PAPER II

Projector Resolved Seed Calculus II

Refinement towers, weighted transfer, and integral separation of operator packets

Current formal scope and evidence, followed by the inherited declaration-level walkthrough and exact source excerpts.

Komorebi P.

Lean and evidence guide • 19 October 2026

How to use this volume

This guide separates the current formal development from the inherited 25-row contract. The current overview records exact source scope and verification status. The later walkthrough preserves the canonical source architecture, representative proofs, and historical evidence with their original boundaries. **Current source snapshot (19 October 2026).** The correspondence records 81 source items, including 53 proof-level items. Its status is `FINAL_SOURCE_BOUND_CORRESPONDENCE_53_OF_53`. The tables retain the recorded states and release gates. The inherited 25-row map remains a separate historical contract.

53 of 53 proof-level source records have kernel checked, semantically accepted scope. No partial or pending proof records are listed in this correspondence. These are source-record counts under their stated hypotheses, not counts of independent discoveries.

Contents

1 Current formal scope and evidence 1
- 1.1 Current proof inventory . . . 1
- 1.2 Release gates . . . 1
- 1.3 The rectifiable analytic endpoints . . . 1
- 1.4 Current statement map . . . 1
- 1.5 Reading the historical walkthrough . . . 3

2 From original contours to canonical values 4
- 2.1 The original tagged integral . . . 4
- 2.2 The geometric length used in the estimate . . . 4
- 2.3 General Cauchy comparison . . . 5
- 2.4 The canonical defect and compatibility . . . 6

3 Reading the inherited formal walkthrough 7
- 3.1 Three useful reading routes . . . 7

4 Lean types as mathematical bookkeeping 8
- 4.1 Definitions, certificates, and algorithms . . . 9

5 Source architecture and entry points 9

6 Exact packets and refinement 10
- 6.1 What a refinement witness supplies . . . 10
- 6.2 A proof walkthrough: recovering a selected finite decomposition . . . 10
- 6.3 Coordinate changes need no parent map . . . 11
- 6.4 Naturality and independently refined blocks . . . 13

7 Residue selectors, Fourier certificates, and CRT 13
- 7.1 The finite Fourier table is an input over general semirings . . . 13
- 7.2 Refining residues and factoring their labels . . . 14

8 Weighted inversion: three separate branches 14
- 8.1 The finite geometric identity . . . 15

8.2 The filtration is deliberately minimal . . . 15
8.3 Written products and chronological paths . . . 16
8.4 What is actually retained . . . 16
8.5 The analytic branch and the algorithm contract . . . 18

9 Transfer zeta: from independent walks to rationality 18
9.1 Why a coefficient has a finite enumeration . . . 18
9.2 The recurrence from the first edge . . . 19
9.3 A second construction: determinant and adjugate . . . 20
9.4 The recurrence is extracted from the proved identity . . . 20

10 Evaluated separation and lattice precision 21
10.1 A proof walkthrough: the square identity without division . . . 21
10.2 What the formal lattice means . . . 22
10.3 Precision delay as transport of a witness . . . 22

11 Concrete Smith fibres and the dyadic sign model 23
11.1 The solution fibre is an actual type of solutions . . . 23
11.2 A proof walkthrough: counting an admissible fibre . . . 24
11.3 The abstract extension keeps its certificate boundary . . . 25
11.4 Rational sign projectors and local denominator weight . . . 25
11.5 The Smith phase identity has only algebraic scope . . . 26

12 Typed defects: domain and codomain stay visible 26
12.1 The telescoping mechanism . . . 26
12.2 The operator norm estimate . . . 27
12.3 Resolvents from the inverse equations actually used . . . 27
12.4 What remains in the written companion . . . 28

13 Reproducing the historical formal project 28
13.1 Start with the preserved pins . . . 29
13.2 The audit gate and its trust boundary . . . 29
13.3 A minimal session for source inspection . . . 30
13.4 Common failure modes . . . 30

14 Evidence retained and evidence newly checked 31
14.1 A practical rule for reporting success . . . 32

15 The inherited contract of 25 rows 33
15.1 Row 1: Refinement map . . . 33
15.2 Row 2: Nested refinement theorem . . . 33
15.3 Row 3: Exact coordinate refinement . . . 34
15.4 Row 4: Perfect reconstruction across levels . . . 34
15.5 Row 5: Transfer-block refinement . . . 35
15.6 Row 6: Residue projector algebra . . . 35
15.7 Row 7: Root-of-unity realization . . . 35
15.8 Row 8: Polyphase refinement . . . 36
15.9 Row 9: Dyadic-odd packet factorization . . . 36
15.10 Row 10: Certified weighted inverse . . . 37

15.11 Row 11: Finite precision and weighted pruning 37
15.12 Row 12: Finite-graph transfer-zeta rationality 38
15.13 Row 13: Linear recurrence 38
15.14 Row 14: Evaluated Bezout spectral projector 39
15.15 Row 15: Separation weight 39
15.16 Row 16: Projector precision delay 40
15.17 Row 17: Odd-adic versus dyadic sign behavior 40
15.18 Row 18: Concrete ZMod Smith-weighted congruence solve 41
15.19 Row 19: Power-defect identity 41
15.20 Row 20: Power-defect norm bound 42
15.21 Row 21: Resolvent-defect identity 42
15.22 Row 22: Smith phase inversion identity 43
15.23 Row 23: Residue analysis and synthesis algorithm 43
15.24 Row 24: Weighted finite-precision inverse algorithm 43
15.25 Row 25: Transfer-zeta walk construction algorithm 44

A Navigation in the final traditional paper 45

B Source imports and pinned dependencies 46

C Source excerpts and reproduction inventory 47

1 Current formal scope and evidence

The current Lean project is `formal/current/`. Its correspondence records the exact current statement and proof text, source hashes, fully qualified targets, and the evidence attached to each item. The complete machine-readable record is `formal/coverage/Paper_II_Current_Correspondence_2026-10-19.json`.

Current source snapshot (19 October 2026). The correspondence records 81 source items, including 53 proof-level items. Its status is `FINAL_SOURCE_BOUND_CORRESPONDENCE_53_OF_53`. The tables retain the recorded states and release gates. The inherited 25-row map remains a separate historical contract.

53 of 53 proof-level source records have kernel checked, semantically accepted scope. No partial or pending proof records are listed in this correspondence. These are source-record counts under their stated hypotheses, not counts of independent discoveries.

1.1 Current proof inventory

Recorded status	Items
Kernel checked, scope accepted	53

Exact statement comparison, semantic acceptance, aggregate compilation and kernel replay are distinct evidence. Publication copies their recorded states. It performs none of these mathematical acceptance steps.

1.2 Release gates

Gate	Recorded state
exact source aggregate compile	PASS
owned kernel type body replay	PASS
standard axiom and runtime helper closure	PASS
current manuscript semantic acceptance	53_COMPLETE_SCOPES_ACCEPTED

1.3 The rectifiable analytic endpoints

The rectifiable analytic endpoints use a representation whose velocity length is bounded by the original geometric length. This inequality supplies the manuscript estimate. Equality of the two lengths is not required. The accepted correspondence identifies the exact canonical endpoints and their hypotheses.

1.4 Current statement map

The following table lists the proof-level source records. The correspondence gives exact declaration names, statement and proof hashes, hypotheses and inherited evidence. Mathematical definitions marked `noncomputable` do not supply an executable solver.

Current source label	Recorded status
`thm:children_transitivity`	Kernel checked, scope accepted
`thm:coordinate_coherence`	Kernel checked, scope accepted
`cor:orthogonal_isometry`	Kernel checked, scope accepted
`thm:block_refinement`	Kernel checked, scope accepted
`cor:block_support_refinement`	Kernel checked, scope accepted
`thm:common_refinement`	Kernel checked, scope accepted
`prop:coarsest_common_refinement`	Kernel checked, scope accepted
`prop:residue_algebra`	Kernel checked, scope accepted
`thm:root-of-unity`	Kernel checked, scope accepted
`thm:character_realization`	Kernel checked, scope accepted
`thm:residue_refinement`	Kernel checked, scope accepted
`thm:crt_factorization`	Kernel checked, scope accepted
`cor:dyadic_odd_coordinates`	Kernel checked, scope accepted
`thm:weighted_inverse`	Kernel checked, scope accepted
`thm:block_path_formula`	Kernel checked, scope accepted
`cor:minimum_path_weight`	Kernel checked, scope accepted
`cor:acyclic_inverse`	Kernel checked, scope accepted
`cor:inverse_stability`	Kernel checked, scope accepted
`cor:finite_precision`	Kernel checked, scope accepted
`thm:weighted-inverse`	Kernel checked, scope accepted
`cor:finite-precision`	Kernel checked, scope accepted
`thm:transfer_rationality`	Kernel checked, scope accepted
`cor:path_recurrence`	Kernel checked, scope accepted
`cor:multigraded_series`	Kernel checked, scope accepted
`thm:transfer-zeta`	Kernel checked, scope accepted
`cor:zeta-recurrence`	Kernel checked, scope accepted
`thm:bezout_projector`	Kernel checked, scope accepted
`thm:several_projectors`	Kernel checked, scope accepted
`cor:resultant_bound`	Kernel checked, scope accepted
`thm:bezout-projector`	Kernel checked, scope accepted
`thm:precision-delay`	Kernel checked, scope accepted
`thm:gluing_weight`	Kernel checked, scope accepted
`cor:length_and_weight`	Kernel checked, scope accepted

Current source label	Recorded status
`cor:binary_gluing`	Kernel checked, scope accepted
`cor:integral_split`	Kernel checked, scope accepted
`thm:gluing_refinement`	Kernel checked, scope accepted
`thm:precision_delay`	Kernel checked, scope accepted
`prop:sign_projectors`	Kernel checked, scope accepted
`prop:character_integrality`	Kernel checked, scope accepted
`prop:dyadic-sign`	Kernel checked, scope accepted
`prop:weight_from_inclusion`	Kernel checked, scope accepted
`thm:smith_congruence`	Kernel checked, scope accepted
`cor:smith_obstruction`	Kernel checked, scope accepted
`thm:smith-solve`	Kernel checked, scope accepted
`thm:power_defect`	Kernel checked, scope accepted
`cor:power_norm`	Kernel checked, scope accepted
`cor:defect-norm`	Kernel checked, scope accepted
`cor:polynomial_defect`	Kernel checked, scope accepted
`thm:resolvent_defect`	Kernel checked, scope accepted
`thm:holomorphic_defect`	Kernel checked, scope accepted
`thm:exponential_defect`	Kernel checked, scope accepted
`thm:multilevel_defect`	Kernel checked, scope accepted
`cor:exact_compatibility`	Kernel checked, scope accepted

1.5 Reading the historical walkthrough

The remaining chapters explain the canonical v1.1.1 Lean component preserved inside manuscript release v1.2.1. Their counts, excerpts and certificate boundaries refer to that snapshot. Use the current map above for later constructions and current verification status. Historical reports retain their original dates and evidence claims.

2 From original contours to canonical values

The analytic part of the current project follows the contours that occur in the traditional paper. A contour is a continuous complex path of bounded variation on the closed unit interval. A cycle is a finite family of closed contours. An occurrence in that family contributes once to every sum, so a repeated contour retains its multiplicity. The construction permits constant paths, retracing, and an empty family.

The proof has three distinct tasks. It identifies the classical contour integral, controls the original geometric length, and proves that admissible contours give the same functional calculus value. Each task has an explicit endpoint in the current sources. The identity of a chosen integral is kept separate from independence of that choice.

2.1 The original tagged integral

For a path γ and a continuous function F on its trace with values in a complex Banach space, the tagged sums are

$$S(\gamma, F; P, \tau) = \sum_{j=0}^{m-1} (\gamma(t_{j+1}) - \gamma(t_j)) F(\gamma(\tau_j)), \qquad t_j \le \tau_j \le t_{j+1}.$$

The factor on the left acts by complex scalar multiplication. The partition is monotone, begins at 0, and ends at 1. Repeated partition points are allowed. The definition of the integral quantifies over every such tagged partition with sufficiently small mesh. Arbitrarily fine partitions are constructed, and the value is proved unique. The definition therefore expresses an actual limit of the original path increments.

For an absolutely continuous path, the difference between a tagged increment and its velocity integral is an integral of the oscillation of F along that subinterval. Its norm is bounded by that oscillation times the integral of the speed. Summing over the partition and using uniform continuity on the compact trace proves convergence of all the tagged sums to the Bochner velocity integral.

An arbitrary continuous rectifiable path need not be absolutely continuous in its original parameter. The project constructs its normalized arclength path and a continuous monotone map from the original parameter interval. Tagged partitions are transported through this map. Their increments and tagged function values agree exactly. Uniform continuity controls the transported mesh, including flat portions of the parameter map. The zero length case is handled separately. This proves equality between the original contour integral and the represented velocity integral without assuming a comparison certificate.

The source `ReleaseRectifiableRSError.lean` contains the tagged definition, existence and uniqueness results, monotone transport, and the finite cycle comparison. In particular, the original winding expression obtained by integrating $(z - w)^{-1}$ agrees with the represented winding whenever w avoids the trace.

2.2 The geometric length used in the estimate

Write $L(\gamma)$ for the total variation of the original path. The normalized arclength path is Lipschitz with constant $L(\gamma)$. Its coordinate velocity consequently satisfies

$$\int_0^1 \|v_\gamma(t)\|\, dt \le L(\gamma).$$

For a cycle, these inequalities add over its occurrences. This is the inequality needed by the manuscript. The proof does not require an additional theorem asserting equality between integrated speed and variation.

Combining this inequality with the actual cycle integral estimate gives

$$\left\| \int_\Gamma F(z)\,dz \right\| \le \operatorname{length}(\Gamma) \sup_{z\in\Gamma} \|F(z)\|,$$

where the length on the right is the original geometric length with multiplicities. Each supremum of norms is defined to be zero when its trace is empty. The separate supremum factors in Theorem 11.6 are retained by `ReleaseRectifiableDefect.lean`. They are not replaced by an unspecified product bound.

2.3 General Cauchy comparison

The represented cycle has integrable velocities. Its curves are absolutely continuous, and composition with a holomorphic scalar function on a neighborhood of the compact trace is absolutely continuous. The chain rule and the fundamental theorem of calculus therefore give

$$\int_\Gamma G'(z)\,dz = 0$$

for a supplied holomorphic primitive G. This scalar primitive result also proves local constancy of the actual normalized winding integral away from the trace. Applying it to an inverse kernel on an enclosing ball proves that winding vanishes sufficiently far from the trace.

For a holomorphic function F with values in a complex Banach space on an arbitrary open set U, define

$$H_{1,\Gamma}(w) = \int_\Gamma [w,z]F\,dz, \qquad H_{2,\Gamma}(w) = \int_\Gamma (z-w)^{-1}F(z)\,dz.$$

The divided difference $[w,z]F$ takes the value $F'(w)$ on the diagonal. The first function is holomorphic on U. The second is holomorphic away from the trace. Compact trace bounds justify differentiation under the actual Banach integral in both cases.

Suppose that two cycles Γ and Λ lie in U and have equal winding outside U. Use $H_{1,\Gamma} - H_{1,\Lambda}$ on U and $H_{2,\Gamma} - H_{2,\Lambda}$ outside U. At an exterior point, both traces are absent from a neighborhood. Local constancy of winding and the divided difference identity make the two expressions agree there. The resulting function is entire, even when U is disconnected or unbounded. Far from the traces it tends to zero. Banach valued Liouville then makes it identically zero.

To recover the integral of F, replace it by $(z-a)F(z)$ for any $a \in U$. Its divided difference at a equals $F(z)$, including at $z = a$. Thus no point outside the traces has to be chosen. The empty domain case is handled directly by the zero bound on an empty trace. The resulting theorem is

$$\int_\Gamma F(z)\,dz = \int_\Lambda F(z)\,dz.$$

Its only geometric inputs are the actual finite cycles and equality of winding outside the open domain. It assumes neither a general Cauchy conclusion nor contour independence.

2.4 The canonical defect and compatibility

For a bounded operator T, admissibility records an open holomorphy domain, a genuine cycle in that domain and the resolvent set, winding one on the spectrum, and winding zero outside the domain. These are input data. No functional calculus value or independence equation is a field of that record.

Two admissible choices are compared on the union of their holomorphy domains intersected with the resolvent set. Outside this comparison domain, their windings are both zero or both one. General Cauchy comparison proves equality of their normalized Dunford integrals. A canonical value can therefore be selected from any nonempty family of admissible data and then represented by every admissible original contour.

The original contour supplied in Theorem 11.6 provides this witness for both operators. After rewriting the two canonical values through that contour, the actual resolvent defect identity proves the manuscript's original tagged integral formula and its geometric length bound. No extra existence assumption is needed for this theorem.

For compatibility, the two endpoint operators may use different admissible contours. When the operator defect is zero, their two intertwined resolvents agree on the overlap of their resolvent sets. They glue to one holomorphic map on the union. General Cauchy comparison again identifies the integrals. Applying this naturality result to the total map of a finite tower proves Corollary 11.10 whenever its endpoint expressions are defined. The formal witnesses express this definedness. They do not assert compatibility as a premise.

The current coverage statement concerns the manuscript's enumerated proof scopes. It does not assert a separate theorem constructing an admissible contour for every possible operator and holomorphic germ, or an identification with a different software library's calculus API.

The relevant final endpoints are in `ReleaseCanonicalHFC.lean`, `ReleaseCanonicalNaturality.lean`, and `ReleaseCanonicalDefect.lean`. Their exact source identity and execution receipts are recorded separately from this exposition.

3 Reading the inherited formal walkthrough

The traditional paper develops the mathematical theory and its written proofs. The visual guide provides finite models for its identities, precision statements, and boundaries. This guide explains how to read and use the preserved Lean component: the types accepted by each theorem, the objects actually constructed, the obligations supplied by a caller, and the evidence attached to the source. These three companions have complementary roles.

The component described in this walkthrough is the historical version 1.1.1, preserved within canonical manuscript release 1.2.1. Current additions and their boundaries are recorded in Section 1. In the delivered package it is located at `source/canonical_v1.2.1/formalization/lean/`. The final traditional edition incorporates a broader written development. Its ordinary proofs of lattice gluing, necessary precision delay, convergence in an inverse limit, general DVR arithmetic, holomorphic and exponential defects, and multilevel identities are not converted into Lean proofs by appearing beside formally mapped results.

The fixed formal contract contains 25 rows: 21 numbered canonical statements, one Smith phase identity, and three mathematical algorithm specifications. The appendix crosswalk identifies their locations in the integrated traditional edition. Throughout this guide, "row" means a row of the canonical formal map, never a count of every theorem in the final traditional paper.

Quantity	What is counted
15 Lean source files	Four shared foundation modules, eight substantive Paper II modules, two aggregate imports, and one audit module.
5,603 source lines	Physical lines in those files, including comments and blank lines.
25 map rows	Statement families, constructions, and specifications in the frozen contract.
160 unique targets	Distinct mapped declarations; not all are theorem declarations.
171 target occurrences	Targets counted again when reused by another contract row.
1,034 namespace entries	Entries reported by the historical compiled namespace scan, including generated auxiliaries.

A name count is not a measure of mathematical depth. A short wrapper can collect several substantial conclusions. A large structure can expose a difficult hypothesis without constructing its witness. The useful unit is a complete type together with its proof, its dependencies, and an accurate comparison with the prose statement.

3.1 Three useful reading routes

For a first reading, follow Sections 4–12: exact coordinates lead to weighted transfer, then arithmetic and typed defects. For a coverage review, use the crosswalk to the final paper in Appendix A, read the corresponding row in Section 15, and inspect its source type. For a reproduction attempt, read Section 13 before running any build command. The source checksum check and the compiler audit establish different things.

The main walkthroughs use exact source excerpts, with explanatory equations outside the listings. Names are case sensitive. `PowerSeries`, `Polynomial`, `LinearMap`, and `ContinuousLinearMap` are different types. Similar notation is not a license to replace one with another in a theorem application.

4 Lean types as mathematical bookkeeping

The basic theorem syntax places assumptions before a colon and a conclusion after it. A term after `:=` supplies the definition or proof. A `by` block is tactic syntax that elaborates a proof term. A successful elaboration still leaves a semantic question for the reader: does this exact type express the intended mathematical assertion?

Expression	Reading in this component
`[Semiring R]`	A typeclass instance supplying semiring operations and laws. Subtraction need not exist.
`[Ring R]`	A ring, possibly noncommutative; multiplicative order remains significant.
`[CommSemiring R]`	Commutative multiplication is available, as in the general Fourier certificate.
`[Module R M]`	A scalar action with its laws; the scalar type is explicitly named.
`M →ₗ[R] N`	An R-linear map from M to N, including additive and scalar compatibility.
`Module.End R M`	The endomorphism ring or semiring of a single module.
`X →L[k] Y`	A continuous linear map with operator norm in the relevant normed setting.
`M ≃ₗ[R] N`	A linear equivalence, including both inverse laws.
`R →+* S`	A ring or semiring homomorphism; it preserves the operations used to transport an inverse.
`[Fintype E]`	A finite enumeration of all elements of E.
`Fin n`	Natural numbers strictly below n, bundled with their bound.
`ZMod n`	Integers modulo n; the nonzero modulus hypotheses matter for finite cardinality.
`AddSubgroup M`	An additive subgroup, without automatically being a valuation-ring lattice.
`Nonempty A`	Existence of a term of type A; it does not by itself supply a computational search procedure.
`Prop`	The sort of propositions, used for certificates whose fields are proof obligations.

Parenthesized arguments are normally supplied explicitly. Braced arguments can be inferred. Arguments in square brackets request typeclass synthesis. Section variables become parameters of later declarations when used. Therefore a source theorem must be read with its section context. In this project, `autoImplicit = false` and `relaxedAutoImplicit = false` prevent undeclared identifiers from silently becoming new implicit parameters.

The common `constructor` tactic splits a structure or conjunction into its fields. `rcases` opens a supplied witness into its data and proofs. `rw` rewrites using proved equalities. `simp` applies simplification rules and produces a proof term. `ext` reduces equality of structured maps to pointwise equality. `exact` supplies a term of the required type. `rfl` uses definitional equality. None of these words is a substitute for a missing mathematical hypothesis.

4.1 Definitions, certificates, and algorithms

A structure such as `TwoSidedInverse x y` records equations about already supplied x and y. A theorem that accepts this structure does not discover y. By contrast, the rationality branch for finite graphs defines a denominator and numerator from the graph and proves their properties without accepting rationality as an input.

`noncomputable` does not mean unproved. It marks a definition that is not offered as executable code through Lean's ordinary computational evaluation. This component uses noncomputable choices from finite sets, instances for finite types, determinants, adjugates, and constructions of least exponents. Their mathematical properties can be proved even though a standalone executable implementation is not supplied.

The three algorithm rows certify mathematical contracts relating inputs to outputs. They do not certify the companion Python package, its storage layout, its numerical residuals, or its performance. Even where a finite enumeration is executable in principle, the contract contains no general complexity estimate or extraction theorem.

5 Source architecture and entry points

In the historical snapshot, the main entry point is `import KomorebiVerification.PaperII`. It imports the eight substantive modules but not the audit command. The root module `KomorebiVerification.lean` imports the aggregate for Lake's default library target. The standalone audit imports the substantive modules explicitly and inspects the resulting environment.

Module or layer	Responsibility
Shared foundation	`ProjectorCore`, `ExactCoordinates`, `FibreRefinement`, and `ResidueProjectors`: complete finite projector families, range coordinates, masks, and refinement.
`NestedRefinement`	Parent maps, absorption, child uniqueness, exact coordinate equivalences, cocycles, naturality, and block aggregation.
`ResidueFourierCRT`	Division-free masks, finite Fourier certificates, primitive-root specialization, polyphase refinement, and CRT labels.
`WeightedTransfer`	Ordered geometric inverses, normed Neumann inverses, multiplicative filtrations, and precise word pruning.
`TransferZeta`	Finite weighted walks, exact enumeration, first-edge recurrence, determinant rationality, and coefficient recurrence.
`ResultantSeparation`	Evaluated Bézout identities, numerator and kernel actions, denominator clearing, and certificate-based precision delay.
`SmithDyadic`	Concrete modular fibres, abstract certified products, rational sign matrices, local denominator weights, and the Smith-phase formula.
`TypedLinearDefects`	Fine-to-coarse maps between distinct spaces, power and resolvent defects, polynomial compatibility, and operator-norm estimates.
`LinearDefects`	Ring-valued counterparts and auxiliary inverse identities.
`KernelAudit`	Representative axiom reports and the fail-closed namespace gate.

The main mathematical dependencies are modular. Coordinate refinement does not require a valuation. Weighted pruning needs a filtration but not a graph incidence law. Transfer zeta counts graph walks but uses integer formal power series, not operator norms. Smith fibres work in finite modular arithmetic. The typed defect module is independent of a particular packet construction: a caller may use a coarsening map, but the theorem accepts any map of the stated type.

6 Exact packets and refinement

Rows 1–5 start from finite complete projector families. A family $P = (P_i)$ satisfies $P_i^2 = P_i$, $P_i P_j = 0$ for $i \neq j$, and $\sum_i P_i = I$. Orthogonality here is algebraic annihilation. An inner product, self-adjointness, and topology are absent from these assumptions. The canonical formal family also permits zero projectors, whereas the expanded traditional packet terminology requires nonzero members.

The packet space is the product of actual ranges,

$$\mathcal{P}(P) = \prod_i \operatorname{range}(P_i).$$

Its elements carry their proofs of membership in a range. Analysis sends x to $(P_i x)_i$. Synthesis sums the underlying vectors. Completeness proves synthesis after analysis. Pairwise annihilation and idempotence prove analysis after synthesis. Both maps are independently defined, so the second inverse law has mathematical content: every tuple in the product of ranges is accepted.

6.1 What a refinement witness supplies

A parent map $\pi : J \to I$ assigns fine labels to coarse labels. The finite set `children parent a` is the fibre $\pi^{-1}(a)$. The refinement witness records surjectivity and exact equality of coarse maps with sums of their fine children.

Source: `KomorebiVerification/PaperII/NestedRefinement.lean`.

```
structure Refines
    (coarse : LinearProjectorFamily R M ι)
    (fine : LinearProjectorFamily R M κ)
    (parent : κ → ι) : Prop where
  surjective : Function.Surjective parent
  coarse_eq_sum_children :
    ∀ a, coarse.proj a = ∑ b ∈ children parent a, fine.proj b
```

The structure does not construct a parent map between arbitrary families. To apply its theorems, a caller supplies a map and proves these two fields. The resulting absorption identities are genuine consequences. If $\pi(j) = i$, then

$$P_i Q_j = Q_j = Q_j P_i;$$

if $\pi(j) \neq i$, both products vanish. Expand P_i as the sum of its children, distribute composition, and use orthogonality of the fine family. Transitivity follows by applying this reasoning through two successive parent maps.

6.2 A proof walkthrough: recovering a selected finite decomposition

The most general uniqueness theorem in row 2 does not use the refinement witness. Suppose a selected finite set S of packet vectors sums to x, with $v_j \in \operatorname{range}(P_j)$ for $j \in S$. Applying P_i kills all terms except

v_i, so $v_i = P_i x$. The exact proof shows which hypotheses enter.

Source: `KomorebiVerification/PaperII/NestedRefinement.lean`.

```
theorem finitePacket_decomposition_unique
    (P : LinearProjectorFamily R M κ) (s : Finset κ) (x : M)
    (packets : κ → M)
    (hrange : ∀ b ∈ s, packets b ∈ LinearMap.range (P.proj b))
    (hsum : ∑ b ∈ s, packets b = x) :
    ∀ b ∈ s, packets b = P.proj b x := by
  classical
  intro b hb
  have hself : P.proj b (packets b) = packets b := by
    rcases hrange b hb with ⟨y, hy⟩
    rw [← hy]
    exact P.idempotent b y
  have hproject : P.proj b (∑ j ∈ s, packets j) = packets b := by
    rw [map_sum]
    calc
      (∑ j ∈ s, P.proj b (packets j)) = P.proj b (packets b) := by
        apply Finset.sum_eq_single_of_mem b hb
        intro j hj hjb
        rcases hrange j hj with ⟨y, hy⟩
        rw [← hy]
        exact P.orthogonal hjb.symm y
      _ = packets b := hself
  calc
    packets b = P.proj b (∑ j ∈ s, packets j) := hproject.symm
    _ = P.proj b x := congrArg (P.proj b) hsum
```

First `rcases hrange b hb with` extracts a preimage y for the selected packet. Rewriting the packet as $P_b y$ makes idempotence applicable. Next `map_sum` distributes P_b across the finite sum. `Finset.sum_eq_single_of_mem` leaves the diagonal term. Each other term vanishes after its own range witness is opened. Finally `congrArg` applies P_b to the supplied sum equality.

The conclusion is restricted to $b \in S$. No conclusion is drawn about values of the packet function outside that finite set. The source family is complete as a bundled object, but the proof of this result never uses its completeness field. This explains the map's "Stronger" classification without suggesting a theorem about arbitrary infinite sums.

6.3 Coordinate changes need no parent map

Let A_P and S_P denote analysis and synthesis. For any two complete finite families on the same module, define

$$R_{P\to Q} = \mathsf{A}_Q \mathsf{S}_P, \qquad R_{Q\to P} = \mathsf{A}_P \mathsf{S}_Q.$$

The source calls these maps refinement and coarsening.

Source: `KomorebiVerification/PaperII/NestedRefinement.lean`.

```
variable (coarse : LinearProjectorFamily R M ι)
variable (fine : LinearProjectorFamily R M κ)
```

```
/-- Resolve coarse range coordinates by synthesis followed by fine analysis. -/
def refineCoordinates : PacketSpace coarse →ₗ[R] PacketSpace fine :=
  fine.rangeAnalysis.comp coarse.rangeSynthesis

/-- Regroup fine range coordinates by synthesis followed by coarse analysis. -/
def coarsenCoordinates : PacketSpace fine →ₗ[R] PacketSpace coarse :=
  coarse.rangeAnalysis.comp fine.rangeSynthesis
```

These maps are inverses because $\mathsf{S}_Q\mathsf{A}_Q = I$ and $\mathsf{A}_P\mathsf{S}_P = I$. The proof uses the earlier simplification rules for exact coordinates.

Source: `KomorebiVerification/PaperII/NestedRefinement.lean`.

```
@[simp] theorem coarsenCoordinates_refineCoordinates
    (packets : PacketSpace coarse) :
    coarsenCoordinates coarse fine (refineCoordinates coarse fine packets) =
      packets := by
  simp [coarsenCoordinates, refineCoordinates]

/-- Refinement after coarsening is the identity on fine range coordinates. -/
@[simp] theorem refineCoordinates_coarsenCoordinates
    (packets : PacketSpace fine) :
    refineCoordinates coarse fine (coarsenCoordinates coarse fine packets) =
      packets := by
  simp [coarsenCoordinates, refineCoordinates]

/-- The exact linear equivalence between the two independent range products. -/
def coordinateRefinementEquiv : PacketSpace coarse ≃ₗ[R] PacketSpace fine where
  toFun := refineCoordinates coarse fine
  invFun := coarsenCoordinates coarse fine
  left_inv := coarsenCoordinates_refineCoordinates coarse fine
  right_inv := refineCoordinates_coarsenCoordinates coarse fine
  map_add' := (refineCoordinates coarse fine).map_add
  map_smul' := (refineCoordinates coarse fine).map_smul
```

The short `simp` proofs rest on established inverse laws. They are not assumptions that refinement loses no data. The two packet products are full exact presentations of the same module. Counting the number of labels alone does not measure their dimension: every label contains an entire projector range.

A parent witness adds component formulas. The fine component is obtained by applying the fine projector to its coarse parent packet. The coarse component is the sum of its fine children. These are additional identities about the already defined coordinate equivalence. They do not turn it into averaging, sampling, or truncation.

With a third complete family H, the same inverse laws give the cocycle

$$R_{Q\to H}R_{P\to Q} = R_{P\to H}.$$

The theorem `refineCoordinates_comp` first proves the equality on each packet and then uses extensionality of linear maps. A reader can thus separate the simple conceptual proof from the bookkeeping needed to state it as equality of bundled maps.

6.4 Naturality and independently refined blocks

For $T : M \to N$ with source family P and target family Q, the exact coordinate operator is $\mathsf{A}_Q T \mathsf{S}_P$. Its composition law cancels the middle pair of synthesis and analysis maps. Changing the source and target coordinate systems independently gives the corresponding commuting naturality identity.

The block convention places the output index first:

$$T_{ji} = Q_j T P_i.$$

If source and target families refine separately, a coarse block is the double sum over fine descendants of its source and target labels. The mapped theorem permits different source and target modules, which is stronger than the endomorphism special case. Its composite traceability wrapper also records reconstruction of T from all coarse blocks and from all fine blocks.

The finite algebraic result does not establish sparsity of a computed representation, a block extraction routine with a finite complexity bound, or convergence of an infinite block matrix. These need separate hypotheses and, where intended, separate formal developments.

7 Residue selectors, Fourier certificates, and CRT

Rows 6–9 act on coefficient functions $x : \mathbb{N} \to A$. At a positive modulus N, the residue projector keeps $x(n)$ when $n \bmod N = j$ and returns zero otherwise. Since exactly one residue label matches each n, idempotence, pairwise annihilation, and completeness follow pointwise. No division or roots of unity are required.

This representation is a sequence module, not a formal power series object with a multiplication law. It is sufficient for coefficient masks. A user wishing to apply the result to a particular function space must supply the connection to that space and any closure or convergence facts.

7.1 The finite Fourier table is an input over general semirings

A Fourier realization uses scalars ω, $\omega^\vee$, and η. The weighted residue kernel is

$$\eta \sum_{k=0}^{N-1} (\omega^\vee)^{jk} \omega^{rk}.$$

Over a general commutative semiring the source does not assume that $\omega^\vee$ is an inverse or that $\eta = 1/N$. Instead it accepts a finite orthogonality table with exactly the required conclusion at residue addresses.

Source: `KomorebiVerification/PaperII/ResidueFourierCRT.lean`.

```
structure FiniteRingFourierOrthogonalityCertificate
    (R : Type u) [CommSemiring R] (N : Nat)
    (omega omegaDual eta : R) : Prop where
  omega_pow_order : omega ^ N = 1
  weighted_residue_sum_eq (j r : Fin N) :
    eta * ringFourierResidueKernel N omega omegaDual j r =
      if r = j then 1 else 0
```

The substantive bridge is from that finite table to every coefficient address. The equation $\omega^N = 1$

reduces the exponent n to n mod N. The last line below uses `Fin.ext_iff` to equate residue labels through their underlying natural numbers.

Source: `KomorebiVerification/PaperII/ResidueFourierCRT.lean`.

```
theorem FiniteRingFourierOrthogonalityCertificate.toRingCertificate
    {R : Type u} [CommSemiring R]
    (N : Nat) (hN : 0 < N) (omega omegaDual eta : R)
    (certificate :
      FiniteRingFourierOrthogonalityCertificate R N omega omegaDual eta) :
    RingFourierOrthogonalityCertificate R N omega omegaDual eta := by
  refine ⟨?_⟩
  intro j n
  rw [ringFourierKernel_eq_residueKernel N hN omega omegaDual
    certificate.omega_pow_order j n]
  rw [certificate.weighted_residue_sum_eq]
  congr 1
  simp only [Fin.ext_iff]
```

After this bridge, the average and the mask are equal coefficient by coefficient: the normalized scalar is one at the selected address and zero elsewhere. The proof works with a finite table rather than an infinite family of independent premises.

There is also a stronger field branch. A primitive root of positive order supplies the normalized orthogonality certificate for the inverse root internally. Its field hypothesis is essential to that branch's scope. Root order by itself is not a replacement for the table over arbitrary rings with zero divisors.

7.2 Refining residues and factoring their labels

For positive N and k, the modulus-N selector is the sum of all modulus-Nk selectors whose labels reduce to it. This is exact regrouping of masks. No packet is discarded and there is no approximation error to estimate.

For positive coprime m, n, CRT gives an equivalence between labels modulo mn and pairs of labels modulo m and n. The corresponding selector is the composite of the two independent selectors in either order. For $m = 2^a$ and positive odd n, the factorization into dyadic and odd moduli is a specialization.

These are coordinate facts. They do not prove that rational spectral projectors have integral denominators, that a completed lattice splits, or that solving a lifting problem is possible. The later arithmetic modules use different data for those questions. This separation is useful when reading the same terminology about dyadic and odd moduli in the traditional paper: residue factorization and spectral denominator clearing are distinct statements.

8 Weighted inversion: three separate branches

Rows 10–11 concern $D + H$ in a possibly noncommutative ring. A supplied inverse D^{-1} that works on both sides permits the factorization

$$D + H = D(1 + A), \qquad A = D^{-1}H.$$

Thus an inverse of $1 + A$ is multiplied by D^{-1} on the right. Reversing this order changes the formula in a noncommutative ring. The source parameter named `K` in these declarations means the perturbation H. It is not a fraction field.

Three hypotheses lead to three different results. Nilpotence gives a finite ring identity. A homomorphism that kills a filtration level gives a finite identity after passing to precision. A strict norm bound in a complete normed ring gives an infinite Neumann sum. The formal component does not merge these into an unrestricted adic convergence theorem.

8.1 The finite geometric identity

For $A^N = 0$, define $B = \sum_{n=0}^{N-1}(-A)^n$. The product formulas for finite geometric sums prove both $(1 + A)B = 1$ and $B(1 + A) = 1$.

Source: KomorebiVerification/PaperII/WeightedTransfer.lean.

```
def finiteGeometricInverse {R : Type u} [Ring R] (a : R) (N : ℕ) : R :=
  ∑ n ∈ Finset.range N, (-a) ^ n

/-- If `a ^ N = 0`, the finite geometric candidate is a genuine two-sided inverse of `1 + a`. -/
theorem finiteGeometricInverse_spec_of_pow_eq_zero {R : Type u} [Ring R]
    (a : R) (N : ℕ) (hN : a ^ N = 0) :
    TwoSidedInverse (1 + a) (finiteGeometricInverse a N) := by
  have hneg : (-a) ^ N = 0 := by
    rw [neg_pow, hN, mul_zero]
  constructor
  · simpa [finiteGeometricInverse, hneg] using geom_sum_mul_neg (-a) N
  · simpa [finiteGeometricInverse, hneg] using mul_neg_geom_sum (-a) N
```

The source separately invokes the two ordered geometric sum lemmas. This is why `TwoSidedInverse` has two fields. An inverse from one side does not automatically supply the other in a general noncommutative ring. Once this result is available, the theorem for a diagonal part with transfers transports the candidate through the factorization of $D + H$.

The cutoff N and its nilpotence proof are inputs. The theorem is not a procedure for deciding whether an arbitrary ring element is nilpotent. Its finite sum is explicit, but a concrete matrix application must still identify the matrix ring, supply D^{-1}, and prove the required equations.

8.2 The filtration is deliberately minimal

A multiplicative filtration records additive closure, decreasing levels, and multiplicative addition of indices.

Source: KomorebiVerification/PaperII/WeightedTransfer.lean.

```
structure MultiplicativeFiltration (R : Type u) [Ring R] where
  Mem : ℕ → R → Prop
  zero_mem : ∀ n, Mem n 0
  add_mem : ∀ {n x y}, Mem n x → Mem n y → Mem n (x + y)
  neg_mem : ∀ {n x}, Mem n x → Mem n (-x)
  one_mem : Mem 0 1
  antitone : ∀ {m n x}, m ≤ n → Mem n x → Mem m x
  mul_mem : ∀ {m n x y}, Mem m x → Mem n y → Mem (m + n) (x * y)
```

Write F^n for the predicate `Mem n`. Multiplicativity is $F^m F^n \subseteq F^{m+n}$. Antitonicity sends membership at a larger index to membership at a smaller one. Completeness, separation, and a topology for an

inverse limit are not fields of this structure. Thus the word "filtration" in its name does not imply any convergence statement.

A precision certificate for a ring homomorphism $q : R \to S$ says $q(F^L) = 0$. It does not claim that S is a particular quotient or that $\ker q = F^L$. Killing the level is sufficient for the theorem. The proof below first puts a word into its cumulative filtration level and then uses antitonicity to reach F^L.

Source: `KomorebiVerification/PaperII/WeightedTransfer.lean`.

```
structure PrecisionCertificate {R : Type u} {S : Type v} [Ring R] [Ring S]
    (filtration : MultiplicativeFiltration R) (q : R →+* S) (L : ℕ) : Prop where
  map_eq_zero : ∀ {x : R}, filtration.Mem L x → q x = 0

/-- A transfer word whose cumulative weight reaches `L` vanishes in every realization carrying a
precision-`L` certificate. -/
theorem wordProduct_vanishes_at_precision {R : Type u} {S : Type v}
    [Ring R] [Ring S] (filtration : MultiplicativeFiltration R)
    (q : R →+* S) (L : ℕ) (certificate : PrecisionCertificate filtration q L)
    {E : Type w} (weight : E → ℕ) (edge : E → R)
    (hedge : ∀ e, filtration.Mem (weight e) (edge e))
    (word : List E) (hweight : L ≤ wordWeight weight word) :
    q (wordProduct edge word) = 0 := by
  apply certificate.map_eq_zero
  apply filtration.antitone hweight
  exact filtration.wordProduct_mem weight edge hedge word
```

The caller supplies the facts about edge membership $A_e \in F^{w(e)}$. Induction on a list gives $A_{e_1} \cdots A_{e_r} \in F^{w(e_1)+\cdots+w(e_r)}$. All words of weight at least L therefore disappear after applying q.

8.3 Written products and chronological paths

The definition `wordProduct edge word` is `(word.map edge).prod`. For the list $[e_1, \dots, e_r]$, it gives the written product $A_{e_1} \cdots A_{e_r}$. When these are operators, the rightmost factor acts first. This is the convention in the canonical formal branch for weighted words.

The expanded traditional discussion of block paths writes a chronologically traversed path $e_1, \dots, e_r$ as $A_{e_r} \cdots A_{e_1}$. To translate an indexed term, reverse the list. The full sums over all words of a fixed length agree under this bijection. The sums truncated by weight also agree because reversal preserves cumulative weight. Individual indexed products need not agree. An equation for one labelled path must therefore retain its order convention.

The generic formal sum over weighted words ranges over edge lists without imposing graph adjacency. In a concrete block realization, incompatible products may vanish for structural reasons, but those incidence facts are not fields of `MultiplicativeFiltration`. They must be supplied when interpreting the sum as a sum over directed block paths.

8.4 What is actually retained

The pruned expression contains only words with total weight strictly below L. Its length cutoff N is a separate parameter.

Source: KomorebiVerification/PaperII/WeightedTransfer.lean.

```
def retainedFixedWordSum {E : Type u} {R : Type v} [Fintype E] [Ring R]
    (weight : E → ℕ) (edge : E → R) (L n : ℕ) : R :=
  ∑ word : Fin n → E with wordWeight weight (List.ofFn word) < L,
    wordProduct edge (List.ofFn word)

/-- The explicit signed, weight-pruned word candidate through word length `N - 1`.

Unlike `finiteGeometricInverse`, this expression visibly contains only retained transfer words:
every summand has cumulative weight strictly below `L`.  The independent length cutoff `N` permits
the sharper cutoff obtained from a lower bound on all edge weights. -/
def prunedWordInverse {E : Type u} {R : Type v} [Fintype E] [Ring R]
    (weight : E → ℕ) (edge : E → R) (L N : ℕ) : R :=
  ∑ n ∈ Finset.range N, (-1 : R) ^ n * retainedFixedWordSum weight edge L n
```

If each edge weight is at least m, a word of length N has weight at least mN. Hence $L \leq mN$ makes the Nth mapped power vanish. For $m > 0$, the proved ceiling inequality allows $N = \lceil L/m \rceil$. With merely positive integral weights, $N = L$ is a valid coarse cutoff.

The equality between the mapped full geometric sum and the mapped sum of retained words is proved by splitting the finite word sums into retained and killed terms. The inverse conclusion then follows from mapped nilpotence. This order of reasoning matters: pruning is justified by exact zero in the precision realization, not by a small numerical residual.

The full theorem for a diagonal part with transfers exposes all remaining data. It accepts the inverse of D, the normalized edge expansion, the precision certificate, the lower bound on the weights, the cutoff inequality, and the proofs about edge membership.

Source: KomorebiVerification/PaperII/WeightedTransfer.lean.

```
theorem finitePrecisionPrunedDiagonalTransferInverse_of_minWeight
    {R : Type u} {S : Type v} [Ring R] [Ring S]
    (filtration : MultiplicativeFiltration R)
    (q : R →+* S) (L m N : ℕ) (certificate : PrecisionCertificate filtration q L)
    (D Dinv K : R) (hD : TwoSidedInverse D Dinv)
    {E : Type w} [Fintype E] (weight : E → ℕ) (edge : E → R)
    (htransfer : Dinv * K = ∑ e : E, edge e)
    (hmin : ∀ e, m ≤ weight e) (hcutoff : L ≤ m * N)
    (hedge : ∀ e, filtration.Mem (weight e) (edge e)) :
    TwoSidedInverse
      (q (D + K))
      (q (prunedDiagonalTransferInverse weight edge Dinv L N)) := by
  have hpruned := prunedWordInverse_spec_of_minWeight
    filtration q L m N certificate weight edge hmin hcutoff hedge
  have htransferMapped : q Dinv * q K = ∑ e : E, q (edge e) := by
    simpa only [map_mul, map_sum] using congrArg q htransfer
  have hnormalized :
      TwoSidedInverse (1 + q Dinv * q K) (q (prunedWordInverse weight edge L N)) := by
    simpa only [map_add, map_one, map_sum, htransferMapped] using hpruned
  have hdiagonal := diagonal_inverse_of_perturbation_inverse
    (q D) (q Dinv) (q K) (q (prunedWordInverse weight edge L N)) (hD.map q) hnormalized
  simpa only [prunedDiagonalTransferInverse, map_add, map_mul] using hdiagonal
```

The proof first obtains the pruned inverse of $1 + \sum_e A_e$. It maps the supplied equality for normalized transfers, rewrites the inverse target, and applies the diagonal factorization lemma. The final `simpa` only unfolds the candidate and the homomorphism's preservation of addition and multiplication. No inverse of D or quotient construction is hidden in that final simplification.

8.5 The analytic branch and the algorithm contract

In a complete normed ring, $\|D^{-1}H\| < 1$ supplies a separate inverse given by a Neumann series. The source defines the analytic candidate noncomputably and proves both inverse laws using the normed summability theory. This is neither the finite nilpotence premise nor completeness of an arbitrary filtered ring. An application using a valuation needs a bridge to one of the formal settings.

Row 24 packages the bounded geometric/pruned expression as a mathematical algorithm specification. It proves that the chosen ceiling cutoff meets the required inequality and that the mapped result is an inverse on both sides. It does not construct a modular inverse routine in a chosen basis, decide symbolic zero, certify the Python implementation, or establish a runtime bound.

9 Transfer zeta: from independent walks to rationality

Rows 12–13 and 25 are among the component's strongest construction results. The input is a finite directed multigraph with positive natural edge weights. Parallel edges are distinct edge values and therefore contribute their multiplicities automatically. Repeated vertices and edges are allowed: the objects counted are walks.

Source: `KomorebiVerification/PaperII/TransferZeta.lean`.

```
structure PositiveWeightedMultigraph (V : Type u) (E : Type v) where
  source : E → V
  target : E → V
  weight : E → ℕ
  weight_pos : ∀ e, 0 < weight e

namespace PositiveWeightedMultigraph

variable {V : Type u} {E : Type v}

/-- The recursive endpoint condition for an ordered list of edges. -/
def IsWalk (G : PositiveWeightedMultigraph V E) (s t : V) : List E → Prop
  | [] => s = t
  | e :: es => G.source e = s ∧ G.IsWalk (G.target e) t es
```

The recursion `IsWalk` says an empty edge list starts and ends at the same vertex. A nonempty list starts with an edge whose source is the current vertex, then continues from that edge's target. The total weight is the sum of edge weights.

9.1 Why a coefficient has a finite enumeration

Positivity implies that a walk of total weight n has length at most n. The enumerator can therefore range over lengths $0, \dots, n$ and all edge tuples of each length, then filter by endpoint validity and exact total weight.

Source: `KomorebiVerification/PaperII/TransferZeta.lean`.

```
/--
The exact finite enumeration used for the coefficient of weight `n`.
-/
def walkTuplesAtWeight [Fintype E] [DecidableEq E] [DecidableEq V]
    (G : PositiveWeightedMultigraph V E) (s t : V) (n : ℕ) :
    Finset (BoundedEdgeTuple E n) :=
  Finset.univ.filter fun x ↦
    G.IsWalk s t (tupleEdges x) ∧ G.pathWeight (tupleEdges x) = n

/-- Membership in the finite enumerator has exactly the advertised meaning. -/
theorem mem_walkTuplesAtWeight_iff [Fintype E] [DecidableEq E] [DecidableEq V]
    (G : PositiveWeightedMultigraph V E) (s t : V) (n : ℕ)
    (x : BoundedEdgeTuple E n) :
    x ∈ G.walkTuplesAtWeight s t n ↔
      G.IsWalk s t (tupleEdges x) ∧ G.pathWeight (tupleEdges x) = n := by
  simp [walkTuplesAtWeight]
```

The module also defines an independent semantic type of finite edge lists having the requested endpoints and weight, without building the enumeration bound into that definition. The capture theorem uses positivity to connect the unrestricted semantic representation to the bounded tuple enumeration. The counting theorem and the trace wrapper identify the returned natural number with the cardinality of this semantic walk type.

This avoids defining correctness merely as equality with the same algorithm under another name. The finite enumeration and the semantic list predicate are separate constructions connected by equivalences. The implementation remains a potentially large exhaustive enumeration. No efficiency theorem is claimed.

9.2 The recurrence from the first edge

Fix a target t and let $a_{st}(n)$ count walks from s to t of weight n. The empty walk contributes 1 exactly when $s = t$ and $n = 0$. Every other walk has a unique first edge. Removing it gives

$$a_{st}(n) = \mathbf{1}_{s=t,\, n=0} + \sum_{\substack{e:\, s \to u \\ w(e) \le n}} a_{ut}(n - w(e)).$$

The source constructs the corresponding equivalence from the first edge and obtains the recurrence by cardinality. Positivity ensures the recursive weight decreases. The bound $w(e) \le n$ is explicit, avoiding an incorrect use of truncated natural subtraction when the first edge is too heavy.

Define $Z_{st}(X) = \sum_{n \ge 0} a_{st}(n) X^n$. This is a formal power series over $\mathbb{Z}$. The weighted adjacency entry $A_{su}(X)$ sums $X^{w(e)}$ over all edges $s \to u$. The recurrence from the first edge becomes

$$(I - A(X))Z(X) = I.$$

The row/column orientation here follows traversal from the first edge. When comparing it with operator blocks with the output index first, read the definitions rather than transferring an index convention by analogy.

9.3 A second construction: determinant and adjugate

Finite vertex data define the polynomial matrix $B(X) = I - A(X)$ and

$$Q(X) = \det B(X), \qquad P_{st}(X) = \operatorname{adj}(B(X))_{st}.$$

Positive edge weights give $A(0) = 0$, hence $B(0) = I$ and $Q(0) = 1$. Thus Q is a unit in $\mathbb{Z}[\![X]\!]$. The adjugate identity constructs a matrix Z^{mat} that is an inverse on both sides.

The crucial proof is the identification of the independently enumerated series matrix with this rational matrix. It uses associativity and the two inverse equations.

Source: `KomorebiVerification/PaperII/TransferZeta.lean`.

```
theorem enumeratedPathZetaMatrix_eq_matrixPathZetaMatrix :
    G.enumeratedPathZetaMatrix = G.matrixPathZetaMatrix := by
  calc
    G.enumeratedPathZetaMatrix = 1 * G.enumeratedPathZetaMatrix :=
      (Matrix.one_mul _).symm
    _ = (G.matrixPathZetaMatrix * G.transferSeriesMatrix) *
        G.enumeratedPathZetaMatrix := by
      rw [G.matrixPathZetaMatrix_mul_transferSeriesMatrix]
    _ = G.matrixPathZetaMatrix *
        (G.transferSeriesMatrix * G.enumeratedPathZetaMatrix) :=
      Matrix.mul_assoc _ _ _
    _ = G.matrixPathZetaMatrix * 1 := by
      rw [G.transferSeriesMatrix_mul_enumeratedPathZetaMatrix]
    _ = G.matrixPathZetaMatrix := Matrix.mul_one _
```

This is the familiar uniqueness calculation $Z = IZ = (Z^{\mathrm{mat}}B)Z = Z^{\mathrm{mat}}(BZ) = Z^{\mathrm{mat}}$. It makes the construction claim auditable: rationality was not an assumption about the enumerated coefficients. The final existential theorem simply packages the already constructed numerator and denominator.

Source: `KomorebiVerification/PaperII/TransferZeta.lean`.

```
theorem finite_graph_pathZeta_rationality (s t : V) :
    ∃ P Q : Polynomial ℤ,
      Q.coeff 0 = 1 ∧ Q ≠ 0 ∧
        (Q : PowerSeries ℤ) * G.pathZeta s t = (P : PowerSeries ℤ) := by
  refine ⟨G.transferNumerator s t, G.transferDenominator,
    G.transferDenominator_coeff_zero, G.transferDenominator_ne_zero, ?_⟩
  exact G.transferDenominator_mul_pathZeta s t
```

The older `RationalCertificate` interface remains available for an externally supplied pair P, Q. Its theorem is explicit about its certificate. The canonical branch for finite graphs does not take that certificate. It derives its own data from the graph. The map preserves both interfaces so users can choose one without confusing their hypothesis burdens.

9.4 The recurrence is extracted from the proved identity

Write $Q(X) = \sum q_j X^j$. Taking coefficient n in $QZ = P$ gives an antidiagonal convolution. Once $n > \deg P$, the coefficient on the right is zero.

Source: KomorebiVerification/PaperII/TransferZeta.lean.

```
theorem finite_graph_constant_coefficient_recurrence (s t : V)
    {n : ℕ} (hn : (G.transferNumerator s t).natDegree < n) :
    ∑ p ∈ Finset.HasAntidiagonal.antidiagonal n,
        G.transferDenominator.coeff p.1 *
          (G.coefficientAlgorithm s t p.2 : ℤ) = 0 := by
  have hcoeff := congrArg (PowerSeries.coeff n)
    (G.transferDenominator_mul_pathZeta s t)
  rw [PowerSeries.coeff_mul] at hcoeff
  simpa [Polynomial.coeff_eq_zero_of_natDegree_lt hn] using hcoeff
```

Because $q_0 = 1$, this can be read as an eventual recurrence with constant coefficients for $a_{st}(n)$. The source's antidiagonal form states the identity without needing to rearrange its leading term. It does not guess a recurrence from sampled coefficients.

The determinant denominator may have canceling factors for a particular matrix entry, so this construction does not prove minimality of Q or of the recurrence order. The determinant and adjugate definitions are noncomputable here. The result concerns formal power series and supplies no disk of analytic convergence, extracted symbolic determinant program, or rationality theorem for infinite graphs.

10 Evaluated separation and lattice precision

Rows 14–16 start with endomorphisms F, G, U, V of a vector space over a field K. The central premise is an evaluated Bézout equality

$$UF + VG = rI.$$

The source does not construct polynomials whose evaluation gives these maps, compute a general resultant, factor a characteristic polynomial, or instantiate Cayley–Hamilton. These operations may justify the premise in a mathematical application, but they remain upstream of this formal interface.

Let $N = VG$. The identity without denominators is $N^2 = rN$. After adding $r \neq 0$, the normalized map $E = r^{-1}N$ is idempotent. The bundled `EvaluatedBezoutCertificate` contains six pairwise commutations as well as nonvanishing, Bézout, and annihilation. The mapped minimal theorems need fewer fields and state this directly.

10.1 A proof walkthrough: the square identity without division

Only $FG = GF$, $GU = UG$, and $FG = 0$ are needed to kill the cross term $VGUF$. Using $VG = rI - UF$ on one occurrence of the numerator then proves the square identity.

Source: KomorebiVerification/PaperII/ResultantSeparation.lean.

```
theorem separatorNumerator_mul_self_of_bezout
    (bezout : U * F + V * G = r • (1 : End (K := K) (M := M)))
    (commute_FG : Commute F G) (commute_GU : Commute G U) (annihilate : F * G = 0) :
    separatorNumerator V G * separatorNumerator V G = r • separatorNumerator V G := by
  calc
    separatorNumerator V G * separatorNumerator V G =
        separatorNumerator V G *
```

```
        (r • (1 : End (K := K) (M := M)) - U * F) := by
          exact congrArg (separatorNumerator V G * ·)
            (separatorNumerator_eq_scalar_sub_of_bezout bezout)
    _ = separatorNumerator V G * (r • (1 : End (K := K) (M := M))) -
        separatorNumerator V G * (U * F) := by rw [mul_sub]
    _ = r • separatorNumerator V G := by
      rw [separatorNumerator_mul_complement_eq_zero_of_commute
        commute_FG commute_GU annihilate]
      simp only [sub_zero, mul_smul_comm, mul_one]
```

The proof rewrites just the second factor, distributes multiplication over subtraction, and invokes vanishing of the cross term. It needs neither $r \neq 0$ nor four unused commutations from the stronger bundle. Consequently the theorem without denominators remains meaningful even when normalization is unavailable. The later normalized theorem uses scalar associativity and field simplification with the explicit nonzero hypothesis.

On $\ker F$, the evaluated Bézout equation gives $Nx = rx$ directly. On $\ker G$, $Nx = V(Gx) = 0$. After normalization, E acts as identity and zero respectively. A statement that F is nilpotent on some larger subspace would not imply $Fx = 0$ there. The formal theorem uses exact kernel membership. This prevents an intuition about primary summands from silently strengthening the proved conclusion.

10.2 What the formal lattice means

The lattice parameter is an `AddSubgroup M`. Preservation by a map means that each member is sent back into this additive subgroup. A general valuation ring, finite generation, full rank, completeness, and a chosen basis are not built into its type.

A clearing certificate at exponent s states that $\pi^s E$ preserves the lattice. The source names this `Is SeparationWeight`. If multiplication by π also preserves the lattice, the set of clearing exponents is upward closed. Given existence of some clearing exponent, `Nat.find` defines a least one and proves its minimality property. This is a noncomputable construction of a least witness, not a search algorithm over matrix entries.

If $r = u\pi^s$, then

$$\pi^s E = u^{-1} V G.$$

To turn this algebraic identity into lattice preservation, one must know that u^{-1} preserves the chosen lattice, together with V and G. An abstract field unit need not preserve an arbitrary additive subgroup. The formal certificate for the unit action on the lattice makes this assumption visible. The resulting exponent bounds the least separation weight from above. It need not be the minimal exponent.

The expanded traditional theory identifies exact weights through a lattice gluing module and gives necessary as well as sufficient delay statements. Those ordinary written proofs are beyond these mapped consequences of clearing certificates. They should not be attributed to `separationWeight` merely because the names describe related mathematics.

10.3 Precision delay as transport of a witness

Formal congruence at depth n means an explicit equation $x - y = \pi^n z$ with z in the lattice. If $\pi^s E$ preserves the lattice, input congruence at $m + s$ becomes output congruence at m.

Source: KomorebiVerification/PaperII/ResultantSeparation.lean.

```
/-- Congruence modulo the scalar lattice `pi^n lattice`, expressed with an explicit witness. -/
def CongruentModLattice (pi : K) (lattice : AddSubgroup M) (n : ℕ) (x y : M) : Prop :=
  ∃ z : M, z ∈ lattice ∧ x - y = (pi ^ n) • z

/-- An `s`-digit denominator certificate turns `m+s` input digits into `m` output digits. -/
theorem projector_precision_delay {pi : K} {E : End (K := K) (M := M)}
    {lattice : AddSubgroup M} {s m : ℕ}
    (hweight : IsSeparationWeight pi E lattice s) {x y : M}
    (hxy : CongruentModLattice pi lattice (m + s) x y) :
    CongruentModLattice pi lattice m (E x) (E y) := by
  rcases hxy with ⟨z, hz, hxy⟩
  refine ⟨((pi ^ s) • E) z, hweight z hz, ?_⟩
  rw [← map_sub, hxy, map_smul]
  simp only [LinearMap.smul_apply, pow_add, smul_smul]
```

The proof opens the original witness z. It chooses the new witness $z' = (\pi^s E)z$, whose lattice membership is exactly the clearing hypothesis. Linearity and $\pi^{m+s} = \pi^m \pi^s$ establish the new difference equation. No digit expansion, topology, or valuation API enters this proof. It is sufficient delay from a supplied certificate, without a claim that fewer input digits must fail.

11 Concrete Smith fibres and the dyadic sign model

Row 18 proves a concrete arithmetic statement over `ZMod (p^depth)` for $p > 1$. The hypothesis is not that p is prime. Since the modulus and coefficient are powers of the same integer, the needed gcd and divisibility identities hold at this stated scope.

11.1 The solution fibre is an actual type of solutions

For a weight s and depth L, multiplication by p^s is an additive endomorphism of $\mathbb{Z}/p^L\mathbb{Z}$. Its fibre over b is the subtype of elements that map to b.

Source: KomorebiVerification/PaperII/SmithDyadic.lean.

```
def smithZModMulHom (p weight depth : ℕ) :
    ZMod (p ^ depth) →+ ZMod (p ^ depth) :=
  nsmulAddMonoidHom (p ^ weight)

/-- The actual solution fibre of the one-coordinate Smith congruence
`p ^ weight * x = target` modulo `p ^ depth`. -/
abbrev SmithZModFiber (p weight depth : ℕ) (target : ZMod (p ^ depth)) :=
  ↥(smithZModMulHom p weight depth ⁻¹' {target})

/-- The concrete fibre is finite, inherited from the finite quotient `ZMod (p ^ depth)`. -/
noncomputable instance smithZModFiberFintype (p weight depth : ℕ) [NeZero (p ^ depth)]
    (target : ZMod (p ^ depth)) : Fintype (SmithZModFiber p weight depth target) :=
  Fintype.ofFinite _

/-- Concrete divisibility admissibility for a one-coordinate Smith congruence. -/
def SmithZModAdmissible (p weight depth : ℕ) (target : ZMod (p ^ depth)) : Prop :=
  p ^ min weight depth | target.val
```

The criterion is

$$p^{\min(s,L)} \mid \mathrm{val}(b),$$

where `target.val` is the canonical natural representative. The source proves that this is equivalent to nonemptiness of the actual fibre. If $s \leq L$, an admissibility witness supplies a quotient of the target representative and hence a solution. If $L \leq s$, admissibility forces the target to be zero. The zero residue supplies a solution.

The forward direction uses the modular representative of a supplied solution and divisibility by both the coefficient and modulus. Thus solvability is proved from concrete modular arithmetic, rather than inserted as a certificate field.

11.2 A proof walkthrough: counting an admissible fibre

Once a solution x_0 exists, translation by x_0 identifies the fibre with the kernel of multiplication by p^s. The kernel cardinality in this finite cyclic additive group is the gcd of its order with p^s.

Source: `KomorebiVerification/PaperII/SmithDyadic.lean`.

```
theorem smithZModFiber_card_of_admissible {p weight depth : ℕ} [NeZero (p ^ depth)] (hp : 1 < p)
    (target : ZMod (p ^ depth))
    (hadmissible : SmithZModAdmissible p weight depth target) :
    Fintype.card (SmithZModFiber p weight depth target) = p ^ min weight depth := by
  obtain ⟨x, hx⟩ := (smithZModFiber_nonempty_iff hp target).mpr hadmissible
  let f : ZMod (p ^ depth) →+ ZMod (p ^ depth) := smithZModMulHom p weight depth
  have hxEq : f x = target := Set.mem_singleton_iff.mp hx
  have e : SmithZModFiber p weight depth target ≃ f.ker := by
    simpa only [SmithZModFiber, f, hxEq] using f.fiberEquivKer x
  calc
    Fintype.card (SmithZModFiber p weight depth target) =
        Nat.card (SmithZModFiber p weight depth target) := Nat.card_eq_fintype_card.symm
    _ = Nat.card f.ker := Nat.card_congr e
    _ = (Nat.card (ZMod (p ^ depth))).gcd (p ^ weight) := by
      simpa [f, smithZModMulHom] using
        IsAddCyclic.card_nsmulAddMonoidHom_ker (ZMod (p ^ depth)) (p ^ weight)
    _ = (p ^ depth).gcd (p ^ weight) := by rw [Nat.card_zmod]
    _ = p ^ min weight depth := by
      rcases Nat.le_total weight depth with hwd | hdw
      · rw [Nat.min_eq_left hwd, Nat.gcd_eq_right_iff_dvd.mpr (Nat.pow_dvd_pow p hwd)]
      · rw [Nat.min_eq_right hdw, Nat.gcd_eq_left_iff_dvd.mpr (Nat.pow_dvd_pow p hdw)]
```

The final two cases use the divisibility of one power of p by the other to obtain

$$\#\{x : p^s x = b \pmod{p^L}\} = p^{\min(s,L)}.$$

This is an exact cardinality of a finite type. It does not enumerate the solutions or provide a particular coordinate normal form for every one of them.

For a finite diagonal system, the global fibre is a dependent function type: one solution in each coordinate. Products of finite cardinalities give the count, while one inadmissible coordinate makes the product type empty.

Source: KomorebiVerification/PaperII/SmithDyadic.lean.

```
theorem smithZMod_global_conditional_count {p depth : ℕ} [NeZero (p ^ depth)] (hp : 1 < p)
    (weight : κ → ℕ) (target : κ → ZMod (p ^ depth)) :
    Fintype.card (GlobalFiber (fun i => SmithZModFiber p (weight i) depth (target i))) =
      if ∀ i, SmithZModAdmissible p (weight i) depth (target i)
      then p ^ ∑ i, min (weight i) depth else 0 := by
  classical
  exact smith_conditional_product_count
    (fiber := fun i => SmithZModFiber p (weight i) depth (target i))
    (admissible := fun i => SmithZModAdmissible p (weight i) depth (target i))
    p depth weight (fun i => smithZModCoordinateCertificate hp (target i))
```

If all coordinates are admissible, the cardinality is $p^{\sum_i \min(s_i, L)}$. Otherwise it is zero. At depth zero, the quotient has one element and the exponent is zero, consistent with the stated formula. A literal zero diagonal coefficient can be represented at precision L by any weight at least L, since p^s then vanishes in the quotient.

11.3 The abstract extension keeps its certificate boundary

The same module has separate `SmithSolvabilityCertificate` and `SmithCardinalityCertificate` structures. For arbitrary finite coordinate fibres, these supply the equivalence and count for each coordinate, with an externally chosen base q. The global product theorems then assemble them.

In a general discrete valuation ring with residue field of cardinality q, the familiar coordinate count has the analogous form. The formal component does not construct that DVR arithmetic or derive its cardinality theorem. Likewise it does not calculate Smith normal form of an input matrix, find matrices for a change of basis, or extract diagonal weights. The automatic result for row 18 begins with a diagonal system over the concrete modular quotient. The abstract extension begins with certificates.

11.4 Rational sign projectors and local denominator weight

Row 17 fixes the rational matrices

$$J = \begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix}, \qquad P_+ = \begin{pmatrix} 1 & 1/2 \\ 0 & 0 \end{pmatrix}, \qquad P_- = \begin{pmatrix} 0 & -1/2 \\ 0 & 1 \end{pmatrix}.$$

Entrywise proofs establish $J^2 = I$, projector idempotence, mutual annihilation, and $P_+ + P_- = I$. The local integrality predicate says that a rational entry's reduced denominator is not divisible by p. Each projector has exact local denominator weight zero at every prime other than two and exact weight one at two.

The local word matters. Global integrality demands integer entries. Integrality at p permits denominators prime to p. A global denominator weight and a local denominator weight are defined separately in the source. The resultant of the sign factors is explicitly computed as $\mathrm{Res}(X-1, X+1) = 2$, and its dyadic valuation is matched to the exact local weights of these matrices.

This is a model with rational entries using valuations with natural number values. It does not construct $\mathbb{Q}_p$ or its valuation ring, identify arbitrary completed lattices, or prove a general equivalence between all rational integrality criteria and lattice preservation. Reduction modulo two applies to the characteristic factors. The rational matrices containing $1/2$ do not have an entrywise reduction modulo two.

11.5 The Smith phase identity has only algebraic scope

Row 22 checks the fractional expression `smithPhase`, obtained by pulling back the action, at phase -1. Over a field with $2 \neq 0$ and $z \neq 0$, its value is z^{-1}. The extra manuscript condition $z \neq -1$ is unnecessary for this algebraic identity and is removed by the formal theorem.

The result does not formalize the complex Smith disk, an interpretation on a Möbius domain, radius preservation, or physical impedance. In the traditional companion, see the subsection on Smith chart coordinates and its unnumbered phase identity. The subsection counter is not an equation number.

12 Typed defects: domain and codomain stay visible

Rows 19–21 keep the domain and codomain explicit. Let X_f and X_c be fine and coarse modules, $C : X_f \to X_c$, and let T_f and T_c act on their respective spaces. The defect for one step is

$$\Delta = CT_f - T_cC : X_f \longrightarrow X_c.$$

The source does not assume that the two spaces coincide.

Source: `KomorebiVerification/PaperII/TypedLinearDefects.lean`.

```
variable {R : Type u} [Semiring R]
variable {Xf : Type v} {Xc : Type w}
variable [AddCommGroup Xf] [AddCommGroup Xc]
variable [Module R Xf] [Module R Xc]

/-! ## Typed power defects -/

/-- The typed one-step defect `C ∘ Tf - Tc ∘ C`. -/
def defect (C : Xf →ₗ[R] Xc)
    (Tf : Module.End R Xf) (Tc : Module.End R Xc) : Xf →ₗ[R] Xc :=
  C.comp Tf - Tc.comp C
```

Notice that the scalar type is a semiring while the spaces have additive group structures, allowing subtraction of maps. Scalar subtraction is not needed for the basic power identity. This is one example of why the complete type is more informative than a loose description such as “linear algebra over a ring.”

12.1 The telescoping mechanism

Define $E_n = CT_f^n - T_c^nC$. The induction uses

$$E_{n+1} = T_cE_n + \Delta T_f^n.$$

Expanding E_n gives

$$CT_f^n - T_c^nC = \sum_{k=0}^{n-1} T_c^{n-1-k}\Delta T_f^k.$$

Every summand is a map $X_f \to X_c$: the right power acts in X_f, the defect crosses to X_c, and the left power acts there. Treating C as an endomorphism would hide this type constraint.

The theorem for all powers includes $n = 0$. The left side is $C - C = 0$ and the finite sum is empty. The theorem for positive powers is a wrapper around this stronger statement. In the successor proof, `Finset.sum_range_succ` separates the new last term. Arithmetic of natural numbers verifies the shifted exponents. The exponent $n - 1 - k$ occurs only at $k < n$, where it has the intended value.

If the defect for one step is zero, every term in the sum vanishes.

Source: `KomorebiVerification/PaperII/TypedLinearDefects.lean`.

```
theorem powers_compatible_of_defect_eq_zero
    (C : Xf →ₗ[R] Xc) (Tf : Module.End R Xf) (Tc : Module.End R Xc)
    (hdefect : defect C Tf Tc = 0) (n : ℕ) :
    C.comp (Tf ^ n) = (Tc ^ n).comp C := by
  apply sub_eq_zero.mp
  rw [power_defect_identity_all]
  simp [hdefect]
```

This extends to finite polynomial functional calculus by summing the corresponding power identities with scalar coefficients. The definition of polynomial action explicitly carries the commutative scalar assumptions needed for those coefficients to act centrally. The result does not establish holomorphic, exponential, or other infinite functional calculus without further analytic work.

12.2 The operator norm estimate

In the branch for continuous linear maps, the same finite identity yields

$$\|CT_f^n - T_c^n C\| \le \|\Delta\| \sum_{k=0}^{n-1} \|T_c\|^{n-1-k} \|T_f\|^k.$$

The triangle inequality bounds the norm of the sum. Submultiplicativity bounds each composition, and the endomorphism power estimate bounds each power. Completeness of the spaces is unnecessary because this is a finite sum.

If both operator norms are at most ρ, the component derives the following majorant

$$\|CT_f^n - T_c^n C\| \le n\rho^{n-1}\|\Delta\|.$$

The source states the endpoint for natural exponents carefully, so the theorem for all n also includes zero. The hypotheses supply ρ and the two norm bounds. A bound on the spectral radius is not interchangeable with an operator norm bound here, and the majorant is not asserted to be sharp.

For example, exact intertwining $\Delta = 0$ forces every defect for finite powers to vanish. A small nonzero defect may still be amplified by large powers. The theorem gives an upper bound with the stated norm data. It does not identify the actual error for a particular trajectory or nonlinear dynamics.

12.3 Resolvents from the inverse equations actually used

Let R_f and R_c be supplied inverse candidates for the scalar shifts. The continuous theorem needs only

$$(zI - T_f)R_f = I, \qquad R_c(zI - T_c) = I.$$

The fine witness is a right inverse. The coarse witness is a left inverse. They are sufficient for the ordered sandwich calculation

$$CR_f - R_cC = R_c\Delta R_f.$$

Source: KomorebiVerification/PaperII/TypedLinearDefects.lean.

```
theorem continuous_scalarShift_resolvent_identity
    (z : 𝕜) (C : Ef →L[𝕜] Ec)
    (Tf Rf : Ef →L[𝕜] Ef) (Tc Rc : Ec →L[𝕜] Ec)
    (hf : (continuousScalarShift z Tf).comp Rf = 1)
    (hc : Rc.comp (continuousScalarShift z Tc) = 1) :
    C.comp Rf - Rc.comp C =
      (Rc.comp (continuousDefect C Tf Tc)).comp Rf := by
  calc
    C.comp Rf - Rc.comp C =
        (Rc.comp
          ((continuousScalarShift z Tc).comp C -
            C.comp (continuousScalarShift z Tf))).comp Rf :=
      continuous_inverse_sandwich_identity
        (continuousScalarShift z Tc) Rc
        (continuousScalarShift z Tf) Rf C hc hf
    _ = (Rc.comp (continuousDefect C Tf Tc)).comp Rf := by
      rw [continuousScalarShift_comp_sub_comp_continuousScalarShift]
```

The source first applies a general sandwich identity for inverse candidates. The difference of shifted compositions simplifies to $CT_f - T_cC$ because the two scalar terms cancel. The sign is therefore positive on the right for this convention of Δ. Reversing the defect definition would change that sign.

Applying the operator norm gives

$$\|CR_f - R_cC\| \le \|R_c\|\,\|\Delta\|\,\|R_f\|.$$

No theorem here constructs the inverse candidates or proves that z lies in a resolvent set. The algebraic typed branch uses explicit structures certifying inverse equations on both sides. The continuous branch needs only the two inverse equations displayed above. A user should choose the theorem matching the available data rather than infer inverse existence from the word "resolvent."

12.4 What remains in the written companion

The final traditional development also proves holomorphic functional calculus, exponential, and multilevel defect identities with their own analytic or compositional hypotheses. The preserved Lean map records finite polynomial compatibility, the power estimates, and the resolvent identities with inverse witnesses. Those additional results are outside this inherited map. The current extension status, including its remaining analytic bridge, is recorded in Section 1.

13 Reproducing the historical formal project

Reproduction has two separate layers. The lightweight guide checks confirm that the supplied source matches its checksum manifest, that all mapped names can be found lexically, and that the printed excerpts are exact slices of those files. The Lean replay builds the project, resolves names in the compiled environment, and traverses axiom dependencies. Only the latter executes Lean.

13.1 Start with the preserved pins

Work in a writable copy of `source/canonical_v1.2.1/formalization/lean/`. Keep `lean-toolchain`, `lakefile.toml`, and `lake-manifest.json` together. The exact toolchain is `leanprover/lean4:v4.34.0-rc2`. The Mathlib revision is `aeb72f1e9981b6d35b426b16dca7759be3e7ba06`. Appendix B records every dependency revision in the delivered manifest.

A working Lean/Elan installation, network access for dependencies or a suitable local cache, and sufficient disk space are prerequisites. The following commands are the preserved report's replay procedure, not commands executed while producing this guide.

```
cd source/canonical_v1.2.1/formalization/lean
lake exe cache get
lake build
python3 scripts/verify_theorem_map.py
lake env lean -E warning KomorebiVerification/PaperII/KernelAudit.lean
```

The cache command obtains compiled dependency artifacts. The build checks the aggregate library against the pinned environment. The Python map checker parses exactly one versioned theorem map, expands its namespace prefixes, requires rows 1 through 25, and writes a temporary Lean command that checks every mapped name in the compiled environment. The script removes that temporary file afterward. Its environment check is stronger than this guide's lexical source lookup.

The final command separately executes the audit that rejects violations. Warning escalation is part of the documented invocation. Preserve the exit status and full output of each command. A report written after the fact should not substitute for those transcripts in a new replay.

13.2 The audit gate and its trust boundary

The audit has two parts. `#print axioms` displays dependencies for a representative public sample. A subsequent command traverses every compiled declaration in the imported `Komorebi` namespace and rejects unsafe entries or dependencies outside the allowed basis.

Source: `KomorebiVerification/PaperII/KernelAudit.lean`.

```
run_cmd do
  let env ← getEnv
  let mut audited : Nat := 0
  for (name, info) in env.checked.get.constants.toList do
    if Name.isPrefixOf `Komorebi name then
      audited := audited + 1
      if info.isUnsafe then
        throwError "unsafe project declaration: {name}"
      let axioms ← Lean.collectAxioms name
      for axiomName in axioms do
        unless axiomName == ``propext || axiomName == ``Classical.choice ||
            axiomName == ``Quot.sound do
          throwError "unexpected axiom {axiomName} in project declaration {name}"
  if audited == 0 then
    throwError "kernel audit found no Komorebi declarations"
  logInfo m!"FAIL-CLOSED KERNEL AUDIT PASS: {audited} Komorebi declarations; allowed axioms only:
  ↪ propext, Classical.choice, Quot.sound"
```

The approved foundations are `propext`, `Classical.choice`, and `Quot.sound`. The gate rejects any other axiom dependency, including `sorryAx`, and explicitly fails if no project declaration was inspected. A successful audit is a property of the imported compiled environment at that run. It is not inferred solely from the absence of the word `sorry` in source text.

The sample of printed axioms is not the complete traversal. Conversely, the namespace count includes generated declarations and helper entries, so it is not a count of independent mathematical theorems. The map checker and namespace gate also do not decide whether a theorem's type faithfully represents a prose theorem. That requires scope comparison such as the canonical map and this guide.

13.3 A minimal session for source inspection

After a successful build, create a scratch Lean file beside the project's aggregate import. The following inspection commands ask Lean to display two representative declarations and the axiom dependencies of the automatic rationality result.

```
import KomorebiVerification.PaperII
#check Komorebi.PaperII.NestedRefinement.coordinateRefinementEquiv
#check Komorebi.PaperII.SmithDyadic.smithZMod_global_conditional_count
#print axioms
 ↪ Komorebi.PaperII.TransferZeta.PositiveWeightedMultigraph.finite_graph_pathZeta_rationality
```

Run it with `lake env lean YourInspection.lean`. These are suggested inspection commands, not a new executed transcript. Use `#print` or editor hover to inspect elaborated types, including inferred section parameters and typeclass assumptions. The printed source excerpt alone may omit surrounding binders. The full declaration type is decisive.

When applying a theorem, first fix the intended scalar type and domain/codomain. Then assemble the structural object and prove its fields. For a weighted inverse, this means the filtration, map q, precision certificate, diagonal inverse, and edge expansion. For a finite graph, it means finite vertex/edge types, endpoint maps, and positive weights. For a Smith system, it means a diagonal weight function and modular target. This preparation makes failures informative: a missing instance differs from a missing mathematical witness.

13.4 Common failure modes

Symptom	What to check
Toolchain mismatch	Read `lean-toolchain`; do not silently substitute a stable release for the pinned release candidate.
Missing imported module	Confirm dependencies and cache availability, then build from the Lean project root.
Unknown mapped name	Run the map checker only after a successful aggregate build; verify source and manifest identity.
Unsolved typeclass	Inspect the theorem's complete type and the scalar/module, finite-type, or normed-space assumptions.
Missing inverse proof	Supply the actual equation required by the chosen theorem; naming a candidate does not prove it invertible.
Unexpected axiom	Preserve the failing declaration and transcript; do not weaken the allowed-axiom gate to make the run pass.

Symptom	What to check
Different namespace count	Compare toolchain, dependencies, source hashes, and imports before treating the run as a replay of this release.
Successful Python tests	Interpret them as tests of the Python component; they are not Lean kernel evidence.

A new successful run should record the source hashes, dependency pins, commands, exit codes, output hashes, and environment. If any formal source changes, the old report no longer supplies execution evidence for that changed source, even if the filenames remain the same.

14 Evidence retained and evidence newly checked

The canonical `VERIFICATION_REPORT.md`, dated 3 September 2026, reports successful aggregate build, theorem-map resolution of 160 unique declarations, and a successful scan of 1,034 compiled namespace entries with a gate that rejects violations. It records the approved axiom basis and states that no audited declaration reports `sorryAx`. This guide inherits those statements as a release report.

The inspected canonical component contains no raw transcript of those Lean commands and no log hash attestation for each command. Consequently the historical component alone does not establish a fresh compiler replay, a fresh environment scan, or an identity chain to missing transcripts. Current execution evidence is reported separately in Section 1. The report itself is checksummed, so its preservation is verifiable. Checksumming a report does not independently establish its execution claims.

The new lightweight check establishes the following facts directly from the delivered files:

1. All 22 files under the preserved formal component match their entries in the release `SHA256SUMS.txt` manifest.
2. The same 22 files match the separately preserved baseline copy in this editing workspace.
3. The normative map has exactly rows 1–25 and 160 distinct expanded declaration names, with 171 occurrences across rows.
4. Every mapped name has a lexical declaration location in the preserved source. This is navigation evidence, not compiled name resolution.
5. The 15 Lean source files contain 5,603 physical lines. A scan that accounts for comments and strings finds no active token from the documented placeholder/unsafe source policy list.
6. Every exact excerpt matches its recorded inclusive source slice. Each source and excerpt has a SHA-256 digest.

The scan policy checks active occurrences of `sorry`, `admit`, `sorryAx`, `axiom`, `unsafe`, `native_decide`, and named shortcuts for dynamic declaration. It strips nested block comments, line comments, and strings while retaining line positions. This is a useful static check, not a replacement for Lean elaboration or the environment's transitive axiom inspection.

The historical guide audit records PDF and static source checks, including `lean_compiler_run: false`. Current PDF checks and current Lean replay are separate evidence records. A document build verifies rendering and navigation; formal coverage depends on the current source types and their own compiler evidence.

14.1 A practical rule for reporting success

Report the strongest evidence actually obtained. After running only the data generator, say that static source identity and map reconciliation passed. After a new successful Lean replay, cite that replay's own logs and environment. When quoting the preserved count of 1,034 entries, attribute it to the historical report. When using an expanded traditional theorem, identify its ordinary written proof unless a precise Lean target with matching hypotheses has been supplied.

This discipline makes the component usable: readers can trust an automatic construction where it exists, supply a certificate where it is required, and locate the written mathematics that remains beyond the formal contract.

15 The inherited contract of 25 rows

The following map preserves the canonical v1.1.1 classifications, checked scopes, and explicit boundaries. The targets use the namespace abbreviations below. These abbreviations are only typography: the companion JSON expands every name and records its source file and line. A declaration may occur in several rows. There are 171 occurrences and 160 distinct names. Definitions, structures, equivalences, and theorems all count as declaration targets.

Prefix	Full namespace
PC	Komorebi.Foundation.LinearProjectorFamily
NR	Komorebi.PaperII.NestedRefinement
R	Komorebi.Residue
WT	Komorebi.PaperII.WeightedTransfer
TZ	Komorebi.PaperII.TransferZeta.PositiveWeightedMultigraph
TZC	Komorebi.PaperII.TransferZeta.PositiveWeightedMultigraph.RationalCertificate
RS	Komorebi.PaperII.ResultantSeparation
SD	Komorebi.PaperII.SmithDyadic
LD	Komorebi.PaperII.LinearDefects
TLD	Komorebi.PaperII.TypedLinearDefects

15.1 Row 1: Refinement map

Classification: Exact.

Declaration targets

`NR.children`

`NR.mem_children`

`NR.Refines`

Checked scope. For finite complete projector families over a semiring module, `Refines` records a surjective parent map and equality of every coarse projector with the finite sum over its fine children.

Boundary. No refinement is constructed from arbitrary projector families; no infinite or topological sum is asserted.

15.2 Row 2: Nested refinement theorem

Classification: Stronger.

Declaration targets

`NR.finitePacket_decomposition_unique`

`NR.finitePacket_decomposition_ext`

`NR.Refines.coarse_comp_fine_eq_self`

`NR.Refines.coarse_comp_fine_eq_zero`

`NR.Refines.fine_comp_coarse_eq_self`

`NR.Refines.fine_comp_coarse_eq_zero`

`NR.Refines.coarse_packet_reconstruction`

`NR.Refines.child_decomposition_unique`

`NR.Refines.trans`

Checked scope. Parent/child absorption holds in both composition orders and refinements compose. More generally, any finite sum of vectors lying in mutually orthogonal packet ranges has uniquely determined components, without a refinement or completeness hypothesis.

Boundary. Uniqueness is only asserted on the selected finite set and requires each component to lie in its stated projector range. All index types and sums are finite.

15.3 Row 3: Exact coordinate refinement

Classification: Stronger.

Declaration targets

`NR.refineCoordinates`

`NR.coarsenCoordinates`

`NR.refineCoordinates_self`

`NR.refineCoordinates_comp`

`NR.coordinateRefinementEquiv_trans`

`NR.Refines.refineCoordinates_component`

`NR.Refines.coarsenCoordinates_component`

Checked scope. Analysis after synthesis defines cross-level linear maps for any two finite complete projector families; these maps obey identity and cocycle laws through every intermediate family. A `Refines` witness gives the parent/child component formulas.

Boundary. These are coherent changes of exact coordinates, not truncation or dimensional reduction operators; the cocycle uses exact completeness at each finite level.

15.4 Row 4: Perfect reconstruction across levels

Classification: Stronger.

Declaration targets

`NR.coarsenCoordinates_refineCoordinates`

`NR.refineCoordinates_coarsenCoordinates`

`NR.coordinateRefinementEquiv`

`NR.coordinateOperator`

`NR.coordinateOperator_comp`

`NR.coordinateOperator_naturality`

Checked scope. Cross-level refinement and coarsening are mutual inverses over a semiring module. The typed coordinate representation of every linear map preserves identities and composition and commutes with independent exact coordinate changes on source and target.

Boundary. The inverse laws need completeness of the projector families but no parent map. This is finite algebraic naturality, not convergence for an infinite packet tower.

15.5 Row 5: Transfer-block refinement

Classification: Stronger.

Declaration targets

`NR.transferBlock_refinement_between`

`NR.transferBlock_refinement`

`NR.transferBlock_refinement_and_reconstruction`

Checked scope. Source and target projector families may live on different modules and refine independently; every coarse linear-map block is the double finite sum of its fine descendant blocks. The direct traceability wrapper also records reconstruction of the same map from all coarse blocks and from all fine blocks. The endomorphism formula is a specialization.

Boundary. No sparse extraction algorithm, infinite block matrix, or norm convergence is proved here.

15.6 Row 6: Residue projector algebra

Classification: Exact.

Declaration targets

`R.residueProj_idempotent`

`R.residueProj_orthogonal`

`R.residueProj_complete`

Checked scope. For every positive modulus, coefficient residue selectors on `Nat -> A` are idempotent, mutually annihilating, and complete over an arbitrary semiring and module.

Boundary. This is the division-free coefficient selector, not a character-average formula.

15.7 Row 7: Root-of-unity realization

Classification: Stronger / certificate-explicit.

Declaration targets

`R.ringFourierResidueKernel`

`R.ringFourierKernel_eq_residueKernel`

`R.FiniteRingFourierOrthogonalityCertificate`

`R.FiniteRingFourierOrthogonalityCertificate.toRingCertificate`

`R.ringFourierAverage`

`R.ringFourierAverage_eq_residueProj_of_finite_certificate`

`R.primitiveRoot_fourierOrthogonalityCertificate`

`R.fourierAverage_eq_residueProj_of_primitiveRoot`

Checked scope. Over any commutative semiring, `omega^N = 1` reduces the arbitrary coefficient address to a finite residue address. Consequently an explicit finite `N x N` orthogonality table identifies the division-free Fourier average with the residue selector. Over a field, a positive-order primitive root automatically supplies the normalized inverse-root certificate.

Boundary. The general semiring branch still requires the finite orthogonality table; root-of-unity order alone is not enough in rings with zero divisors. The primitive-root construction is field-only.

15.8 Row 8: Polyphase refinement

Classification: Exact.

Declaration targets

`R.polyphase_refinement`

Checked scope. For positive `N` and `k`, the modulus-`N` selector is the finite sum of exactly those modulus-`N*k` selectors that merge to it, over any semiring module.

Boundary. The theorem concerns exact coefficient masks; it gives no approximation or discarded-packet estimate.

15.9 Row 9: Dyadic-odd packet factorization

Classification: Exact.

Declaration targets

`R.crtEquiv`

`R.crtComponents_bijective`

`R.residueProj_crt`

`R.residueProj_crt_comm`

`R.dyadicOddEquiv`

`R.dyadicOddComponents_bijective`

`R.residueProj_dyadicOdd`

Checked scope. Positive coprime moduli give a bijection of residue labels and the composite selector in both composition orders. For positive odd `M`, this specializes to `2^a * M`.

Boundary. This is coordinate CRT factorization. It does not identify the denominator or lifting behavior of spectral projectors.

15.10 Row 10: Certified weighted inverse

Classification: Stronger / certificate-explicit.

Declaration targets

`WT.TwoSidedInverse`

`WT.finiteGeometricInverse_spec_of_pow_eq_zero`

`WT.diagonalTransferInverse_spec`

`WT.analyticGeometricInverse_spec`

`WT.analyticDiagonalTransferInverse_spec`

Checked scope. In a possibly noncommutative ring, an explicit two-sided inverse for `D` and nilpotence of the normalized perturbation `Dinv*H` give a finite two-sided inverse of `D+H`. In a complete normed ring, `norm (Dinv*H) < 1` gives the separate infinite Neumann inverse.

Boundary. The manuscript perturbation is `H`; the Lean declaration's parameter name `K` has no fraction-field meaning. There is no unrestricted convergence theorem for an abstract complete filtered ring. The analytic candidate is noncomputable.

15.11 Row 11: Finite precision and weighted pruning

Classification: Stronger / certificate-explicit.

Declaration targets

`WT.MultiplicativeFiltration`

`WT.PrecisionCertificate`

`WT.retainedFixedWordSum`

`WT.prunedWordInverse`

`WT.map_finiteGeometricInverse_eq_prunedWordInverse`

`WT.minWeight_mul_length_le_wordWeight`

`WT.map_sum_pow_eq_zero_at_scaled_cutoff`

`WT.prunedWordInverse_spec_of_minWeight`

`WT.finitePrecisionPrunedInverse_of_weightedEdges`

`WT.finitePrecisionPrunedDiagonalTransferInverse_of_minWeight`

`WT.finitePrecisionPrunedDiagonalTransferInverse_of_weightedEdges`

Checked scope. The mapped finite geometric inverse equals an explicit noncommutative word sum retaining exactly the words of cumulative weight below `L`. If every edge weight is at least `m` and `L <= m*N`, the `N`th mapped power vanishes and the pruned word sum is a two-sided inverse; the corresponding diagonal-plus-transfer formula follows from supplied `Dinv` and `Dinv*H = sum edge`.

Boundary. The filtration, precision quotient map, edge-membership data, diagonal inverse, and edge expansion remain inputs. No DVR quotient, quotient map, basis-level modular matrix routine, or extracted pruning implementation is constructed.

15.12 Row 12: Finite-graph transfer-zeta rationality

Classification: Automatic.

Declaration targets

`TZ.coefficientAlgorithm_correct`

`TZ.coefficientAlgorithm_firstEdge`

`TZ.weightedAdjacency`

`TZ.transferMatrix`

`TZ.transferDenominator`

`TZ.transferNumerator`

`TZ.transferDenominator_coeff_zero`

`TZ.pathZeta_eq_matrixPathZeta`

`TZ.transferDenominator_mul_pathZeta`

`TZ.finite_graph_pathZeta_rationality`

`TZ.finite_graph_transferZeta_trace`

`TZC.certificate_explicit_rationality`

Checked scope. For every finite positive-weight directed multigraph and source/target pair, finite walk enumeration satisfies the first-edge recurrence. Lean constructs `Q = det(I-A(X))` and `P` as the corresponding adjugate entry, proves `Q(0)=1`, identifies the independently enumerated `pathZeta` with the matrix inverse entry, and proves `Q*Z=P` without a rationality certificate. A direct wrapper packages the manuscript's enumeration, recurrence, normalization, nonvanishing, and rational identity in one declaration. The older coefficient-certificate route remains available.

Boundary. Walks may repeat vertices and edges; no simple-path restriction is imposed. Parallel edges are distinct edge values. Determinant and adjugate are noncomputable Lean definitions here; no denominator cancellation, minimality, extracted symbolic implementation, or analytic convergence claim is made.

15.13 Row 13: Linear recurrence

Classification: Automatic.

Declaration targets

`TZ.finite_graph_constant_coefficient_recurrence`

`TZC.constant_coefficient_recurrence`

Checked scope. The automatically constructed determinant denominator gives an antidiagonal constant-coefficient recurrence for every `n` beyond the degree of the canonical adjugate numerator. A separately supplied `P,Q` certificate yields the same form through the legacy route.

Boundary. The canonical recurrence is not claimed minimal, and no recurrence is guessed or certified from finitely many sampled coefficients alone.

15.14 Row 14: Evaluated Bezout spectral projector

Classification: Stronger / certificate-explicit.

Declaration targets

`RS.EvaluatedBezoutCertificate`

`RS.separatorNumerator_eq_scalar_sub_of_bezout`

`RS.separatorNumerator_mul_self_of_bezout`

`RS.separator_idempotent_of_bezout`

`RS.separatorNumerator_apply_of_mem_ker_F_of_bezout`

`RS.separator_apply_of_mem_ker_F_of_bezout`

`RS.separator_apply_of_mem_ker_G`

`RS.scalar_mul_separator_of_ne_zero`

`RS.scalar_mul_separator_preservesLattice_of_ne_zero`

Checked scope. Over a field, the evaluated equality `U*F + V*G = r*1`, `F*G=0`, and only the two commutations actually used prove the denominator-free identity `(VG)^2=r(VG)`; adding `r!=0` gives idempotence of `r^-1*VG`. The kernel actions need only the displayed evaluated Bezout or kernel hypotheses, and lattice preservation of the numerator follows when `V,G` preserve the lattice.

Boundary. The evaluated Bezout equality and annihilation are supplied. No polynomial Bezout identity, resultant, characteristic factorization, or Cayley-Hamilton instance is constructed; nilpotence alone is not used to infer exact kernel action.

15.15 Row 15: Separation weight

Classification: Stronger / certificate-explicit.

Declaration targets

`RS.ScalarPreservesLattice`

`RS.IsLatticeUnit`

`RS.IsSeparationWeight`

`RS.IsSeparationWeight.mono`

`RS.separationWeight`

`RS.separationWeight_clears`

`RS.separationWeight_le_of_clears`

`RS.pow_smul_separator_eq_unit_inv_smul_separatorNumerator`

`RS.separator_isSeparationWeight_of_unit_mul_pow_eq_of_inv_preserves`

`RS.separator_hasFiniteSeparationWeight_of_eq_unit_mul_pow`

`RS.separator_separationWeight_le_of_eq_unit_mul_pow`

Checked scope. `IsSeparationWeight pi E lattice s` means that `(pi^s)E` preserves the chosen lattice; weights are upward closed when scalar multiplication by `pi` preserves it. If `r=u*pi^s`, then `pi^s E= u^-1 VG`; preservation by the inverse unit and numerator proves clearing, while a bundled lattice-unit certificate gives existence and the least-weight upper bound `separationWeight<=s`.

Boundary. The unit action on the chosen lattice is explicit data. The least weight is noncomputable via `Nat.find`; no valuation, polynomial resultant construction, search procedure, or claim that the displayed exponent is minimal is supplied.

15.16 Row 16: Projector precision delay

Classification: Exact from certificate.

Declaration targets

`RS.CongruentModLattice`

`RS.projector_precision_delay`

Checked scope. For an arbitrary field scalar `pi`, additive lattice, and clearing-weight certificate, an explicit congruence witness at depth `m+s` maps to one at depth `m`.

Boundary. Congruence is witness-based scalar-lattice membership; no topology, completion, digit representation, or valuation API is required or concluded.

15.17 Row 17: Odd-adic versus dyadic sign behavior

Classification: Stronger / scoped realization.

Declaration targets

`SD.signInvolution_mul_self`

`SD.positiveSignProjector_idempotent`

`SD.negativeSignProjector_idempotent`

`SD.signProjectors_complete`

`SD.HasExactPLocalDenominatorWeight`

`SD.positiveSignProjector_exact_pLocal_weight_zero_of_prime_ne_two`

`SD.negativeSignProjector_exact_pLocal_weight_zero_of_prime_ne_two`

`SD.signFactors_resultant`

`SD.signProjectors_resultant_dyadic_weight_endpoint`

`SD.padicValNat_two_at_odd_prime`

`SD.oddAdicVersusDyadicSignBehavior_trace`

Checked scope. The displayed rational `2 x 2` involution and sign projectors are checked entrywise. Each projector has exact local denominator weight zero at every prime other than `2` and exact `2`-local weight one. Lean also computes `resultant(X-1,X+1)=2` and identifies its `2`-adic valuation with the exact local weight of both projectors. The direct trace wrapper combines both local regimes and the resultant statements.

Boundary. This is an explicit rational-entry, natural-valuation model. Reduction modulo `2` applies to the characteristic factors, not to the rational projector matrices containing `1/2`. Lean does not construct `Q_p`, its valuation ring, or a general equivalence between rational `p`-integrality and preservation of an arbitrary completed lattice.

15.18 Row 18: Concrete ZMod Smith-weighted congruence solve

Classification: Automatic at concrete scope / certificate-explicit extension.

Declaration targets

`SD.smithZModMulHom`

`SD.SmithZModAdmissible`

`SD.smithZModFiber_nonempty_iff`

`SD.smithZModFiber_card_of_admissible`

`SD.smithZModCoordinateCertificate`

`SD.smithZMod_global_solvable_iff`

`SD.smithZMod_global_conditional_count`

`SD.SmithSolvabilityCertificate`

`SD.SmithCardinalityCertificate`

`SD.smith_conditional_product_count`

Checked scope. For `p>1`, Lean proves directly over `ZMod(p^depth)` that `p^weight*x=target` is solvable exactly when `p^min(weight,depth)` divides `target.val`, and every admissible fibre has cardinality `p^min(weight,depth)`. It then proves the coordinatewise criterion and the exact conditional count `p^(sum min(weight,depth))` for finite diagonal systems, including cardinality zero when a coordinate is inadmissible. The split certificate API gives the analogous product statement for abstract finite fibres, with the residue cardinality supplied as `q`.

Boundary. The counted manuscript theorem is the concrete `ZMod(p^L)` result with `q=p`. The arbitrary-DVR, residue-size-`q` extension is classical/certificate-explicit, not derived by Lean. Diagonal weights and targets are inputs; Lean does not construct Smith normal form. A literal zero diagonal factor is represented at precision `L` by a weight at least `L`, or handled by an explicit abstract coordinate certificate.

15.19 Row 19: Power-defect identity

Classification: Stronger.

Declaration targets

`TLD.defect`

`TLD.power_defect_identity_all`

`TLD.power_defect_identity`

`TLD.powers_compatible_of_defect_eq_zero`

`TLD.polynomialAction`

`TLD.polynomialAction_compatible_of_defect_eq_zero`

`TLD.continuousPolynomialAction_compatible_of_defect_eq_zero`

`LD.power_defect_identity_all`

Checked scope. The primary theorem is typed for distinct fine and coarse semiring modules and proves the ordered telescoping identity for every natural `n`, including zero. Zero one-step defect gives exact compatibility with every power and every finite polynomial functional calculus; the continuous version is also proved.

Boundary. This is linear algebra and finite polynomial calculus; no nonlinear higher-jet, operator-tree defect expansion, or analytic functional calculus is asserted.

15.20 Row 20: Power-defect norm bound

Classification: Stronger.

Declaration targets

`TLD.continuousDefect`

`TLD.continuous_power_defect_identity_all`

`TLD.continuous_end_pow_norm_le`

`TLD.continuous_norm_power_defect_le`

`TLD.continuous_norm_power_defect_le_geometric_majorant`

`LD.norm_power_defect_le_raw`

`LD.norm_power_defect_le`

Checked scope. Over a nontrivially normed field, continuous maps between distinct normed fine and coarse spaces satisfy the exact finite-sum operator-norm bound for every natural `n`, including zero. If both dynamics have norm at most `rho`, Lean derives the closed majorant `n*rho^(n-1)*norm(defect)`. No completeness assumption on the spaces is needed.

Boundary. These are operator-norm estimates, not spectral-radius estimates, equality statements, or nonlinear defect bounds. The majorant takes `rho` and both norm inequalities as hypotheses.

15.21 Row 21: Resolvent-defect identity

Classification: Stronger / certificate-explicit.

Declaration targets

`TLD.TwoSidedInverse`

`TLD.scalarShift_resolvent_identity`

`TLD.resolvents_compatible_of_defect_eq_zero`

`TLD.continuousScalarShift`

`TLD.continuous_scalarShift_resolvent_identity`

`TLD.continuous_norm_resolvent_defect_le`

`TLD.continuous_resolvents_compatible_of_defect_eq_zero`

`LD.resolvent_defect_identity`

Checked scope. Algebraically, supplied two-sided inverse witnesses give the typed resolvent identity on distinct modules. For continuous maps, the proof needs only `(zI-Tf)Rf=I` and `Rc(zI-Tc)=I` and yields `norm(C Rf-Rc C) <= norm(Rc)*norm(defect)*norm(Rf)`; zero defect gives exact intertwining.

Boundary. All inverse equations are supplied witnesses. Lean does not construct resolvents, prove spectral membership, or infer inverse existence from analytic hypotheses.

15.22 Row 22: Smith phase inversion identity

Classification: Stronger / scoped realization.

Declaration targets

`SD.smithPhase`

`SD.smithPhase_neg_one_eq_inv`

Checked scope. Over any field in which `2` is nonzero, the pulled-back phase formula at `omega = -1` equals `z^-1` for `z != 0`; this includes `Q`, `R`, and `C` and does not need the manuscript's additional `z != -1` condition.

Boundary. Only the algebraic formula is checked. No Smith disk, Mobius-domain, radius-preservation, or physical impedance theorem is formalized.

15.23 Row 23: Residue analysis and synthesis algorithm

Classification: Specification.

Declaration targets

`PC.rangeSynthesis_rangeAnalysis`

`PC.rangeAnalysis_rangeSynthesis`

`PC.exactPacketCoordinates`

`NR.coordinateRefinementEquiv`

Checked scope. For finite complete projector families, range-restricted analysis and finite-sum synthesis are inverse linear maps; cross-level analysis/synthesis gives an exact coordinate equivalence.

Boundary. This is the mathematical contract behind the array routine, not extracted interleaving code or a proof about Python storage shapes.

15.24 Row 24: Weighted finite-precision inverse algorithm

Classification: Specification / certificate-explicit.

Declaration targets

`WT.finiteGeometricInverse`

`WT.prunedWordInverse`

`WT.map_finiteGeometricInverse_eq_prunedWordInverse`

`WT.le_mul_ceilDiv`

`WT.prunedWordInverse_spec_of_minWeight`

`WT.prunedDiagonalTransferInverse`

`WT.finitePrecisionPrunedDiagonalTransferInverse_of_minWeight`

`WT.finitePrecisionPrunedDiagonalTransferInverse_of_weightedEdges`

Checked scope. The bounded geometric loop maps exactly to an explicit signed sum of retained non-commutative words. For `m>0`, Lean proves `L <= m * ceil(L/m)`, so the pseudocode choice `N=ceil(L/m)` meets the cutoff premise. Filtration data and `L<=m*N` prove that the pruned expression is a two-sided

inverse; supplied `Dinv` and a normalized-transfer edge expansion lift it to the explicit pruned diagonal-plus-transfer inverse after applying `q`.

Boundary. This is a mathematical specification. The manuscript calls the perturbation `H`; the Lean parameter name `K` has no fraction-field meaning. The result does not provide an extracted modular matrix inverse, basis representation, zero test, complexity bound, or proof of equivalence to the companion Python implementation.

15.25 Row 25: Transfer-zeta walk construction algorithm

Classification: Specification / automatic.

Declaration targets

`TZ.walkTuplesAtWeight`

`TZ.mem_walkTuplesAtWeight_iff`

`TZ.tuple_captured_iff`

`TZ.coefficientAlgorithm`

`TZ.coefficientAlgorithm_correct`

`TZ.exactWalkFirstEdgeEquiv`

`TZ.coefficientAlgorithm_firstEdge`

`TZ.pathZetaAlgorithm_trace`

`TZ.weightedAdjacency`

`TZ.weightedAdjacency_coeff`

`TZ.enumeratedPathZetaMatrix_eq_matrixPathZetaMatrix`

`TZ.pathZeta_eq_inverse_mul_transferNumerator`

`TZ.finite_graph_pathZeta_rationality`

`TZ.finite_graph_constant_coefficient_recurrence`

Checked scope. The coefficient loop is a finite `Finset` enumeration with exact capture and cardinality theorems. A direct trace wrapper pairs its semantic walk count with the first-edge recurrence. Finite matrix determinant/adjugate data are then constructed and proved equal to the enumerated series, yielding the normalized rational identity and eventual recurrence without a supplied rationality certificate.

Boundary. The objects counted are directed walks, with repetitions allowed. The Lean polynomial matrix definitions are noncomputable, and no extracted determinant/adjugate program, cancellation or minimal-denominator routine, performance bound, or analytic convergence theorem is claimed.

A Navigation in the final traditional paper

The printed page numbers below refer to the final traditional companion, whose cover is unnumbered and whose preliminary pages use Roman numerals. They are not PDF viewer page indices. A shared label is a navigation aid. It does not extend the formal scope to additional claims in the surrounding section. In particular, the Smith phase display has no equation number.

Row	Canonical label	Final locator
1	`def:refinement-map`	Statement 3.2, p. 3
2	`thm:nested-refinement`	Statement 3.3, p. 3
3	`def:coordinate-refinement`	Statement 3.4, p. 4
4	`prop:perfect-reconstruction`	Statement 3.5, p. 4
5	`thm:transfer-block-refinement`	Statement 3.8, p. 6
6	`prop:residue-algebra`	Statement 4.2, p. 9
7	`thm:root-of-unity`	Statement 4.3, p. 10
8	`thm:polyphase-refinement`	Statement 4.6, p. 11
9	`thm:dyadic-odd`	Statement 4.9, p. 11
10	`thm:weighted-inverse`	Statement 5.10, p. 17
11	`cor:finite-precision`	Statement 5.11, p. 18
12	`thm:transfer-zeta`	Statement 6.6, p. 21
13	`cor:zeta-recurrence`	Statement 6.7, p. 22
14	`thm:bezout-projector`	Statement 7.5, p. 25
15	`def:separation-weight`	Statement 7.6, p. 26
16	`thm:precision-delay`	Statement 7.7, p. 26
17	`prop:dyadic-sign`	Statement 9.3, p. 32
18	`thm:smith-solve`	Statement 10.6, p. 36
19	`thm:power-defect`	Statement 11.1, p. 37
20	`cor:defect-norm`	Statement 11.3, p. 38
21	`thm:resolvent-defect`	Statement 11.5, p. 38
22	`eq:smith-inversion`	Section 12.3, Smith-phase identity, p. 43
23	`alg:residue`	Statement B.1, p. 47
24	`alg:weighted-inverse`	Statement B.2, p. 47
25	`alg:path-zeta`	Statement B.3, p. 48

B Source imports and pinned dependencies

This table records direct local imports. It is a graph of file dependencies rather than a logical dependency analysis of the theorems. The JSON companion also retains every direct external import. Some modules import the broad `Mathlib` module, while others list focused imports.

Module	Direct local imports
`ExactCoordinates`	`ProjectorCore`
`FibreRefinement`	`ProjectorCore`
`PaperII.KernelAudit`	`PaperII.NestedRefinement`, `PaperII.ResidueFourierCRT`, `PaperII.WeightedTransfer`, `PaperII.TransferZeta`, `PaperII.ResultantSeparation`, `PaperII.SmithDyadic`, `PaperII.LinearDefects`, `PaperII.TypedLinearDefects`
`PaperII.LinearDefects`	External imports only.
`PaperII.NestedRefinement`	`ExactCoordinates`
`PaperII.ResidueFourierCRT`	`ResidueProjectors`
`PaperII.ResultantSeparation`	External imports only.
`PaperII.SmithDyadic`	External imports only.
`PaperII.TransferZeta`	External imports only.
`PaperII.TypedLinearDefects`	External imports only.
`PaperII.WeightedTransfer`	External imports only.
`PaperII`	`PaperII.NestedRefinement`, `PaperII.ResidueFourierCRT`, `PaperII.WeightedTransfer`, `PaperII.TransferZeta`, `PaperII.ResultantSeparation`, `PaperII.SmithDyadic`, `PaperII.LinearDefects`, `PaperII.TypedLinearDefects`
`ProjectorCore`	External imports only.
`ResidueProjectors`	`FibreRefinement`
`KomorebiVerification`	`PaperII`

The exact toolchain string is `leanprover/lean4:v4.34.0-rc2`. The Lake manifest pins the following source revisions. Its descriptive `inputRev` fields such as `master` and `main` do not replace these exact revisions.

Package	Exact revision
mathlib	`aeb72f1e9981b6d35b426b16dca7759be3e7ba06`
plausible	`d9598f07b1bc701f1e3aae163d2681c1fd978793`
LeanSearchClient	`ba67e212be1197b84c1f1f6299488a10a3002713`
importGraph	`d8823026ac7ef130c253089d95685f9877b95323`
proofwidgets	`a8acbfd87375ff4abe14ce09db5b7664d383bc7f`

Package	Exact revision
aesop	`18889deb9e83ea7420ef51c160d6f88552e744e3`
Qq	`507746ab8f4b643ccdacb2ec4cdb5853fa9f8ab3`
batteries	`4cac2177c37f5530c4da76aa8e4307f3fc9e4dcb`
Cli	`ab3a82db9fea14cf0fd7f5a2de650f4b534640af`

C Source excerpts and reproduction inventory

Every printed Lean listing comes from an exact UTF-8 excerpt in `excerpts/*.lean.txt`. The adjacent `*.render.txt` file escapes TeX control characters and substitutes a math font glyph where the code font lacks one. It preserves token order. Exact source text, not the display fallback, is the object to copy into an editor. Excerpts retain their surrounding source context only when that context is printed. They are not standalone Lean files. `generated/excerpts.json` records each source path, inclusive source line range, the SHA-256 digests of each full source file, exact excerpt, and display file. `generated/theorem_map.json` contains all 25 rows with fully qualified targets, classifications, scopes, boundaries, and lexical source locations. `generated/mapped_source_index.json` provides an alphabetical target index. `generated/static_identity_audit.json` records all manifest comparisons and the optional baseline comparison performed for this edition.

`build_guide_data.py` uses only the Python standard library. It generates the map, import table, pins, excerpts, and current static audit without invoking Lean. `build_guide.sh` calls that generator and runs XeLaTeX four times. The bundled P052, Nimbus Sans, DejaVu Sans Mono, and Latin Modern Math fonts provide the typography for this edition. The required LaTeX packages are listed in the preamble. The independent PDF check script uses PyMuPDF and Pillow for page rendering and contact sheets.

The historical source snapshot remains preserved. Current additions live in the separate current project and are indexed in the current correspondence. Editing this guide cannot by itself repair or strengthen a Lean theorem. A mathematical change to the formal component requires a separate source change, a new compile, an updated map when appropriate, and new execution evidence.