. This block establishes a linear equivalence; its isometry is supplied later in Block 9.

If $\sigma > 0$ and $\sigma\|x\| \le \|\delta x\|$ for every $x \in K^{\perp}$, then

$$\operatorname{dist}(v,R)^2 \le \|o\|^2 + \left(\frac{\|\delta v\|}{\sigma}\right)^2.$$

Compatibility residual, obstruction, and conditioning remain distinct. Finite-dimensional V supplies some positive coercivity constant, not its optimal value or a uniform constant over presentations. Closed range remains explicit in Hilbert-space existence statements.

Principal declarations: `OrthogonalResolution.norm_sq_components`, `OrthogonalResolution.best_approximation_unique`, `HilbertGlobalization.obstructionEquivHarmonic`, `HilbertConditioning.conditioned_distance_sq`, `HilbertConditioning.exists_coercivity_finite`, `FiniteGlobalization.finite_residual_bound_iff_exact`, `GlobalizationRefinement.distance_refinement_sq`.

For $R_0 \subseteq R_1 \subseteq K$ with fixed metric and compatibility space, the exact gain $(P_{R_1} - P_{R_0})v$ accounts for decreased obstruction and distance. It is not a theorem about arbitrary changes of packetization or metric.

11 Block 8: actual minimum-norm reconstruction

For bounded real $A : G \to V$, the core assumes complete source and complete actual range, with the required orthogonal projections. The public Hilbert constructor takes an explicit proof that Ran A is closed. It constructs the inverse of the restriction

$$A : (\ker A)^{\perp} \longrightarrow \operatorname{Ran} A$$

using the open mapping theorem, then composes it with $P_{\operatorname{Ran} A}$. The resulting $B = A^{\dagger}$ is constructed; it is not supplied as an inverse or defined by assuming the desired optimum.

$$AB = P_{\operatorname{Ran} A}, \qquad BA = P_{(\ker A)^{\perp}}.$$

The source proves the Penrose identities, actual least-squares optimality, characterization of every minimizer as $Bv + k$ with $k \in \ker A$, and uniqueness of the minimum-norm minimizer. The symmetry conditions are stated in real inner-product form, rather than as separate adjoint equalities.

For arbitrary noise e and source g,

$$\|B(Ag + e) - g\|^2 = \|Be\|^2 + \|P_{\ker A} g\|^2.$$

Whole-source conditioning $\alpha > 0$ gives $\|B\| \leq 1/\alpha$. This α belongs to A and is independent of compatibility conditioning σ for δ. A computed candidate needs control of both image-fit error and source-kernel leakage to certify proximity to the minimum-norm source.

Principal declarations: `MinimumNormReconstruction.recover`, `MinimumNormReconstruction.recover_unique`, `MinimumNormOptimality.unique_minimum_norm_least_squares`, `MinimumNormOptimality.noisy_source_error_sq`, `SourceConditioning.candidate_source_bound_sq`, `ClosedRangeReconstruction.closed_range_iff_coercivity`, `ClosedRangeReconstruction.nonclosed_no_bounded_recovery`.

Closed range is equivalent to off-kernel coercivity and to existence of bounded recovery satisfying the source-projection identity. Density does not supply actual bounded reconstruction.

12 Block 9: the intrinsic obstruction metric

The source uses Mathlib's native quotient seminorm on the same Block 1 obstruction quotient. Under the projection hypotheses, it upgrades the previous equivalence to a genuine linear isometry

$$K/R \simeq_{\text{lin,isom}} K \cap R^{\perp}.$$

For compatible v, $\|[v]\| = \operatorname{dist}(v, R)$. The harmonic section is the unique minimum-norm compatible representative of the class. This local representative is different from Block 8's minimum-norm global source.

For arbitrary data, the bounded intrinsic class is $c(v) = [P_K v]$, with norm at most one. Then

$$\operatorname{dist}(v, R)^2 = \|c(v)\|^2 + \|(I - P_K)v\|^2.$$

Forward bounded presentation maps induce bounded maps on the existing quotients. A reverse quantitative estimate requires a reverse presentation map and a proved left-inverse identity on quotients. Forward boundedness alone does not reflect zero obstruction.

For arbitrary submodule R, including nonclosed ones,

$$\|[v]\| = 0 \iff v \in \overline{R}, \qquad [v] = 0 \iff v \in R.$$

Thus a nonclosed quotient can contain a nonzero zero-seminorm class. The source proves that such a quotient has no linear isometry equivalence to a genuine normed space. It does not prohibit noninjective norm-preserving maps out of a seminormed space.

Principal declarations: `MetricObstruction.obstructionIsometry`, `MetricRepresentatives.unique_minimum_representative`, `IntrinsicGlobalization.intrinsic_distance_sq`, `MetricObstructionTransport.two_sided_bound`, `MetricRefinement.quotient_refinement_energy`, `QuotientMetricBoundaries.nonclosed_nonzero_zero_norm`, `QuotientMetricBoundaries.nonclosed_no_normed_isometry`.

13 Block 10: constructed regularized recovery

On complete real inner-product spaces, with actual bounded A and $\lambda > 0$, define

$$N_\lambda = A^*A + \lambda I, \qquad R_\lambda = N_\lambda^{-1}, \qquad B_\lambda = R_\lambda A^*.$$

Mathlib's Lax–Milgram theorem supplies the inverse after coercivity of the actual normal form is proved. Closed range and finite dimension are unnecessary for this construction. Both inverse laws and the actual normal equation are established.

The literal objective has the exact increment

$$F_\lambda(v, x) = \|Ax - v\|^2 + \lambda\|x\|^2,$$
$$F_\lambda(v, x) - F_\lambda(v, B_\lambda v) = \|A(x - B_\lambda v)\|^2 + \lambda\|x - B_\lambda v\|^2.$$

This proves unique global optimality. The fixed-parameter bounds include

$$\|R_\lambda\| \le \lambda^{-1}, \qquad \|B_\lambda\| \le \frac{1}{2\sqrt{\lambda}}, \qquad \|AB_\lambda v\| \le \|v\|.$$

An actual normal residual $r = N_\lambda x - A^*v$ gives $x - B_\lambda v = R_\lambda r$ and source allowance $\|r\|/\lambda$. Bias and propagated noise are generally not orthogonal; their sum is orthogonal only to the source-kernel component in the retained squared-error identity.

Changing operators and parameters have explicit resolvent sensitivity bounds. Those estimates need not preserve rank but are not uniform as $\lambda \to 0$. Integration with harmonic obstruction and Moore–Penrose recovery reinstates actual complete/closed range precisely where those predecessor objects are used.

Principal declarations: `TikhonovConstruction.normalEquiv`, `TikhonovConstruction.normal_resolvent`, `TikhonovOptimality.unique_minimizer`, `TikhonovBounds.regularized_opNorm_bound`, `TikhonovBounds.approximate_solver_bound`, `TikhonovBias.noisy_source_error_squared`, `TikhonovPerturbation.resolvent_difference`, `TikhonovGlobalization.regularized_intrinsic_error`.

14 Block 11: strong limits and the uniform boundary

For fixed bounded A on real Hilbert spaces and a positive parameter family tending to zero along a filter, the source proves

$$B_\lambda A g \longrightarrow P_{(\ker A)^\perp} g.$$

The proof establishes density of $\mathrm{Ran}(A^*A)$ in $(\ker A)^\perp$ and an explicit approximation bound. It does not assume every identifiable source is exactly A^*Aw with a controlled w.

Noise satisfies strong source consistency when $\|e_i\|/\sqrt{\lambda_i} \to 0$. An inexact normal solve adds the actual residual condition $\|r_i\|/\lambda_i \to 0$. These are sufficient conditions, not universal necessary ones.

Strong convergence and operator-norm convergence differ. For nonclosed actual range, the source-error operator has norm one at **every** positive parameter:

$$\|B_\lambda A - P_{(\ker A)^\perp}\| = 1.$$

For a nontrivial parameter filter, uniform convergence to zero is equivalent to closed range. Fitted data always converge to $P_{\overline{\mathrm{Ran}\, A}} v$. Data in closure but outside actual range may be fitted asymptotically while the reconstructed source family has no norm limit; norm divergence is not asserted.

Principal declarations: `TikhonovDensity.gram_range_closure`, `TikhonovConvergence.tendsto_source`, `NoisyRegularization.noisy_inexact_solver_tendsto`, `UniformRegularization.nonclosed_sourceError_norm`, `UniformRegularization.uniform_convergence_iff_closed_range`, `RegularizedDataLimit.unrealizable_no_source_limit`.

15 Block 12: data-selected discrepancy

For actual B_λ, let $r_v(\lambda) = \|AB_\lambda v - v\|$ on positive parameters. Continuity, monotonicity, and strict monotonicity when $A^*v \neq 0$ are proved. If $A^*v = 0$, the residual is constant.

The parameter-selection contract is

$$\mathrm{dist}(v, \mathrm{Ran}\, A) < \rho < \|v\| \Longrightarrow \exists! \lambda > 0 :\ r_v(\lambda) = \rho.$$

Distance means actual `Metric.infDist`; no closed range or nearest point is assumed. The noncomputable selector uses the intermediate value theorem and strict monotonicity. The Python search returns a rational point in a certified band, not necessarily the exact real root.

The `Accepted` predicate covers small-data zero and actual positive-parameter Tikhonov outputs satisfying $\eta \le r_v(\lambda) \le \tau\eta$. For valid data allowance $\|Ag - v\| \le \eta$ with $\eta \to 0$ and fixed $\tau > 1$, accepted selections converge to the identifiable projection without a separately imposed parameter schedule. Exact Tikhonov output remains part of this predicate; inexact solves enter the next block.

Principal declarations: `DiscrepancySelection.exists_unique_parameter_of_distance`, `DiscrepancySelection.parameter_hits_target`, `DiscrepancyConvergence.norm_control_residual_tendsto`, `DiscrepancyRule.selectedSource_accepted`, `DiscrepancyRule.selectedSource_consistent`, `DiscrepancyBoundaries.zero_residual_iff_zero_data`.

16 Block 13: protected inexact certificates

For computed x, actual normal residual $r = (A^*A + \lambda I)x - A^*v$, and $\lambda > 0$, the source separates source and fitting scales:

$$\|x - B_\lambda v\| \le \frac{\|r\|}{\lambda}, \qquad \|A(x - B_\lambda v)\| \le \frac{\|r\|}{2\sqrt{\lambda}}.$$

If $\|r\| \le \lambda\kappa$, $\|r\| \le 2\sqrt{\lambda}\xi$, and the measurement satisfies $|m - \|Ax - v\|| \le \omega$, the protected band

$$\eta + \omega + \xi \le m \le \tau\eta - \omega - \xi$$

certifies the exact companion's accepted residual and source correction at most κ. Its scalar interval is nonempty exactly when $2(\omega + \xi) \le (\tau - 1)\eta$; this does not prove a computed candidate satisfying every field exists.

`PositiveCertificate` records nonnegative scalar allowances and squared norm inequalities, including the squared enclosure equivalent to the measurement bound. `certificate_sound` produces a genuine near-accepted witness. For fixed true operator A and fixed discrepancy factor τ, valid noise tending to zero and source-correction budget $\kappa \to 0$ imply consistency, including mixed small-data and positive branches. No arbitrary solver is declared near-accepted without its certificate.

Principal declarations: `InexactResidual.correction_bound`, `InexactResidual.image_resolvent_bound`, `InexactDiscrepancy.measured_certificate`, `InexactConsistency.near_consistent`, `InexactToleranceCertificates.certificate_sound`, `InexactToleranceCertificates.uncertain_normal_certificate`, `InexactExamples.small_raw_residuals_do_not_force_convergence`.

The formal boundary shows raw normal residual and fitting residual can both vanish while the candidate stays a unit from the selected source. The required scaled residual need not vanish in that example.

17 Block 14: changing available operators

Let A be fixed true bounded operator, C_i available operators on the same Hilbert spaces, and $p = P_{(\ker A)^\perp} g$. The true premises are

$$\|C_i - A\| \le \delta_i, \qquad \|Ag - v_i\| \le \eta_i, \qquad \|p\| \le M.$$

Effective model noise is $\eta_i + \delta_i M$. Actual accepted companions under C_i have source norm dominated by the fixed p and true residual controlled by $(\tau + 1)(\eta_i + \delta_i M)$. Near-accepted sources add explicit κ_i terms.

For fixed discrepancy factor τ, the formal limit is $x_i \to p$ when $\eta_i, \delta_i, \kappa_i \to 0$ and the acceptance conditions hold. It does not require equal ranks, matching kernels, converging model-kernel projections, or a prescribed δ_i/λ_i schedule. The source ambiguity belongs to the fixed true A.

Operational certificates contain C, data, and declared allowances; their soundness under unknown A remains conditional on the true operator and data bounds. The exact normal-residual transport identity distinguishes solver error from model error.

Principal declarations: `OperatorUncertainty.effective_data_noise`, `OperatorResidualTransport.normal_residual_difference`, `ChangingDiscrepancy.norm_control`, `ChangingConsistency.projected_consistent`, `OperatorCertificates.certified_consistent`, `OperatorUncertaintyExamples.biased_model_wrong_limit`.

18 Block 15: source-bound-free generalized discrepancy

The new candidate-dependent allowance is

$$\text{candidateNoise}(\eta, \delta, x) = \eta + \delta\|x\|.$$

The acceptance predicate contains no supplied source cap M. An accepted positive-parameter source is an actual Tikhonov solution satisfying

$$\eta + \delta\|x\| \le \|Cx - v\| \le \tau(\eta + \delta\|x\|).$$

Literal objective comparison proves self-bounding: for a feasible comparison p, the lower band implies $\|x\| \le \|p\|$. Thus accepted sources obtain the source-norm control previously supplied by M. With valid true data/operator allowances tending to zero, they converge to the fixed identifiable source.

Acceptance and feasibility are separate. One sufficient condition for selection is

$$\delta\|p\| < (\tau - 1)\eta, \qquad \eta > 0, \quad \tau > 1.$$

The comparison source appears in the existence theorem's hypotheses; it is not an input to the generalized acceptance predicate. The raw `GeneralizedDiscrepancySelection.selectedSource` constructor itself takes the true operator, a comparison source, and truth/feasibility proofs, and uses classical choice. It is a theorem-level existence witness with a proved specification, rather than an executable available-data API. The formally checked zero-model example has valid true allowances but no accepted generalized source, refuting unconditional feasibility. Near-accepted computed outputs retain their explicit source-correction budget.

Principal declarations: `GeneralizedDiscrepancy.regularized_norm_domination`, `GeneralizedDiscrepancy.accepted_consistent`, `GeneralizedDiscrepancySelection.exists_accepted`, `GeneralizedDiscrepancyConsistency.selected_from_true_source_consistent`, `GeneralizedInexact.near_consistent`, `GeneralizedFeasibility.exists_accepted_of_scaled_error`, `GeneralizedDiscrepancyExamples.zero_model_not_accepted`.

19 Block 16: calibration generates whole-operator uncertainty

19.1 Exact and redundant frames

`CalibrationFrame.Frame K G` contains bounded $U : K \to G$, $R : G \to K$, and $UR = I_G$. The calibration synthesis need not be injective, square, or nonredundant. For measured response operator $Y : K \to V$, true operator A, and model C, assume

$$\|Y - AU\| \le \eta_{\mathrm{cal}}, \qquad \|Y - CU\| \le f.$$

The exact factorization and allowance are

$$A - C = (AU - CU)R, \qquad \|C - A\| \le (\eta_{\mathrm{cal}} + f)\|R\|.$$

This converts justified calibration-truth error and measured model fit into a whole-source operator bound. It does not infer η_{cal} from observations. Exact calibration identifies the operator because the frame spans the source through its bounded right inverse.

19.2 Recovery, hierarchy, and approximate frames

The generated allowance enters the existing generalized discrepancy and feasibility-aware selector. For a fixed true bounded operator A between real Hilbert spaces, a fixed bounded split frame, and fixed $\tau \ge 0$, accepted outputs converge to the fixed identifiable source when valid data, calibration-truth, and fit allowances vanish. Actual selected-source existence additionally uses $\tau > 1$ and the relative-accuracy margin involving that comparison source. Changing reconstruction maps with unbounded norms are not covered merely by vanishing calibration errors.

For an arbitrary true source g, apply the identifiable theorem with $p = P_{(\ker A)^\perp} g$ and genuine data bounds $\|Ag - v_i\| \le \eta_i$. Then the source-error norm tends to $\|P_{\ker A} g\|$. The noise allowance concerns the original source's data; the projection has the same forward image.

Split frames compose; their reconstruction norms multiply in the bound. For an approximate frame with

$$\|I - UR\| \le \rho < 1,$$

the source proves

$$\|C - A\| \le \frac{(\eta_{\mathrm{cal}} + f)\|R\|}{1 - \rho}.$$

No useful continuation at $\rho = 1$ is asserted. The decomposition $T = (TU)R + T(I - UR)$ supplies the estimate, without assuming the desired conclusion.

19.3 The unobserved-direction boundary

With partial calibration $U(t) = (t, 0)$ on $\mathbb{R}^2$, the operators $C_s(x, y) = sy$ agree perfectly on every calibrated source while their operator norms can be arbitrarily large. Lean proves both the nonspanning premise and the universal unbounded-error conclusion. Finite perfect calibration does not control an unobserved direction.

Principal declarations: `CalibrationFrame.Frame.measured_operator_bound`, `CalibrationFrame.Frame.exact_calibration_identifies`, `CalibrationRecovery.generated_operator_bound`, `CalibrationRecovery.calibrated_consistent`, `CalibrationSelection.exists_accepted_from_calibration`, `CalibrationHierarchy.hierarchical_operator_bound`, `ApproximateCalibration.measured_approximate_operator_bound`, `CalibrationExamples.perfect_partial_calibration_unbounded`.

The rational calibration implementation uses a conservative entrywise-ℓ^1 matrix bound. Those finite certificates can upper-bound Euclidean operator norms, but the checker itself is tested Python. Its norm computation is not silently identified with Lean's native operator norm.

20 Classical support and the proof trust boundary

The source roots import established Mathlib algebra and analysis. The integration uses classical mathematics, including quotient modules, finite-dimensional rank arguments, Cayley–Hamilton, orthogonal projections, the Banach open mapping theorem, operator exponentials, interval integration, adjoints, Lax–Milgram, and the intermediate value theorem. Moore–Penrose and Tikhonov constructions are classical objects. Their inclusion and formal integration do not assert historical priority for those results.

The project-specific statements identify actual constructions with the earlier representation, obstruction, behavioral, and certificate objects. A theorem that imports Mathlib can still prove the specific integration identity; its imported dependencies remain part of the trust boundary.

The archived transitive axiom allowlist is exactly `propext`, `Classical.choice`, and `Quot.sound`. The audit includes the allowed standard foundations rather than claiming an axiom-free project. Deliberate false statements and placeholders were rejected in recorded controls. A custom axiom intentionally compiled in a negative control and was then rejected by the allowlist. Its presence in archived control files is not a production axiom.

`leanchecker` uses the same Lean kernel as compilation. Its replay is an additional check of saved project proof objects, not an independent verifier implementation. Imported pinned Lean/Mathlib artifacts were reused, not rebuilt from first principles.

21 Inspecting a statement locally

With the pinned runtime and dependency cache available, a small inspection file can expose a theorem's complete type and transitive axioms. For example:

```
import Komorebi.SufficientResolution.CalibrationFrame
import Komorebi.SufficientResolution.FiniteCertificate

#check Komorebi.SufficientResolution.FiniteCertificate.certificate_kernel_eq
#check Komorebi.SufficientResolution.CalibrationFrame.Frame.measured_operator_bound
#print axioms Komorebi.SufficientResolution.FiniteCertificate.certificate_kernel_eq
#print axioms Komorebi.SufficientResolution.CalibrationFrame.Frame.measured_operator_bound
```

Compile it in the project environment and inspect the actual printed hypotheses. A declaration inventory row is a navigation aid. It does not replace this inspection, and a successful `#check` in isolation presupposes the imported proof object was obtained under the stated runtime.

22 A realistic low-memory replay

22.1 Required environment

The portable runtime is an external prerequisite. The source ZIP does not contain the five runtime chunks, the restored toolchain, or the `.lake` dependency cache. A Linux x86_64 environment, Bash, Python, and the exact pinned Lean/Mathlib dependency objects are required by the documented portable route.

A compatible WSL Linux environment can be prepared separately; native Windows execution is not certified by the archived runs.

The archived runtime metadata supplies a combined compact-archive SHA-256 identity, but possession of that metadata is not possession of the archive. Obtain the complete matching runtime, check its documented hash, restore it using its own instructions, and identify its `run.sh`. Do not silently substitute a newer Lean release or Mathlib HEAD for the saved proof environment.

22.2 Plan before compilation

From the final edition root:

```
python3 formal/scripts/verify.py --plan-only
```

This discovers all 93 production modules and the 1,159 named declarations, checks source structure, and prints the intended scope without proving the declarations. With an independently restored runtime, use the release wrapper:

```
bash formal/verify_low_memory.sh \
  /absolute/path/Komorebi_Portable_Lean_4.34.0-rc2 \
  --out verification/user-full-replay
```

The wrapper and driver record actual paths, pins, executed inputs, command outcomes, source hashes, axiom results, and replay scope. `formal/PORTABLE_REPLAY.md` gives the current portable procedure. Keep user-generated reports separate from archived verification records. Review the generated report's completion status and every failure; an interrupted or timed-out run is not a complete pass.

22.3 Resource discipline

Use one Lean thread and serial commands. The archived environment had a 4 GiB memory limit and a four-core quota, with serial Lean invocation. A proof that compiles in that environment is not thereby guaranteed to fit every smaller machine. Provide adequate memory and per-command timeouts; changing a timeout does not change a theorem.

The default replay should include compilation, full named transitive-axiom inspection, and same-kernel replay. A deliberate `-skip-kernel-replay` run has reduced verification scope and must be reported that way. Existing compiled project objects may be needed by later modules; the driver builds the local import graph in dependency order. Pinned imported Mathlib artifacts remain reused dependencies.

Never treat the old archived general runner's 81-module success as a full 93-module result. The repaired driver includes Blocks 15 and 16 and the updated finite companions. Its publication-time plan was checked; its fresh proof compilation remains pending because the runtime could not be restored.

23 Executable companions and their evidence

The source archive contains 34 Python scripts and 25 example-data files. They implement finite rational observable-model construction, approximate error budgets, orthogonal decomposition, reconstruction, metric quotient certificates, Tikhonov recovery, discrepancy searches, inexact certificate checks, operator uncertainty, generalized discrepancy, and calibration.

Most mathematical controls use exact `fractions.Fraction` arithmetic. Selected comparison suites also use SymPy. Matrix norms differ by suite: induced infinity norm appears in approximate dynam-

ics, Euclidean geometry in reconstruction, and conservative entrywise-ℓ^1 bounds in calibration. The computation record identifies which norm and range each suite tested.

Useful direct calls inside `formal/` include:

```
python3 scripts/check_observable_model.py
python3 scripts/check_reconstruction.py
python3 scripts/check_regularization_limits.py
python3 scripts/check_inexact_discrepancy.py
python3 scripts/check_operator_uncertainty.py
python3 scripts/check_generalized_discrepancy.py
python3 scripts/check_calibration.py
```

Finite tests check formulas, alternative calculation routes, exact certificates, parser rejection, and selected examples. They do not prove universal strong limits, all-word minimality, exact Hilbert-space existence, calibration noise bounds, or executable-search correctness for every allowed input. The algorithms and parsers are not extracted from Lean or formally verified.

The final-edition replay passed all 16 finite suites, reporting 40,466 native counted assertions, five sanity groups (six main assertions), and 1,073 rejected controls. The 17 current v2 computation records comprise those 16 runs and the repaired-runner/source checks; all passed root and live-hash validation. These totals use the suites' native reporting conventions and are not a count of distinct theorems. The current records live in `verification/python_replay/`, `verification/runner_plan_check/`, and `verification/MANIFEST_VALIDATION.json`.

`verification/CURRENT_RELEASE_VERIFICATION.json` gives the current scope and totals; `verification/RUNTIME_RECOVERY.json` records both failed runtime transfers. The finite replay driver is `verification/replay_finite_companions.py`. The current manifests record live input hashes, output hashes, and validation status. Archived numerical totals in block guides remain historical. Repeating a seeded suite does not create additional distinct tested systems, and summing assertion counts does not measure proof coverage.

24 Coverage boundaries to retain

The formal project establishes the stated linear contracts and numerous actual sharpness or failure examples. It does not establish an unrestricted sufficient realization for every mathematical object or task.

The remaining connections include deriving a calibration-truth allowance from individual noisy samples or a justified stochastic instrument model; linking exact behavioral minimality to stable target-only recovery under task-specific norms; formal verification or extraction of Python algorithms and parameter searches; and extensions to constrained, positive, stochastic, integral, nonlinear, geometric, and varying-metric settings.

The bounded continuous-time layer does not establish unbounded-generator or domain results. The classical strong-limit layer does not supply a common rate over all identifiable sources. Nonclosed actual range does not become closed by approximate fit. Forward obstruction transport does not automatically reflect realizability. No arithmetic existence theorem or Langlands correspondence follows from the generic linear presentation contracts.

Each completed theorem remains valuable within its explicit hypotheses. The paper's claim register groups those results for readers; the declaration inventory supplies the implementation-level detail.

25 Formula-to-Lean and module appendices

The accompanying `Claim_Lean_Concordance.csv` and JSON file map the traditional paper's stable claim IDs to representative declarations and source locations. A group can require several declarations; one representative is a starting point for checking its hypotheses and dependency chain. Definitions that construct a mathematical object are distinguished from theorems about it.

`Declaration_Inventory.csv` and JSON contain all 1,159 named declarations, including helpers and examples. `Module_Inventory.csv` and JSON contain all 93 production modules, block membership, declaration counts, local and external imports, source lines, sizes, and full SHA-256 hashes. `INVENTORY_AUDIT.json` records exact agreement with the inherited v0.16 index and hashes of all sixteen guide inputs.

The following appended module table provides a compact navigation view. Names are relative to `Komorebi.SufficientResolution`; all source files are in `formal/Komorebi/SufficientResolution/`. Full inventory fields remain available in the machine-readable files.

26 Production module inventory

All counts below are textual source counts. The source hashes, imports, and declaration locations are supplied in the full CSV and JSON inventories.

Block	Module	Thm./lemma	Named
01	`Behavioral`	14	21
01	`Compression`	19	25
01	`ExactResolution`	15	21
01	`Examples`	8	12
01	`Globalization`	7	10
01	`ObstructionTransport`	5	9
02	`DiscreteMemory`	15	22
02	`FiniteMemory`	6	6
02	`MemoryClosure`	14	14
02	`MemoryExamples`	12	14
02	`SplitMemory`	10	15
03	`BehavioralMinimality`	13	15
03	`FiniteCertificate`	10	12
03	`FiniteHorizon`	19	20
03	`FiniteObservableModel`	15	24
03	`ObservableExamples`	8	23
04	`ApproximateAlgebra`	12	15
04	`ApproximateBounds`	18	24
04	`ApproximateComposition`	11	12
04	`ApproximateExamples`	12	13
04	`ScalarBudgets`	17	18
04	`StableApproximation`	12	12
05	`ContinuousClosure`	19	21
05	`ContinuousExamples`	16	18
05	`ContinuousFlow`	14	16
05	`ContinuousJets`	9	9
05	`ContinuousMemory`	18	25
05	`ContinuousSplit`	4	5
05	`ExponentialTransfer`	11	12
05	`VariationOfConstants`	6	6
06	`ContinuousApproxExamples`	11	11
06	`ContinuousApproximation`	7	7
06	`ContinuousDefect`	13	17
06	`ContinuousStability`	10	10
06	`ExponentialBudgets`	16	18
07	`FiniteGlobalization`	6	6
07	`GlobalizationExamples`	11	12
07	`GlobalizationRefinement`	9	9
07	`HilbertConditioning`	11	12
07	`HilbertGlobalization`	12	16
07	`OrthogonalResolution`	27	31
08	`ClosedRangeReconstruction`	14	15
08	`MinimumNormOptimality`	13	13
08	`MinimumNormReconstruction`	21	24

Continued on next page

Block	Module	Thm./lemma	Named
08	ReconstructionExamples	13	14
08	ReconstructionGlobalization	8	8
08	SourceConditioning	13	13
09	IntrinsicGlobalization	14	15
09	MetricObstruction	15	16
09	MetricObstructionTransport	7	8
09	MetricRefinement	7	9
09	MetricRepresentatives	9	11
09	QuotientMetricBoundaries	9	9
10	TikhonovBias	13	13
10	TikhonovBounds	14	14
10	TikhonovConstruction	14	19
10	TikhonovExamples	11	12
10	TikhonovGlobalization	10	10
10	TikhonovOptimality	11	12
10	TikhonovPerturbation	10	10
11	NoisyRegularization	7	7
11	RegularizedDataLimit	8	8
11	TikhonovConvergence	6	6
11	TikhonovDensity	8	9
11	UniformRegularization	7	8
12	DiscrepancyBoundaries	7	7
12	DiscrepancyConvergence	7	7
12	DiscrepancyResidual	10	12
12	DiscrepancyRule	10	12
12	DiscrepancySelection	12	13
13	InexactConsistency	7	7
13	InexactDiscrepancy	13	14
13	InexactExamples	10	11
13	InexactResidual	13	14
13	InexactToleranceCertificates	11	12
14	ChangingConsistency	5	5
14	ChangingDiscrepancy	8	8
14	OperatorCertificates	9	10
14	OperatorResidualTransport	4	4
14	OperatorUncertainty	12	13
14	OperatorUncertaintyExamples	11	12
15	GeneralizedDiscrepancy	8	10
15	GeneralizedDiscrepancyConsistency	5	5
15	GeneralizedDiscrepancyExamples	11	11
15	GeneralizedDiscrepancySelection	6	7
15	GeneralizedFeasibility	5	5
15	GeneralizedInexact	6	7
16	ApproximateCalibration	4	4
16	CalibrationExamples	10	13
16	CalibrationFrame	12	14
16	CalibrationHierarchy	5	6
16	CalibrationRecovery	7	8
16	CalibrationSelection	6	7

Continued on next page

Block	Module	Thm./lemma	Named
Total	93 modules	998	1,159

27 Development block census

These are additions by historical development block, not counts of independent claims.

Block	Modules added	Thm./lemma added	Named added
01	6	68	98
02	5	57	71
03	5	65	94
04	6	82	94
05	8	97	112
06	5	57	63
07	6	76	86
08	6	82	87
09	6	61	68
10	7	83	90
11	5	36	38
12	5	46	51
13	5	54	58
14	6	49	52
15	6	41	45
16	6	44	52

28 Traditional claim concordance

Each stable paper claim maps to the declarations named below. These are representative proof anchors; full signatures, dependencies, and hypotheses remain in the source. The formula and hypothesis synopsis is retained in the machine-readable concordance.

ID	Claim group	Representative declarations

SR-01	Exact redundant representation	`Resolution.projector_idempotent`
		`Resolution.fixed_iff_realizable`
		`Resolution.compatible_iff_existsUnique`
		`Resolution.encode_comp`
		`Resolution.encode_id`

SR-02	Hidden-return defect and coherence	`SplitCompression.defect_factorization`
		`SplitCompression.defect_coherence`
		`SplitCompression.defect_then`

SR-03	Actual globalization obstruction	`Globalization.obstructionClass_zero_iff`
		`Globalization.all_obstructions_zero_iff_exact`
		`Globalization.unique_realization_of_zero`
		`Globalization.PresentationMap.obstructionMap_compose`
		`Globalization.PresentationMap.reflects_zero_obstruction`

SR-04	The all-future invisible quotient	`Behavioral.invisible_invariant`
		`Behavioral.invariant_submodule_le_invisible`
		`Behavioral.all_future_observations_preserved`
		`Behavioral.quotient_eq_iff_same_future`
		`Behavioral.exact_model_kernel_le`

SR-05	Exact hidden elimination	`BlockDynamics.hidden_elimination`
		`BlockDynamics.discrete_memory_equation`
		`BlockDynamics.prepared_recurrence`

SR-06	Prepared closure and the first return	`BlockDynamics.prepared_powers_iff_memory_zero`
		`BlockDynamics.prepared_semigroup_iff_memory_zero`
		`BlockDynamics.first_return_discrepancy`
		`BlockDynamics.prepared_prefix_iff_memory_prefix`
		`BlockDynamics.all_state_closure_iff_B_zero`

SR-07	Finite all-depth closure certificate	`BlockDynamics.memory_zero_of_annihilator`
		`BlockDynamics.prepared_powers_iff_finrank_memory`
		`BlockDynamics.prepared_powers_iff_finite_horizon`
		`BlockDynamics.nonclosure_has_finite_feedback_witness`

SR-08	Canonical hidden state of a split compression	`SplitCompression.block_synthesis_analysis`
		`SplitCompression.block_analysis_synthesis`
		`SplitCompression.block_powers_eq`
		`SplitCompression.exact_powers_iff_canonical_memory_zero`
		`SplitCompression.exact_powers_iff_canonical_finite_horizon`

SR-09	Exact horizon and plateau certificate	`FiniteHorizon.visibleKernel_succ`
		`FiniteHorizon.visibleKernel_eq_invisible_of_plateau`
		`FiniteHorizon.visibleKernel_stabilizes`
		`FiniteHorizon.same_future_iff_bounded`

Continued on next page

ID	Claim group	Representative declarations

SR-10	Minimum exact state dimension	`BehavioralMinimality.quotient_is_observable`
		`BehavioralMinimality.imageComparison`
		`BehavioralMinimality.finrank_state_le_image`
		`BehavioralMinimality.image_dimension_minimal_iff`
		`BehavioralMinimality.minimalImageEquiv`

SR-11	Realizable finite observation tables	`FiniteObservableModel.mem_wordsUpTo`
		`FiniteObservableModel.ker_complete_observation`
		`FiniteObservableModel.observableEquiv`
		`FiniteObservableModel.coordinate_all_future_fidelity`
		`FiniteObservableModel.coordinate_dimension_le_exact_model`

SR-12	Finite minimality certificate	`FiniteCertificate.certificate_kernel_eq`
		`FiniteCertificate.certificate_same_future_iff`
		`FiniteCertificate.certificateEquiv`
		`FiniteCertificate.certificateEquiv_dynamics`
		`FiniteCertificate.certificateEquiv_observation`

SR-13	Exact word-defect transport	`ApproximateAlgebra.wordDefect_cons`
		`ApproximateAlgebra.wordDefect_append`
		`ApproximateAlgebra.wordDefect_eq_sum`
		`ApproximateAlgebra.output_defect_decomposition`

SR-14	Heterogeneous operator budgets	`ApproximateBounds.norm_wordError_le_budget`
		`ApproximateBounds.budget_append`
		`ApproximateBounds.norm_outputError_le`
		`ApproximateBounds.norm_state_error_le`

SR-15	Two-rate response and equal-rate limit	`ScalarBudgets.response_eq_sum`
		`ScalarBudgets.response_mul_sub`
		`ScalarBudgets.response_of_ne`
		`ScalarBudgets.response_same_succ`
		`StableApproximation.norm_wordError_uniform`
		`StableApproximation.norm_outputError_uniform`

SR-16	Nonexpansive and contractive continuations	`StableApproximation.norm_wordError_nonexpansive`
		`StableApproximation.norm_outputError_nonexpansive`
		`StableApproximation.norm_wordError_contractive`
		`StableApproximation.norm_output_initial_contractive`

SR-17	Actual flow and variation of constants	`ContinuousFlow.hasDerivAt_flow`
		`ContinuousFlow.flow_add`
		`ContinuousFlow.variation_of_constants`
		`ContinuousFlow.homogeneous_unique`
		`ContinuousFlow.forced_unique`

SR-18	Power identities and exponential identities	`ContinuousFlow.mapped_flow_eq_iff_powers_eq`
		`ContinuousFlow.sandwich_flow_zero_iff`
		`ContinuousFlow.compressed_flow_iff_powers`
		`ContinuousFlow.compressed_flow_defect`

SR-19	Continuous hidden elimination	`ContinuousBlockDynamics.hasDerivAt_visible`
		`ContinuousBlockDynamics.hidden_elimination`
		`ContinuousBlockDynamics.continuous_memory_equation`
		`ContinuousBlockDynamics.prepared_memory_equation`

Continued on next page

ID	Claim group	Representative declarations

SR-20	Continuous closure, finite moments, and jets	`ContinuousBlockDynamics.prepared_exact_iff_memory_zero`
		`ContinuousBlockDynamics.prepared_semigroup_iff_memory_zero`
		`ContinuousBlockDynamics.prepared_exact_iff_finite_memory`
		`ContinuousBlockDynamics.prepared_exact_iff_finite_derivatives`
		`ContinuousBlockDynamics.first_return_derivative`

SR-21	Duhamel transport and composition	`ContinuousDefect.duhamel`
		`ContinuousDefect.generatorDefect_zero_iff`
		`ContinuousDefect.evolutionDefect_add`
		`ContinuousDefect.generatorDefect_composite`
		`ContinuousDefect.evolutionDefect_composite`

SR-22	Growth envelopes and scalar response	`ExponentialBudgets.response_of_ne`
		`ExponentialBudgets.response_self`
		`ContinuousApproximation.norm_evolution_le_response`
		`ContinuousApproximation.norm_output_le_response`

SR-23	Uniform continuous stability and initialization	`ContinuousStability.norm_evolution_uniform`
		`ContinuousStability.norm_output_uniform`
		`ContinuousStability.norm_initialization_uniform`
		`ContinuousStability.norm_composite_evolution`

SR-24	Sharp response and small-defect failure	`ContinuousApproxExamples.scalar_response_sharp`
		`ContinuousApproxExamples.unstable_generator_exact`
		`ContinuousApproxExamples.unstable_error_exact`
		`ContinuousApproxExamples.unstable_error_unbounded`

SR-25	Unique three-component geometry	`OrthogonalResolution.components_sum`
		`OrthogonalResolution.unique_components`
		`OrthogonalResolution.norm_sq_components`
		`OrthogonalResolution.candidate_error_sq`
		`OrthogonalResolution.infDist_sq_decomposition`

SR-26	Harmonic realization of the obstruction quotient	`HilbertGlobalization.obstructionEquivHarmonic`
		`HilbertGlobalization.class_zero_iff_component_zero`
		`HilbertGlobalization.same_class_iff_same_harmonic`
		`HilbertGlobalization.exists_best_global_image`

SR-27	Residual bounds retain conditioning and obstruction	`HilbertConditioning.inconsistency_residual`
		`HilbertConditioning.inconsistency_le_residual`
		`HilbertConditioning.conditioned_distance_sq`
		`HilbertConditioning.obstruction_lower_bound`
		`HilbertConditioning.compatible_distance`

SR-28	Finite conditioning and realizable enlargement	`HilbertConditioning.exists_coercivity_finite`
		`FiniteGlobalization.finite_residual_bound_iff_exact`
		`GlobalizationRefinement.obstruction_refinement_sq`
		`GlobalizationRefinement.distance_refinement_sq`
		`GlobalizationExamples.no_uniform_conditioning_over_scalar_family`

Continued on next page

ID	Claim group	Representative declarations

SR-29	Constructed bounded canonical inverse	`MinimumNormReconstruction.restrictionEquiv`
		`MinimumNormReconstruction.image_comp_recover`
		`MinimumNormReconstruction.recover_comp_image`
		`MinimumNormReconstruction.recover_unique`
		`ClosedRangeReconstruction.closedRecover`

SR-30	All least-squares sources and unique minimum norm	`MinimumNormOptimality.candidate_error_decomposition`
		`MinimumNormOptimality.all_least_squares_sources`
		`MinimumNormOptimality.unique_minimum_norm_least_squares`
		`MinimumNormOptimality.exact_source_recovery_iff`
		`MinimumNormOptimality.noisy_source_error_sq`

SR-31	Source conditioning and solver ambiguity	`SourceConditioning.exists_source_coercivity`
		`SourceConditioning.recover_opNorm_bound`
		`SourceConditioning.noisy_source_bound_sq`
		`SourceConditioning.candidate_source_bound_sq`
		`ReconstructionGlobalization.two_condition_reconstruction`

SR-32	Closed range is equivalent to bounded recovery	`ClosedRangeReconstruction.closed_range_iff_coercivity`
		`ClosedRangeReconstruction.bounded_recovery_iff_closed_range`
		`ClosedRangeReconstruction.nonclosed_no_bounded_recovery`
		`ReconstructionExamples.no_uniform_source_conditioning`

SR-33	Native quotient norm and isometry	`MetricObstruction.quotientToHarmonic_norm`
		`MetricObstruction.obstructionIsometry`
		`MetricObstruction.norm_class_eq_infDist`
		`MetricObstruction.distance_classes`
		`MetricObstruction.quotient_inner_preserved`

SR-34	Minimum representatives and intrinsic class of data	`MetricRepresentatives.unique_minimum_representative`
		`MetricRepresentatives.representative_error_energy`
		`IntrinsicGlobalization.projectedClass_opNorm_le`
		`IntrinsicGlobalization.intrinsic_distance_sq`
		`IntrinsicGlobalization.realizable_iff_class_and_residual_zero`
		`IntrinsicGlobalization.intrinsic_candidate_total_error`

SR-35	Bounded quotient transport and reflection	`MetricObstructionTransport.obstructionMap_norm_le`
		`MetricObstructionTransport.boundedObstructionMap_opNorm`
		`MetricObstructionTransport.reverse_bound`
		`MetricObstructionTransport.two_sided_bound`

SR-36	Metric refinement and nonclosed boundary	`MetricRefinement.refinementMap_surjective`
		`MetricRefinement.refinementMap_norm_le`
		`MetricRefinement.quotient_refinement_energy`
		`QuotientMetricBoundaries.zero_norm_iff_closure`
		`QuotientMetricBoundaries.nonclosed_nonzero_zero_norm`
		`QuotientMetricBoundaries.norm_detects_zero_iff_closed`

Continued on next page

ID	Claim group	Representative declarations

SR-37	Constructed normal inverse and energy	`TikhonovConstruction.normalForm_coercive`
		`TikhonovConstruction.normalEquiv_apply`
		`TikhonovConstruction.normal_resolvent`
		`TikhonovConstruction.resolvent_normal`
		`TikhonovConstruction.normal_equation`
		`TikhonovConstruction.solution_energy`

SR-38	Objective optimality and sharp fixed-parameter bounds	`TikhonovOptimality.objective_increment`
		`TikhonovOptimality.unique_minimizer`
		`TikhonovBounds.resolvent_opNorm_bound`
		`TikhonovBounds.regularized_opNorm_bound`
		`TikhonovBounds.approximate_solver_bound`
		`TikhonovBounds.source_kernel_orthogonal`

SR-39	Bias, noise, and kernel content	`TikhonovBias.filter_decomposition`
		`TikhonovBias.noisy_source_error_squared`
		`TikhonovBias.full_conditioned_error_squared`
		`TikhonovBias.source_condition_bias_bound`

SR-40	Resolvent sensitivity and irreducible local error	`TikhonovPerturbation.resolvent_difference`
		`TikhonovPerturbation.regularized_operator_difference_bound`
		`TikhonovGlobalization.regularized_pseudoinverse_bias`
		`TikhonovGlobalization.regularized_intrinsic_error`

SR-41	Density and strong identifiable-source convergence	`TikhonovDensity.gram_range_closure`
		`TikhonovDensity.source_approximation_bound`
		`TikhonovConvergence.tendsto_source`
		`TikhonovConvergence.exact_limit_iff_identifiable`
		`TikhonovConvergence.tendsto_error_norm`

SR-42	Noise schedules and inexact strong limits	`NoisyRegularization.noisy_source_tendsto`
		`NoisyRegularization.noisy_inexact_solver_tendsto`
		`NoisyRegularization.square_schedule_consistent`

SR-43	Operator-norm convergence exactly at closed range	`UniformRegularization.closed_range_of_approximate_recovery`
		`UniformRegularization.nonclosed_sourceError_norm`
		`UniformRegularization.conditioned_sourceError_norm`
		`UniformRegularization.uniform_convergence_iff_closed_range`

SR-44	Fitted-data limit and absent source limit	`RegularizedDataLimit.fitted_adjoint_identity`
		`RegularizedDataLimit.fitted_data_tendsto`
		`RegularizedDataLimit.noisy_fitted_data_tendsto`
		`RegularizedDataLimit.unrealizable_no_source_limit`

SR-45	Actual residual curve and unique interior root	`DiscrepancyResidual.source_norm_antitone`
		`DiscrepancyResidual.residual_strictMono`
		`DiscrepancyResidual.residual_constant_of_adjoint_zero`
		`DiscrepancyResidual.residual_difference_bound`
		`DiscrepancySelection.exists_unique_parameter_of_distance`

SR-46	Norm domination converts residual convergence to source convergence	`DiscrepancyConvergence.norm_domination_of_residual`
		`DiscrepancyConvergence.source_distance_estimate`
		`DiscrepancyConvergence.norm_control_residual_tendsto`

Continued on next page

ID	Claim group	Representative declarations

SR-47	Accepted bands and the total small-data rule	`DiscrepancyRule.selectedSource_accepted`
		`DiscrepancyRule.accepted_consistent`
		`DiscrepancyRule.selectedSource_consistent`
		`DiscrepancyRule.accepted_error_norm_limit`
		`DiscrepancyRule.accepted_source_rate`

SR-48	Discrepancy endpoints and scalar formulas	`DiscrepancyBoundaries.no_parameter_above_data`
		`DiscrepancyBoundaries.no_parameter_at_data_norm`
		`DiscrepancyBoundaries.zero_residual_iff_zero_data`
		`DiscrepancyBoundaries.identity_parameter_formula`

SR-49	Normal residual, two correction scales	`InexactResidual.correction_identity`
		`InexactResidual.correction_bound`
		`InexactResidual.image_correction_bound`
		`InexactResidual.fitting_residual_difference`

SR-50	Protected measured discrepancy margins	`InexactResidual.measured_band_transfer`
		`InexactDiscrepancy.measured_certificate`
		`InexactToleranceCertificates.protected_window_exists_iff`
		`InexactToleranceCertificates.protected_midpoint`

SR-51	Near-accepted consistency and kernel leakage	`InexactDiscrepancy.NearAccepted`
		`InexactDiscrepancy.near_kernel_leakage`
		`InexactConsistency.near_consistent`
		`InexactConsistency.near_error_norm_limit`
		`InexactConsistency.near_kernel_tendsto_zero`

SR-52	Squared proof-carrying discrepancy certificate	`InexactToleranceCertificates.squared_enclosure_iff`
		`InexactToleranceCertificates.squared_image_budget`
		`InexactToleranceCertificates.PositiveCertificate`
		`InexactToleranceCertificates.certificate_sound`
		`InexactToleranceCertificates.certificate_consistent`

SR-53	Effective model-noise allowance	`OperatorUncertainty.action_error`
		`OperatorUncertainty.effective_data_noise`
		`OperatorUncertainty.fitting_norm_difference`
		`OperatorUncertainty.measured_true_fitting_envelope`
		`OperatorUncertaintyExamples.inflated_noise_sharp`

SR-54	Transport of actual normal residuals	`OperatorResidualTransport.normal_residual_difference`
		`OperatorResidualTransport.normal_residual_available_bound`
		`OperatorResidualTransport.true_normal_residual_bound`

SR-55	Strong consistency without matching model kernels	`ChangingDiscrepancy.norm_control`
		`ChangingDiscrepancy.near_true_residual`
		`ChangingConsistency.projected_consistent`
		`ChangingConsistency.source_error_limit`
		`ChangingConsistency.true_kernel_leakage_tendsto_zero`

SR-56	Available-model certificate and selected-source soundness	`OperatorCertificates.Certified`
		`OperatorCertificates.certified_near`
		`OperatorCertificates.certified_consistent`
		`OperatorCertificates.inflated_feasibility`
		`OperatorCertificates.selected_from_allowances_consistent`

SR-57	Candidate-dependent self-bounding	`GeneralizedDiscrepancy.regularized_norm_domination` `GeneralizedDiscrepancy.accepted_norm_control`

Continued on next page

ID	Claim group	Representative declarations

SR-58	Generalized true residual and strong consistency	`GeneralizedDiscrepancy.accepted_true_residual`
		`GeneralizedDiscrepancy.accepted_consistent`
		`GeneralizedInexact.near_true_residual`
		`GeneralizedInexact.near_consistent`
		`GeneralizedInexact.near_source_condition_rate`

SR-59	Relative feasibility and noncomputable selection	`GeneralizedDiscrepancySelection.positive_parameter_source`
		`GeneralizedDiscrepancySelection.exists_accepted`
		`GeneralizedDiscrepancySelection.selectedSource`
		`GeneralizedFeasibility.relative_accuracy_of_scaled_error`
		`GeneralizedDiscrepancyConsistency.selected_consistent`

SR-60	Inexact generalized sources and the zero-model boundary	`GeneralizedInexact.NearAccepted`
		`GeneralizedInexact.near_consistent`
		`GeneralizedDiscrepancyExamples.identity_candidate_accepted`
		`GeneralizedDiscrepancyExamples.zero_model_not_accepted`
		`GeneralizedDiscrepancyExamples.zero_model_valid_true_bounds`
		`GeneralizedDiscrepancyExamples.zero_model_relative_accuracy_fails`

SR-61	Redundant split calibration identifies the operator	`CalibrationFrame.Frame.factor_through_frame`
		`CalibrationFrame.Frame.difference_factorization`
		`CalibrationFrame.Frame.measured_operator_bound`
		`CalibrationFrame.Frame.measured_operator_bound_of_analysis_bound`
		`CalibrationFrame.Frame.exact_calibration_identifies`

SR-62	Calibrated recovery and conditional selected sources	`CalibrationRecovery.generated_operator_bound`
		`CalibrationRecovery.generatedDelta_tendsto_zero`
		`CalibrationRecovery.calibrated_consistent`
		`CalibrationRecovery.calibrated_error_norm_limit`
		`CalibrationSelection.exists_accepted_from_calibration`
		`CalibrationSelection.calibrated_selected_consistent`

SR-63	Hierarchical and approximate reconstruction frames	`CalibrationHierarchy.comp`
		`CalibrationHierarchy.comp_analysis_norm_le`
		`CalibrationHierarchy.hierarchical_operator_bound`
		`ApproximateCalibration.approximate_factorization`
		`ApproximateCalibration.approximate_frame_bound`
		`ApproximateCalibration.measured_approximate_operator_bound`

SR-64	A missed calibration direction permits unbounded error	`CalibrationExamples.partial_not_surjective`
		`CalibrationExamples.unseen_comp_partial`
		`CalibrationExamples.unseen_norm`
		`CalibrationExamples.arbitrarily_large_unseen_error`
		`CalibrationExamples.perfect_partial_calibration_unbounded`