

RESEARCH LIBRARY

Signal Reconstruction

Polyphase sampling, unknown edges, and self-calibration

Lean and evidence guide

Komorebi P.

Standardized publication edition / 8 October 2026

REALIZATION CALCULUS

Reading scope

Exact source contracts for packets, transfer blocks, supplied edges, measurable masks, and finite pruning.
Unknown-edge estimation and self-calibration obligations are explicitly mapped.

Contents

1 Reading this guide 2

2 The recovered formal source 2

3 How to read the Lean contracts 3

4 Coordinate packets 4

5 Addresses and signed integer polyphase coordinates 5

6 The Hilbert-space bridge 6

7 Transfer blocks and finite-support convolution 8

8 Finite Laurent filter banks 9

9 Supplied known edges 10

10 Measurable masks and extended norms 10

11 Finite packet trees and complete prunings 12

12 Selection, optimality, and work 12

13 The unformalized Signal obligations 14

14 Computation is a separate evidence layer 15

15 Concordance with the traditional paper 16

16 Reproducing and inspecting the evidence 21

17 Source inventory 21

1 Reading this guide

Signal reconstruction has three distinct kinds of evidence in this edition: mathematical proofs in the traditional paper, finite computations with preserved outputs, and related Lean proofs inherited from Paper V. The unknown-edge estimator and the self-calibration procedure are developed in the first two layers. The recovered Lean sources establish packet, transfer, known-edge, mask, and finite-pruning contracts that support parts of the framework. They do not supply an end-to-end formal proof of the July signal-recovery algorithms.

This guide makes those relationships inspectable. It presents literal source excerpts, explains their types and hypotheses, and identifies the additional obligations required by the Signal paper. The editable Markdown files are included with the TeX, PDFs, original sources, mappings, and verification records. A displayed source excerpt retains its original declaration name; explanatory mathematics surrounding it is interpretation, not an additional declaration checked by Lean. Some excerpts stop at the theorem's type and proof delimiter; the complete proof remains in the quoted raw source file. The excerpts are not standalone modules.

The author is **Komorebi P.** The edition date is October 6, 2026. The word "complete" describes this publication set and its disclosed evidence, not a claim that all Signal theorems have been formalized.

1.1 Evidence layers

Evidence	What it establishes	Limit
Current traditional proofs	The stated mathematical implications under their explicit assumptions	Human mathematical proofs, with external theorem inputs identified
Preserved Paper V source and historical build	Related typed theorem statements and the archived checks for that source version	Inherited evidence; no new Signal elaboration
Fresh static correspondence	Saved contract names, source files and lines agree with the inherited audit	Text and hash validation, not kernel execution
Fresh Python runs	The declared finite cases and negative controls reproduce their recorded behavior	Bounded computations, not universal proofs
Source review and counterexamples	Specific overstatements are repaired or excluded	Neither a compiler result nor a blanket correctness certificate

The current runtime probe found neither `lean` nor `lake` on the active PATH. No new Lean build was attempted. `verification/FRESH_LEAN_STATUS.json` records this narrow fact. A passing publication build, an absence of `sorry` in text, or a matching declaration inventory cannot substitute for elaboration and kernel checking.

2 The recovered formal source

The related formal tree is `related_formal/paper_v/formalization/lean/`. It comes from the checked **Paper V v1.1.0 checkpoint**, `Paper_V_Complete_v1.1.0_With_Visual_Appendix.zip`. The archive has

23,437,514 bytes and 493 members, and passed the archive CRC check. Its SHA-256 is:

`7a15b94133739c26b8da5aadb2a2415c0bc539888430cec1baf0fb6425ea580e`

The later archive `Komorebi_Paper_V_Final_Traditional_Visual_Lean_2026-10-05.zip` was retrieved twice, including an independent retry, and neither downloaded copy could be opened as a ZIP. Both had the advertised size and the same SHA-256. Python reported `BadZipFile:Fileisnotazipfile`; an end-of-central-directory signature was not found. The cause of the malformed payload has not been established. It was not used as verified formal evidence. The failure, retry, and checked fallback are separately recorded in `provenance/PAPER_V_FINAL_ARCHIVE_FAILURE.json`, `provenance/PAPER_V_FINAL_ARCHIVE_RETRY.json`, and `provenance/PAPER_V_CHECKED_CHECKPOINT.json`.

The release preserves the selected checkpoint's formal sources, contracts, logs, and relevant metadata. It does not contain a portable Lean runtime or the complete original Paper V distribution. The original full verification driver expects files beyond this selected related-source subset.

2.1 Version pins and historical checks

Item	Recorded identity
Lean toolchain	`leanprover/lean4:v4.34.0-rc2`
Lean commit	`6a10ac8c22beadecabdbb0919c2b50214762f91d`
Mathlib commit	`aeb72f1e9981b6d35b426b16dca7759be3e7ba06`
Warning policy	`-DwarningAsError=true`
Implicit undeclared variables	`autoImplicit=false`
Historical scoped manuscript results	20
Historical explicit theorem targets	225

The archived build log reports `Buildcompletedsuccessfully(3296jobs).` These jobs include dependency compilation; they are not 3,296 new theorems. The inherited axiom audit allows `propext`, `Classical.choice`, and `Quot.sound`, and also records declarations that depend on no axioms. The archived register reports 827 declarations in the namespace, including compiler-generated declarations. That number has a different meaning from the current textual count of explicit source declarations.

The fresh static review checks every one of the 225 contract targets against its exact source filename, line, and theorem base identifier, and checks the inherited audit's corresponding axiom record. `verification/RELATED_FORMAL_STATIC_CORRESPONDENCE.json` reports that consistency check. It is separate from the historical build and does not establish that the saved source can be compiled in the current environment.

The 20-result English-to-Lean correspondence in the old scope contract was manually reviewed. A kernel checks the formal type, not whether an English sentence elsewhere in a PDF has been faithfully translated. The current per-SG concordance therefore includes the live Signal statement, hypotheses and domain, the exact related declarations, and an explicit coverage classification.

3 How to read the Lean contracts

Lean declarations expose several distinctions that are easy to blur in prose. A function type `E → E` does not assert linearity. A continuous linear map type asserts both linearity and continuity. An equivalence `I ≃ J` supplies inverse functions; a linear isometry equivalence adds linearity and norm preservation.

A theorem over an arbitrary type can be algebraic even when the motivating application is an analytic signal.

Source notation	Meaning used here
`[AddCommGroup E]`	An additive commutative group; subtraction and cancellation are available
`[SeminormedAddCommGroupE]`	An additive group with a seminorm; distinct vectors can have zero seminorm
`[Fintype R]`	A finite index type supporting an exact finite sum
`Fin N`	Integers from zero through $N-1$ with a proof of the bound
$\mathbb{Z} \to_0$ `K`	A finitely supported coefficient function, carrying finite support
`H` $\to$`L[`$\mathbb{C}$`] H`	A continuous complex-linear map
`H` $\simeq_{li}$`[`$\mathbb{C}$`] H'`	A complex-linear isometric equivalence
`HasSum u x`	The series of vectors u converges to x in the specified topology
`eLpNorm f p` μ	An extended nonnegative L^p seminorm, possibly infinite

Section variables apply to declarations below them. For example, a theorem whose printed first line mentions only a phase function may still require a finite phase type and an additive commutative monoid through the preceding `variable` command. The source-anchor inventory records these contexts in words alongside the literal excerpt. The full raw file remains authoritative when replaying the declaration.

A `noncomputable def` is a legitimate mathematical definition. It can use classical choice or operations on arbitrary real values that are not provided as executable algorithms. A proof about such a definition does not certify the performance of a Python implementation. This is particularly relevant to the pruning evaluator, whose real-valued cost comparison is defined noncomputably and whose work theorem concerns an explicit abstract cost model.

4 Coordinate packets

The first packet object is a coordinate restriction. Given a phase map $\phi : I \to R$, the packet $P_r x$ retains $x(i)$ exactly when $\phi(i) = r$, and is zero at all other addresses. The formal definition initially requires only a zero value in the coefficient type and decidable equality of phase labels.

Source excerpt: `Packets.lean`, lines 17–23. Context: I, R and V are arbitrary types; decidable equality on R and zero in V.

```
def packet (phase : I → R) (r : R) (x : I → V) : I → V :=
  fun i => if phase i = r then x i else 0

theorem packet_idempotent (phase : I → R) (r : R) (x : I → V) :
    packet phase r (packet phase r x) = packet phase r x := by
  funext i
  by_cases h : phase i = r <;> simp [packet, h]
```

The idempotence proof is pointwise: at an address in the selected phase, applying the same restriction

again retains the same value; outside that phase both restrictions return zero. Distinct phase labels give mutually annihilating restrictions. No Fourier transform, topology, smoothness, or recovery of an unknown phase map enters these proofs.

For a finite phase type and an additive commutative coefficient monoid, the source proves an exact completeness identity.

Source excerpt: `Packets.lean`, lines 36–45. Context: R is finite with decidable equality; V is an additive commutative monoid.

```
theorem packet_complete (phase : I → R) (x : I → V) (i : I) :
    (∑ r : R, packet phase r x i) = x i := by
  classical
  calc
    (∑ r : R, packet phase r x i) = packet phase (phase i) x i := by
      apply Finset.sum_eq_single (phase i)
      · intro b _ hb
        simp [packet, Ne.symm hb]
      · simp
    _ = x i := by simp [packet]
```

At each address exactly one summand survives, the one indexed by $\phi(i)$. This is a finite algebraic partition identity. It applies to all coordinates simultaneously, including negative integer addresses when $I = \mathbb{Z}$, but it does not by itself establish an infinite-series reconstruction or a bounded Hilbert-space operator. The later `HilbertPackets` module supplies those stronger structures on a particular signal space.

`packet_refinement` formalizes a fine phase map and a supplied coarsening map. A coarse packet is the sum of the fine packets mapped to its label. The theorem does not discover a favorable partition, infer hidden parameters, or claim that every refinement improves an approximation. It asserts the representation identity when the phase relation is given.

5 Addresses and signed integer polyphase coordinates

The address-equivalence abstraction starts with a known bijection $I \simeq R \times Q$. Analysis reads a sequence at the inverse address; synthesis sends each original address to its phase and quotient. Both round trips follow from the bijection. Nothing is discarded, so this is exact coordinate rearrangement rather than recovery from fewer observations.

Source excerpt: `Packets.lean`, lines 74–83. Context: I, R, Q and V are arbitrary types; e is a supplied equivalence I ≃ R × Q.

```
theorem address_synthesize_analyze (e : I ≃ R × Q) (x : I → V) :
    synthesizeAddress e (analyzeAddress e x) = x := by
  funext i
  simp [synthesizeAddress, analyzeAddress]

theorem address_analyze_synthesize (e : I ≃ R × Q) (c : R → Q → V) :
    analyzeAddress e (synthesizeAddress e c) = c := by
  funext r q
  simp [synthesizeAddress, analyzeAddress]
end AddressEquivalence
```

The integer specialization makes the division convention explicit. For $N > 0$, every $n \in \mathbb{Z}$ has an address

$n = Nq + r$ with $0 \le r < N$. The source uses Lean's Euclidean integer division and remainder. For example, when $N = 3$ and $n = -1$, the address is $(r, q) = (2, -1)$, not a quotient obtained by truncating toward zero.

Source excerpt: `IntegerPolyphase.lean`, lines 57–64. Context: N is natural and strictly positive; addresses and quotient coordinates are signed integers.

```
def addressEquiv (N : ℕ) (hN : 0 < N) : ℤ ≃ Fin N × ℤ where
  toFun n := (phase N hN n, quotient N n)
  invFun p := address N p.1 p.2
  left_inv := address_phase_quotient N hN
  right_inv p := by
    apply Prod.ext
    · exact phase_address N hN p.1 p.2
```

The `hN : 0 < N` argument is essential: the phase type `Fin N` must contain the remainder and the coordinate map must be invertible. Analysis itself can be written for any N, but the inverse-law theorems require positivity. `IntegerPolyphase.synthesis_analysis` and `analysis_synthesis` prove the two inverse identities for arbitrary coefficient functions.

5.1 Delay conventions

A delay by d in a channel index corresponds to a delay by Nd in the original sample address. A general original-sample delay changes the channel label and introduces a signed carry in the quotient.

Source excerpt: `IntegerPolyphase.lean`, lines 94–104. Context: N > 0; d is any signed integer; x is any coefficient function.

```
theorem sample_delay (N : ℕ) (hN : 0 < N) (d : ℤ) (x : ℤ → V)
    (r : Fin N) (q : ℤ) :
    analysis N (delay d x) r q =
      analysis N x (phase N hN ((r.val : ℤ) - d))
        (q + quotient N ((r.val : ℤ) - d)) := by
  dsimp [analysis, delay]
  congr 1
  have h := address_phase_quotient N hN ((r.val : ℤ) - d)
  dsimp [address] at h ⊢
  linear_combination -h
```

For $N = 3$, output phase $r = 0$, and original delay $d = 1$, the input address uses phase 2 and quotient $q - 1$. This carry is required by $3q - 1 = 3(q - 1) + 2$. An implementation that changes the phase label while retaining quotient q computes a different operator.

`residue_refinement` requires positive N, M and $N \mid M$. It is a literal reduction of a finer residue partition to a coarser one. It supplies infrastructure for phase-resolved signal descriptions; it does not prove that aliased exponential nodes can be identified from a selected measurement lattice. That latter claim needs an additional injectivity theorem for the actual observation set.

6 The Hilbert-space bridge

The formal Hilbert signal is a complex square-summable sequence. The exponent is part of the definition.

Source excerpt: `HilbertPackets.lean`, lines 16–24. Context: Complex square-summable sequences; arbitrary address type; decidable phase labels.

```
abbrev Signal (I : Type*) : Type _ := lp (fun _ : I => ℂ) 2

variable {I R S J : Type*}

noncomputable def projection [DecidableEq R] (phase : I → R) (r : R)
    (x : Signal I) : Signal I :=
  ⟨KomorebiPaperV.packet phase r x, (lp.mem ℓp x).mono' (by
    intro i
    by_cases h : phase i = r <;> simp [KomorebiPaperV.packet, h])⟩
```

The coordinate restriction belongs to the same square-summable space because its pointwise magnitude never exceeds the original sequence's magnitude. The source then proves its norm bound, linearity, and continuity, and constructs a genuine bounded linear projector. Idempotence and annihilation now hold as equalities of continuous linear maps.

`projector_complete` says that the finite sum of these bounded projectors is the identity operator. `projector_symmetric` and `projector_orthogonal` establish the inner-product structure. These results strengthen the earlier coordinate identities in a specified analytic space; they are not generic consequences for every topology on every sequence space.

6.1 Exact polyphase energy

The output is square-summable on the **joint index** `Fin N × ℤ`. It is not the supremum norm on a function space of channels. The isometry therefore gives the intended total channel energy.

Source excerpt: `HilbertPackets.lean`, lines 116–133. Context: $N > 0$ and the input is complex ℓ^2; the output uses joint ℓ^2 on Fin $N \times \mathbb{Z}$.

```
noncomputable def polyphase (N : ℕ) (hN : 0 < N) :
    Signal ℤ ≃ₗᵢ[ℂ] Signal (Fin N × ℤ) :=
  reindexIsometry (KomorebiPaperV.IntegerPolyphase.addressEquiv N hN)

theorem polyphase_apply (N : ℕ) (hN : 0 < N) (x : Signal ℤ) (r : Fin N) (q : ℤ) :
    polyphase N hN x (r, q) = x ((N : ℤ) * q + (r.val : ℤ)) := rfl

theorem polyphase_synthesis_analysis (N : ℕ) (hN : 0 < N) (x : Signal ℤ) :
    (polyphase N hN).symm (polyphase N hN x) = x :=
  (polyphase N hN).symm_apply_apply x

theorem polyphase_analysis_synthesis (N : ℕ) (hN : 0 < N) (x : Signal (Fin N × ℤ)) :
    polyphase N hN ((polyphase N hN).symm x) = x :=
  (polyphase N hN).apply_symm_apply x

theorem polyphase_energy (N : ℕ) (hN : 0 < N) (x : Signal ℤ) :
    ‖polyphase N hN x‖ ^ 2 = ‖x‖ ^ 2 := by
  rw [(polyphase N hN).norm_map]
```

In conventional notation, if $c_r(q) = x(Nq + r)$, the theorem corresponds to

$$\sum_{r=0}^{N-1} \sum_{q \in \mathbb{Z}} |c_r(q)|^2 = \sum_{n \in \mathbb{Z}} |x(n)|^2.$$

The displayed equality interprets the formal norm identity. The proof uses a bijective reindexing of square-summable coordinates, so it does not require a Fourier-series convergence argument.

6.2 Impulses and convergence

Source excerpt: `HilbertPackets.lean`, lines 135–137. Context: x is a complex square-summable sequence on $\mathbb{Z}$.

```
theorem impulse_expansion (x : Signal ℤ) :
    HasSum (fun i : ℤ => lp.single 2 i (x i)) x :=
  lp.hasSum_single (by norm_num) x
```

Here `lp.single 2 i (x i)` is a one-coordinate square-summable vector. `HasSum` asserts convergence in the ℓ^2 norm, not merely formal equality of coefficients. The companion finite-support theorem converts this series to a finite sum when all coefficients outside a supplied finite set vanish. The shift-generated impulse span is dense in the same sequence space.

An impulse in this module is a coordinate vector. It is not the Dirac measure used in the distributional derivative of a discontinuous function on a continuous torus. Moving from a continuous signal to Fourier moments, then to an exponential node model, requires additional definitions and proofs. That bridge is part of the current traditional argument and remains outside this archived formal target set.

7 Transfer blocks and finite-support convolution

`FIRTransfer` formalizes convolution by a finitely supported kernel. Its coefficient type is any semiring, and the input is an arbitrary bilateral sequence. This generality is possible because every sum in the time-domain law is finite.

Source excerpt: `FIRTransfer.lean`, lines 53–57. Context: K is a semiring; h is finitely supported; x is any bilateral sequence.

```
def fir (h : ℤ →₀ K) (x : ℤ → K) (n : ℤ) : K :=
  ∑ t ∈ h.support, h t * x (n - t)

def kernel (N : ℕ) (h : ℤ →₀ K) (k s : Fin N) (m : ℤ) : K :=
  h (tapIndex N k s m)
```

For output channel k, input channel s, and channel delay m, the transfer coefficient is

$$h_{k,s}(m) = h(Nm + k - s).$$

The signed tap formula is derived from the address identity

$$Nn + k - (Nm + k - s) = N(n - m) + s.$$

The `tapEquiv` bijection assigns every original tap to one input-channel/delay pair for each fixed output channel. `transferSupport` and `delays` retain the actual finite support rather than summing an invented infinite tail.

Source excerpt: `FIRTransfer.lean`, lines 96–101. Context: K is a semiring; N > 0; h is finitely supported; x is an arbitrary $\mathbb{Z}$-indexed function.

```
theorem fir_polyphase_transfer (N : ℕ) (hN : 0 < N) (h : ℤ →₀ K)
    (x : ℤ → K) (k : Fin N) (n : ℤ) :
    analysis N (fir h x) k n =
      ∑ s : Fin N, ∑ m ∈ delays N hN h k s,
        kernel N h k s m * analysis N x s (n - m) := by
  change fir h x (address N k n) = _
```

This is an exact operator identity for a supplied kernel. It does not certify a numerical estimator for an unknown kernel, select model order, or prove convergence of an infinite impulse response. To use a bilateral z-transform, the associated transform theorems require a nonzero evaluation point and a convergent weighted series for each input channel.

8 Finite Laurent filter banks

`FilterBank.Laurent` is the finite Laurent coefficient algebra `AddMonoidAlgebra ℂ ℤ`. A matrix of these coefficients acts on a finite set of complex bilateral channels. The source proves that multiplication of Laurent matrices agrees with composition of the corresponding channel operators, and that impulses distinguish their coefficients.

The perfect-reconstruction statement quantifies over **every** channel sequence. It therefore characterizes the operator identity rather than checking a particular example.

Source excerpt: `FilterBank.lean`, lines 133–136. Context: I is finite with decidable equality; E and R have finite complex Laurent entries; c and d are arbitrary.

```
theorem perfect_reconstruction_iff (E R : Matrix I I Laurent) (c : ℂ) (d : ℤ) :
    (∀ x : I → Signal, ∀ k n, bank R (bank E x) k n = c*x k (n-d)) ↔
      R*E = gainDelay c d := by
  rw [← bank_faithful]
```

`gainDelay c d` is a diagonal matrix whose action is gain c and channel delay d. No nonzero-gain hypothesis appears in this equivalence: for $c = 0$, it correctly characterizes the zero composite operator. To obtain an invertible reconstruction interpretation one must separately assume the desired nonzero gain.

The exact correspondence is useful because a matrix formula and its implemented signal action must agree. It does not identify the analysis or synthesis bank from unknown data, and it does not supply the frequency-offset calibration introduced in the Signal paper. The offsets are integers; the recovered node locations are continuous.

Source excerpt: `FilterBank.lean`, lines 155–161. Context: N > 0; the supplied Laurent matrix identity R*E = gainDelay c d holds.

```
theorem original_sample_delay (N : ℕ) (hN : 0 < N)
    (E R : Matrix (Fin N) (Fin N) Laurent) (c : ℂ) (d : ℤ)
    (h : R*E = gainDelay c d) (x : Signal) (n : ℤ) :
    KomorebiPaperV.IntegerPolyphase.synthesis N hN
      (bank R (bank E (KomorebiPaperV.IntegerPolyphase.analysis N x))) n = c*x (n-(N:ℤ)*d) := by
  have hr := (perfect_reconstruction_iff E R c d).mpr h
  unfold KomorebiPaperV.IntegerPolyphase.synthesis
```

After returning to original sample coordinates, the delay is Nd. Replacing it with d changes the claim. The transform-level theorem `bank_transform` also retains `HasTransform` assumptions; arbitrary bilateral inputs need not possess a convergent bilateral transform at a chosen point.

9 Supplied known edges

The known-edge result captures the key error cancellation when an edge component is retained exactly. The namespace is `KomorebiPaperV`; `KnownEdge` is the filename, not an extra namespace component.

Source excerpt: `KnownEdge.lean`, lines 17–19. Context: E is an additive commutative group; $Q : E \to E$ is arbitrary; the same supplied j is retained.

```
theorem known_edge_error (s j : E) (Q : E → E) :
    (s + j) - (Q s + j) = s - Q s := by
  abel
```

The function Q is arbitrary. The proof does not need Q to be linear, idempotent, bounded, or a Fourier truncation. Both sides contain the same supplied edge j, so additive cancellation removes it. In a seminormed additive group the source immediately obtains the equality of error seminorms.

This is a meaningful formal statement, but its strength is conditional: the edge has already been supplied exactly. It contains no edge locations, jump amplitudes, measured moments, noise bound, Hankel matrix, or unknown-order recovery algorithm.

When the supplied edge is an estimate, the algebraic decomposition and triangle inequality give a two-term error budget.

Source excerpt: `KnownEdge.lean`, lines 43–46. Context: E is a seminormed additive commutative group; all four elements are supplied.

```
theorem estimated_edge_norm_bound (s shat j jhat : E) :
    ‖(s + j) - (shat + jhat)‖ ≤ ‖s - shat‖ + ‖j - jhat‖ := by
  rw [estimated_edge_error]
  exact norm_add_le _ _
```

In an L^p application, one can instantiate the relevant normed-space structure once the functions represent members of that space. A rate for $s - Qs$ and a bound for $j - \hat{j}$ remain separate analytic obligations. The source does not provide them simply because the cancellation theorem is available.

The current resolved-synthesis inequality may require a bounded **linear** reconstruction map A_M when it moves an edge-estimation error through that map. That assumption serves a different purpose from the arbitrary Q in exact known-edge cancellation. The two contracts should not be merged.

10 Measurable masks and extended norms

The mask object `jump S a` is the function equal to complex amplitude a on a set S and zero elsewhere. The exact equal-amplitude mask error is controlled by the measure of the symmetric difference. For finite exponent the source uses an extended-valued norm.

Source excerpt: `MaskPerturbation.lean`, lines 43–47. Context: Ω is measurable; S,T measurable; μ arbitrary; p : ENNReal with $1 \le$ p and p $\neq \infty$; a,b complex.

```
theorem finite_p_perturbation (μ : Measure Ω) (S T : Set Ω)
    (hS : MeasurableSet S) (hT : MeasurableSet T) (p : ENNReal)
    (hp : 1 ≤ p) (hpTop : p ≠ ∞) (a b : ℂ) :
    eLpNorm (jump S a-jump T b) p μ ≤
      ‖a-b‖ₑ * μ S ^ (1 / p.toReal) + ‖b‖ₑ * μ (S △ T) ^ (1 / p.toReal) := by
```

The hypotheses are measurable S, T, an arbitrary measure μ, $1 \le p < \infty$, and complex amplitudes a, b. The right side separates amplitude error on the original mask from a changed-mask term:

$$\|a-b\|\,\mu(S)^{1/p} + \|b\|\,\mu(S \triangle T)^{1/p}.$$

This conventional finite expression is an interpretation when the indicated quantities are finite. The actual Lean statement lives in `ENNReal` and remains valid if an extended norm or measure is infinite. One must establish finiteness before translating it into ordinary real arithmetic. In particular, `ENNReal.toReal` is not a safe automatic coercion of an arbitrary extended inequality: it sends infinity to zero.

The full-signal theorem adds the background error and assumes that both backgrounds are almost everywhere strongly measurable. It is again a bound for supplied functions, masks and amplitudes. It is not a theorem deriving those masks from observations.

10.1 Infinity norm and actual set equality

Source excerpt: `MaskPerturbation.lean`, lines 73–77. Context: Ω is measurable; μ arbitrary; S,T arbitrary sets; a,b complex; actual S△T=∅ condition.

```
theorem infinity_perturbation (μ : Measure Ω) (S T : Set Ω) (a b : ℂ) :
    eLpNorm (jump S a-jump T b) ∞ μ ≤
      ENNReal.ofReal (‖a-b‖ + if S △ T = ∅ then 0 else ‖b‖) := by
  rw [eLpNorm_exponent_top]
  exact eLpNormEssSup_le_of_ae_bound (ae_of_all μ (pointwise_sup_bound S T a b))
```

The source's indicator condition is $S \triangle T = \varnothing$ **as an actual set**. It is not the assertion that the symmetric difference has measure zero. The proof first establishes a bound at every point and then passes to the essential supremum. Consequently this bound can remain coarse even if the masks agree almost everywhere. Replacing its set condition by a measure-zero condition would state a different theorem, although a sharper essential-supremum theorem may be possible with an additional argument.

Small displacement of a jump can produce a small finite-p error and a nonvanishing infinity error. This is why an unknown-edge location estimate alone does not automatically yield uniform signal convergence across discontinuities. A Fourier reconstruction also determines an almost-everywhere class unless a representative, such as a right-continuous one, is fixed.

10.2 The zero-mean periodic edge bound

The traditional paper repairs the location term in a zero-mean periodic edge estimate. For the period-T sawtooth $G(x) = 1/2 - x/T$, interpreted periodically, its distributional derivative is $\delta_0 - dx/T$. For canonical mean-zero packets with matched real jumps, both amplitude sums zero, and circular displacement $d_\ell \le T/2$, the repaired estimate is

$$\|j - \widehat{j}\|_1 \le \frac{T}{4} \sum_{\ell} |a_\ell - \widehat{a}_\ell| + 2 \sum_{\ell} |\widehat{a}_\ell| d_\ell (1 - d_\ell / T).$$

The factor two comes from the translated **zero-mean** sawtooth, not merely the measure of one interval. The old printed bound without that factor fails even for a shifted single interval with its mean removed. The original source is preserved; the corrected statement appears in the new paper and review. `MaskPerturbation.finite_p_perturbation` provides related indicator-function infrastructure, but it does not formalize this periodic primitive calculation. The new sawtooth bound has a prose proof and finite controls, without a recovered Lean target.

11 Finite packet trees and complete prunings

The finite-tree formalization distinguishes a complete representation from an arbitrary collection of convenient packets. A cut either stops at a node or splits into cuts of all children. A complete node set meets each terminal path exactly once. The source proves the equivalence between complete sets and covering antichains and provides an equivalence with the inductive cut type.

Under a conservation law saying that each parent value is the sum of its child values, every complete pruning synthesizes the root. For operator-valued labels this yields an actual identity on the input space.

Source excerpt: `PruningSums.lean`, lines 88–97. Context: H is a complex normed space; a finite tree, conservation law, identity root and complete pruning are supplied.

```
theorem packet_projector_synthesis {H : Type*} [NormedAddCommGroup H] [NormedSpace ℂ H]
    (p : A → H →L[ℂ] H) (t : FiniteAdaptive.Tree A)
    (h : conserves p t) (hr : p (label t) = ContinuousLinearMap.id ℂ H)
    (S : Pruning t) (x : H) :
    nodeSum p S = ContinuousLinearMap.id ℂ H ∧
      (∑ v ∈ S.val.toFinite.toFinset, p (nodeLabel v) x) = x := by
  have hs := (antichain_synthesis p t h S).trans hr
  refine ⟨hs, ?_⟩
  have hx := congrArg (fun T : H →L[ℂ] H => T x) hs
  simpa [nodeSum] using hx
```

The conservation assumption `h` and root identity `hr` are input hypotheses. The result proves their propagation to a complete covering antichain; it does not verify that a particular image dictionary satisfies them. Nor does it assert that packets selected by an unrelated greedy fitting routine form such an antichain.

The finite grammar matters. The proof uses finite sums and a finite cut type. An ideal unbounded family of geometric packets and an implementation with finite candidate caps require different arguments for approximation, convergence, or selection.

12 Selection, optimality, and work

For an additive node-cost objective the formal evaluator compares stopping at the current node with splitting into all child solutions. Its real-valued costs are supplied inputs, and it stops on ties.

Source excerpt: `PruningAlgorithm.lean`, lines 13–30. Context: A finite adaptive tree; arbitrary real node costs; noncomputable real comparison and cached child model.

```
structure Result (t : FiniteAdaptive.Tree A) where
  cut : Cut t
  cost : ℝ
  comparisons : ℕ
  additions : ℕ
  visits : ℕ

noncomputable def solve (c : A → ℝ) : (t : FiniteAdaptive.Tree A) → Result t
  | .leaf a => ⟨.stop _, c a, 0, 0, 1⟩
  | .branch a n ch =>
      let child := fun i => solve c (ch i)
      let total := ∑ i, (child i).cost
      let stop := c a ≤ total
      ⟨if stop then .stop _ else .split (fun i => (child i).cut),
       min (c a) total,
       1 + ∑ i, (child i).comparisons,
       (n+1) + ∑ i, (child i).additions,
       1 + ∑ i, (child i).visits⟩
```

`solve_correct` identifies both its reported cost and its chosen cut with the already proved dynamic-programming value and minimizer. `computed_antichain_optimal` carries this additive optimum to a complete antichain.

Source excerpt: `PruningAlgorithm.lean`, lines 89–95. Context: Finite tree and supplied node costs; unit-cost visits/comparisons/additions as defined in solve.

```
theorem solve_linear_work (c : A → ℝ) (t : FiniteAdaptive.Tree A) :
    (solve c t).visits + (solve c t).comparisons + (solve c t).additions ≤ 3 * nodes t := by
  have hv := solve_visits c t
  have hc := solve_comparisons c t
  have ha := solve_additions c t
  have hi := internal_le_nodes t
  omega
```

The work expression counts one node visit, one comparison at each internal node, and one addition per child when summing cached values. The linear bound excludes the cost of producing node costs, geometric fitting, numerical linear algebra, storage of a fully expanded output, and wall-clock execution of Lean or Python. The source makes this unit-cost model explicit.

For a possibly nonadditive objective $R(S) + \eta L(S) + \tau K(S)$, the source proves existence of a minimizer because the admissible pruning family is finite. That does not imply that the same local recurrence computes it. A globally coupled leakage term can break the additive decomposition needed by dynamic programming.

12.1 Stability needs a margin

The antichain selector theorem assumes a positive objective gap γ and an error envelope satisfying $2E < \gamma$. The arbitrary-objective stability theorem similarly assumes a uniform error bound and the same strict margin. These hypotheses give a precise conditional statement. They do not derive a margin from a noise level alone.

The Signal branch-selection argument has the same logical structure: estimation error must be smaller

than the separation of competing branches. A rate $O(1/N)$ does not automatically beat a branch spacing also of order $1/N$. The constants and a strict margin must remain in the theorem.

13 The unformalized Signal obligations

The following obligations explain the gap between supplied-edge infrastructure and unknown-edge reconstruction. They identify concrete mathematical dependencies rather than treating all missing work as a single label.

13.1 Continuous signals to exponential moments

The observation model must fix the torus period and Fourier normalization, define the discontinuous edge component and its representative, and prove the distributional derivative and integration-by-parts relation. The recovered Paper V sequence-space impulses do not provide this continuous distributional theorem. Zero mean, zero total jump, and the missing constant coefficient must be handled explicitly.

If the smooth component occupies a known finite Fourier band, the out-of-band derivative moments reduce to a finite exponential sum $y_n = \sum_{\ell=1}^{L} a_\ell z_\ell^n$. The hypotheses include distinct nonzero unit-circle nodes and nonzero amplitudes after removing redundant zero jumps. The order-zero case is a separate exact branch, with no polynomial roots to solve.

13.2 Hankel factorization and annihilation

A dedicated formalization would need the finite Hankel factorization, Vandermonde invertibility, rank under the distinct-node hypotheses, the annihilating recurrence, and uniqueness of the recovered polynomial. None is a target in the preserved Paper V contract. The prose proof can use standard finite-dimensional linear algebra, but finite computations of ranks or roots do not certify the universal statement.

The full algebraic reconstruction includes polynomial root extraction and matching, not only linear algebra. A computational root routine can choose branches or return noisy approximations; its connection with an exact finite-set theorem must be stated separately.

13.3 Dictionary null secants

For a nuisance basis B and edge dictionary D, the exact identifiability condition excludes every nonzero sparse coefficient difference from the kernel of the nuisance-projected dictionary. It must distinguish a full-column-rank basis from a rank-deficient set of columns and define the sparse support size correctly.

Under the usual convention that a full-column-rank $m \times n$ matrix has spark $n + 1$, the familiar spark condition must be capped at $\min(2s, n)$; a different infinity convention must be declared if used. For a wide support submatrix, the compact SVD lists only $\min(m, k)$ singular values. A positive last returned singular value does not eliminate a column nullspace when $k > m$. Its restricted separation is zero. The numerical execution copy now tests these cases explicitly; this remains computational evidence rather than a Lean theorem.

13.4 Perturbation, roots, and matching

The singular-value threshold argument uses a spectral-norm perturbation bound. A guaranteed order gate requires a supplied threshold between the noise bound and the lower bound on the last nonzero

singular value. The largest adjacent singular-value ratio is a practical heuristic with a different contract.

An exact stress witness uses three unit-circle nodes $1, i, -1$, amplitudes $100, -99, -1$, and a 4×4 Hankel matrix. Its singular values are $(400, 396, 4, 0)$. Adding $0.1(-i)^n$ gives $(400, 396, 4, 0.4)$ with perturbation norm $0.4 < 4/2$. The largest adjacent ratio is $396/4 = 99$, exceeding $4/0.4 = 10$, so the ratio rule chooses order two. A supplied threshold in $(0.4, 3.6)$ still selects the correct order three. The exact Fourier-matrix diagonalization verifies this example; no new Lean certificate is asserted for it.

The Rouché root-disk argument requires disjoint true-root disks, a strict boundary inequality, and a coefficient error bound. For unit-circle root centers, $|z| \le 1 + \rho$ supplies the factor $\sum_k |\delta q_k|(1+\rho)^k$. This is a deterministic conditional stability certificate. Its true centers and true coefficient errors are not generally available from the observed data alone, so the theorem is not an independently computable a posteriori test without further estimates.

Location error must be measured after a specified matching, using a circular distance. An amplitude bound then needs the conditioning of the corresponding Vandermonde system. Without separation, uniformly Lipschitz location recovery fails even for fixed-order pairs of narrow pulses; it is not enough to give only a variable-order or undefined-metric example.

13.5 Self-calibration and branch resolution

The multi-offset phase identity requires at least one adjacent product. A branch-spacing statement requires at least two candidate branches. Phase denominators must be nonzero and must remain bounded away from zero for a stable inverse. The source's integer polyphase coordinate equivalence has none of these data-dependent quantities.

A complete formal statement would need the frequency-offset observation model, the phase-product identity, the normalization/inversion gate, the alias candidate set, the chosen branch metric, the strict separation margin, and the final reconstructed-signal bound. The offsets in this model are integers, while node locations vary continuously. Exact integer address arithmetic is relevant infrastructure but proves none of those estimation gates automatically.

13.6 Resolved synthesis and signal norm

After estimating an edge, the background recovery must be specified as a bounded linear map on the relevant error space when the proof uses its operator norm. The final error estimate combines background reconstruction, edge-estimation error, and any truncation or approximation term. Fourier information yields an almost-everywhere class, so pointwise statements at jumps require a fixed representative. The corrected periodic L^1 bound and the nonvanishing infinity error must retain their different conclusions.

14 Computation is a separate evidence layer

The release preserves the original run records and fresh finite replays, including negative cases. The initial complex benchmark validation required finite location and jump errors in every case. That condition failed: in `bursts_6_jump`, the model-order heuristic estimated nine jumps instead of six, and the evaluator reported infinite location and jump errors for the unmatched orders. The improved reconstruction RMSE was finite. These facts can coexist; improved signal error does not establish correct parameter identification.

The follow-up report scopes the benchmark to its actual measured outputs and retains the order-mismatch stress failure. The original failed record remains available. Neither the benchmark nor its

follow-up is a theorem proving reliable recovery of every complex signal.

The generic dictionary runner also required repairs. A QR basis computed from rank-deficient nuisance columns can include additional orthogonal directions; its projector then removes more than the intended column space. Compact SVD can falsely suggest positive separation for a wide support. The preserved original source and the rank-aware execution copy are distinct, with negative controls and provenance recording the correction. The traditional theorem handles the zero-edge branch explicitly. The native `EdgeModel` and `recover_unknown_edges` API still exclude order zero; the dictionary rank correction did not add an order-zero wrapper.

14.1 Adaptive image packets

The standalone image scripts receive the complete image array. They fit and extend a packet dictionary using those pixels, and their pixel-completion routine can retain the remaining residual. This is adaptive approximation of available data. It is not unknown-edge recovery from a finite collection of Fourier measurements.

Several additional limits matter when comparing the implementation with exact packet theory:

- The ideal geometric grammar is unbounded, while the tested generator uses finite candidate and mask caps.
- The default ridge term modifies least-squares refitting. Its mathematical residual law is exactly $M^{\mathsf{T}} r = \lambda c$. Numerical checks test that law within tolerance. The generally nonzero right side distinguishes ridge refitting from exact orthogonal projection onto a nested subspace.
- Clipping a reconstruction to $[0, 1]$ is nonlinear. For truths in that box it can improve the pixel error, but it is not the bounded linear operator used by the resolved-synthesis proof.
- Retaining a full residual gives an exact archive identity; it does not prove sparse compression or recovery from missing data.
- Some historical per-method runtime fields are zero placeholders because the implementation checked a method label that did not match the stored labels. Case-level runtime and fresh measurements must be read separately.

Finite tolerances in reconstruction, monotonicity, and stationarity tests are part of those computational contracts. Passing them supports the tested cases and tolerances. It does not turn the finite dictionary, ridge refit, or greedy search into the ideal complete-pruning theorem.

15 Concordance with the traditional paper

The machine-readable map `lean/Claim_Lean_Concordance.json` copies each current SG statement, hypotheses, and domain from `traditional/Claim_Concordance.json`, and records the hash of that live input. Related declaration anchors include qualified names, exact source files, lines and SHA-256 hashes. A corresponding CSV supports inspection without a JSON reader.

The classification “direct supplied-object contract” is used only when the archived theorem gives the relevant algebraic or operator implication after an explicitly described instantiation. “Related infrastructure” identifies a narrower result that helps express the mathematics but does not prove the whole SG claim. “No recovered target” means no declaration for that claim was identified in the inspected 225-target corpus; it does not assert that the mathematics is impossible to formalize or absent from all Lean libraries.

Claim	Archived relation	Remaining obligation
SG-01: Complete packet analysis and synthesis	Related infrastructure	The archived source proves coordinate and ℓ^2 projector specializations; the SG result is for arbitrary complete algebraic projectors.
SG-02: Transfer algebra and packet paths	Related infrastructure	Composition is proved for finite Laurent filter-bank operators; arbitrary packet transfer paths remain a prose theorem.
SG-03: Refinement preserves represented operators	Related infrastructure	The source proves supplied coordinate refinement and its bounded ℓ^2 specialization, not every abstract packet operator refinement.
SG-04: Pulled-back transform packets	Related infrastructure	A bijective coordinate reindexing is a particular isometric transform; the arbitrary invertible-transform conjugation theorem is not a recovered target.
SG-05: Polyphase analysis and convolution kernels	Related infrastructure	Exact integer round trips and finite-support convolution are formalized. The SG claim allows $\ell 1$ infinite-support kernels and needs its absolute-convergence argument.
SG-06: The arithmetic of residual alias ambiguity	No recovered target	Unit-circle power observations, finite alias group, and Bézout/gcd identification.
SG-07: Measurement and decomposition obstructions	No recovered target	Measurement-fiber identifiability and direct-sum uniqueness in the declared model.
SG-08: No uniform location stability without separation	No recovered target	Matched circular location metric and the fixed-order narrow-pulse counterexample.
SG-09: Out-of-band matched identification	Related infrastructure	The formal supplied-edge cancellation is related; Fourier out-of-band identification of an unknown scalar coefficient is additional.
SG-10: The singularity-exact block operator	Related infrastructure	Supplied-edge error cancellation is formal. The Fourier-defined block, its projector identities and exactness on the model class are separate prose obligations.

Claim	Archived relation	Remaining obligation
SG-11: Resolved error for an analytic background	Related infrastructure	The cancellation part is formal; analytic Fourier decay, tail summation and the concrete signal-norm instantiation are not targets here.
SG-12: The scaled interval-edge Gibbs profile	No recovered target	Dirichlet-kernel scaled limit, sine-integral profile, and the local Gibbs overshoot.
SG-13: Positive Fourier averaging preserves range	No recovered target	Fejér-kernel positivity, integral normalization, and range preservation almost everywhere.
SG-14: Residualized sparse injectivity	No recovered target	Projected sparse null-secant injectivity and the capped spark convention.
SG-15: Sparse noise stability and support thresholding	No recovered target	Restricted column separation, nuisance projection contraction, and sparse support bounds.
SG-16: Profiled one-packet discovery and its finite gap	Related infrastructure	Finite objective gap and stability are formal under a strict optimum and uniform objective errors; profiled dictionary identification and its particular score are additional.
SG-17: Mask error is a support geometry calculation	Related infrastructure	The source provides extended measurable-mask identities and bounds; the SG mixed-amplitude finite-vector identity is a distinct specialization/derivation.
SG-18: Differentiation isolates an exponential moment packet	No recovered target	Continuous distributional derivative and Fourier normalization to exponential moments.
SG-19: Hankel rank detects the actual order	No recovered target	Hankel/Vandermonde factorization, rank, and zero-order interpretation.
SG-20: Exact Prony annihilation and location recovery	No recovered target	Prony annihilator uniqueness, polynomial roots, and matched location recovery.
SG-21: Canonical periodic integration of jump data	No recovered target	Canonical mean-zero periodic primitive and its representative.
SG-22: Finite exact unknown-order reconstruction	No recovered target	Finite exact data budget, unknown/zero order branch, and synthesis modulo representative.

Claim	Archived relation	Remaining obligation
SG-23: Hankel perturbation from Fourier noise	No recovered target	Fourier-noise scaling and the rectangular Hankel Frobenius/spectral bound.
SG-24: Annihilator coefficient perturbation	No recovered target	Perturbed inverse/Neumann gate and annihilator coefficient error.
SG-25: Conditional simple-root disk localization	No recovered target	Strict Rouché disk gate, one root per disk, and conditional true-root matching.
SG-26: Amplitude stability with perturbed nodes	No recovered target	Perturbed full-column-rank least squares and amplitude stability.
SG-27: Shared-node offset channels	No recovered target	Defective moment model and shared aliased exponential nodes across offsets.
SG-28: Order recovery with a supplied singular threshold	No recovered target	Complex singular-value perturbation and a supplied threshold, including full rank.
SG-29: Exact phase from adjacent offset products	No recovered target	Adjacent offset products with $R \geq 1$ and nonzero amplitude.
SG-30: Deterministic offset-phase perturbation	No recovered target	Offset phase denominator and explicit defect-to-phase bound.
SG-31: A zero offset residual tests a progression	No recovered target	Normalized residual zero exactly characterizes a finite progression.
SG-32: Unique branch selection under a strict margin	Related infrastructure	The finite strict-margin selection principle is formal; circular Nth-root geometry, associated-root error and the particular metric gate remain additional.
SG-33: Associated roots divide the angular error	No recovered target	Associated root branch and divided angular perturbation.
SG-34: An explicit exact self-calibration budget	No recovered target	Explicit exact calibration data budget and distinct active alias nodes.
SG-35: Conditional asymptotic rates after calibrated lifting	No recovered target	Uniform inverse bounds, coefficient lower bounds, strict branch margin, and conditional rates.
SG-36: Resolved synthesis splits the two error sources	Related infrastructure	The additive error split is formal; the A_M linearity rewrite and its operator-norm amplification are not this source type.

Claim	Archived relation	Remaining obligation
SG-37: Sharp additive L1 control for normalized jump packets	Related infrastructure	Measurable-mask error infrastructure is related, but the corrected mean-zero periodic sawtooth calculation and sharp factor two are not formalized here.
SG-38: A conditional global L1 reconstruction rate	Related infrastructure	Supplied-error combination is related; calibrated parameter rates, canonical jump L1 control and Fourier operator-norm growth remain prose obligations.
SG-39: Shared-location multichannel recovery	No recovered target	Stacked full-column-rank recurrence, common annihilator, and channelwise Vandermonde recovery.
SG-40: Confluent annihilation and the leading offset phase	No recovered target	Confluent shift annihilation, polynomial degree, and nonvanishing leading offset coefficients.
SG-41: Model mismatch enters the same sparse stability bound	No recovered target	Restricted sparse injectivity with projected model mismatch and edge approximation error.
SG-42: Ideal nested-span refitting decreases residual	Related infrastructure	The source proves density and orthogonality for particular ℓ^2 coordinate packets; arbitrary nested finite-span projector convergence is not a recovered target.
SG-43: Ridge refitting has a different stationarity law	No recovered target	Positive ridge normal equations, uniqueness, and modified residual stationarity.
SG-44: Clipping is nonexpansive toward a bounded truth	No recovered target	Scalar clipping nonexpansiveness, box truth assumption, and Euclidean assembly.
SG-45: An exact residual archive preserves the object	No recovered target	Exact full residual identity and ideal finite pixel spanning; no compression inference.
SG-46: Bounded identities extend from a dense core	Related infrastructure	An explicit dense core and bounded finite projector identity are formal examples; the general bounded-operator extension theorem is not a recovered target.

The prose interpretation field is deliberately separate from `source_quoted`. An association does not silently import a Fourier model, observation scheme, numerical algorithm, or omitted hypothesis into the Lean type.

16 Reproducing and inspecting the evidence

From the release root, the static inventory and correspondence checks can be regenerated without Lean:

```
python lean/build_lean_inventory.py
python verification/check_related_formal.py
python lean/build_lean_guide.py
python build_release.py --only lean
```

The first two commands inspect saved source and saved evidence. The guide builder writes Markdown, TeX, anchors, and the per-claim map from those inputs; the publication builder invokes pdfLaTeX. None performs a formal proof check.

With the pinned Lean toolchain and Mathlib dependency installed, the preserved project's focused historical commands can be replayed from its Lean folder:

```
cd related_formal/paper_v/formalization/lean
lake env lean --version
lake build KomorebiPaperV
lake env lean -DwarningAsError=true KernelAudit.lean
```

These are future replay instructions, not commands reported as executed for this Signal edition. The `lake-manifest.json` pins the recorded dependency identity. Dependencies and a compatible toolchain must actually be available; the source archive alone does not supply an executable runtime. The selected related-source subset is sufficient to inspect the proof targets, but the original all-purpose `verification/verify.py` expects the full Paper V distribution and should not be treated as a self-contained Signal driver.

The verification register and run directories distinguish current computations, source preservation, static related-formal inspection, and inherited compiler logs. If the sources or claims change, the mappings must be regenerated and their input hashes reviewed. A later successful full Signal formalization would need its own source tree, typed observation model, target contract, actual build and axiom evidence, and English-to-formal correspondence review.

17 Source inventory

The current textual inventory scans nested-comment and string-masked source and includes noncomputable declarations. It finds **23 production modules and 349 explicitly named declarations**: 225 theorems, 107 definitions, seven abbreviations, nine inductive types, and one structure. All production modules occur in the root import list. Including the root import file and axiom harness gives 25 Lean files.

The 225 explicit theorem declarations agree with the checked target contract's names, files, lines and kinds. The remaining named declarations define the objects, coordinate maps, finite trees, evaluators, and related structures. The count measures a preserved source corpus, not independent new results or the completion percentage of Signal formalization.

Module	Theorems	Other explicit declarations	Principal role
`BinaryAdaptive`	6	8	Binary cuts and additive cost recurrence
`CRT`	4	0	Coprime residue arithmetic
`CompletePrunings`	15	7	Complete covering antichains and cut equivalence
`CyclicSeries`	15	1	Convergent cyclic coefficient series
`FIRTransfer`	6	8	Finite-support signed polyphase kernels
`FilterBank`	13	11	Finite Laurent bank action and reconstruction
`FiniteAdaptive`	18	15	Finite arbitrary-arity packet trees and cuts
`FiniteSelection`	5	0	Finite objective minimum and gap stability
`FiniteTransforms`	12	9	Finite signal coordinates, Haar/lifting and matrix action
`HaarChannels`	8	5	Literal normalized ℓ^2 low/detail channels
`HaarHilbert`	5	2	Haar isometry and energy
`HilbertPackets`	23	9	Bounded ℓ^2 projectors, impulses and polyphase isometry
`IntegerPolyphase`	11	7	Signed Euclidean addresses, inverse laws and delays
`KnownEdge`	6	0	Supplied-edge cancellation and error splitting
`Lifting`	4	3	Predict/update inverse and determinant identities

Module	Theorems	Other explicit declarations	Principal role
`MaskPerturbation`	9	1	Extended measurable-mask error bounds
`Packets`	8	4	Pointwise coordinate packets and address round trips
`PruningAlgorithm`	8	3	Cached bottom-up evaluator and work model
`PruningSums`	15	9	Literal antichain sums, synthesis and objectives
`SmithAction`	10	3	Finite matrix basis change for Smith form
`SmithCyclic`	12	8	Smith-compatible cyclic coefficient action
`WaveletPackets`	7	8	Finite-depth Haar packet operators and reconstruction
`ZTransform`	5	3	Convergent bilateral transforms and FIR channel formulae

Full declaration and module inventories are included in JSON and CSV. `lean/SOURCE_ANCHORS.json` contains the exact quoted excerpts used in this guide, their contexts, their interpretations, and their source hashes. `reviews/LEAN_REPORT.md` records the current review and publication checks. Original `.md` materials and source maps are included with the release so that readers can inspect the mathematical statements and their changes alongside the PDFs.