PAPER V

Multiresolution Signal Calculus

Exact coordinates, singular packets, and stable reconstruction

Read the exact declarations, follow the signal models they realize, and separate preserved kernel evidence from current source and computation checks.

Komorebi P.

Lean and evidence guide • 20 October 2026

How to use this volume

A guide to the actual formal development. 20 scoped manuscript results. 225 theorem targets. 23 proof modules. The eight tutorial chapters explain the route from literal samples to Hilbert operators, finite filters, packet synthesis, adaptive trees, and projective actions. The complete ledgers then give exact declarations and manuscript locators.

The full traditional edition also contains 63 broader prose proof records. Their separate inventory is a reading aid and adds no formal coverage. The two edge recovery conjectures remain open.

This edition preserves 52 exact Lean excerpts and reconciles source bytes, declaration locations, dependency pins, and the saved axiom audit. Kernel execution evidence is inherited from the v1.1.0 release dated 6 September 2026. No new Lean build was run for this companion edition.

Contents

1 How to read the formal companion 1

2 Partitions, signed addresses, and the Hilbert bridge 3

3 FIR transfer and faithful perfect reconstruction 7

4 Haar, lifting, and finite depth packet synthesis 11

5 Known edges and measurable perturbations 14

6 Complete prunings, global selection, and stability 16

7 Formal cyclic products and projective Smith actions 22

8 Trust boundaries and reproduction 26

9 The complete twenty-result correspondence 30

10 The broader traditional developments 37

11 All 225 theorem targets 41

12 Source architecture and exact dependency pins 56

1 How to read the formal companion

Paper V places several kinds of mathematics in one signal processing language: coordinate partitions, bounded Hilbert space operators, finite filter banks, finite tree optimization, formal series, and projective actions. The formal development keeps their types and hypotheses distinct. Following those distinctions is the most useful way to read its proofs.

A pointwise partition identity needs little structure. An energy theorem needs a normed space with the intended norm. A transform theorem needs convergence. An optimal finite pruning needs a finite family and an objective. A fractional linear expression needs a chart, while its projective action survives at the chart's poles. Each transition has a named construction or theorem in the source.

> **Follow the model before the theorem name**
>
> Start with the ambient type, then locate the bridge to the intended signal model. A coordinate identity, a bounded operator theorem, and an analytic transform law have different inputs. The exact excerpt shows where those inputs enter the declaration.

1.1 What the completed contract contains

The selected release, `1.1.0-expanded-lean-checked`, records twenty scoped theorem-like manuscript results with explicit proofs and 225 supporting theorem targets. The inherited namespace audit reports 827 declarations. These are different denominators: the 827 includes definitions and compiler-generated declarations, and the 225 includes supporting results rather than 225 independent new mathematical claims.

The formal core is the expanded manuscript preserved as Part I of the final traditional edition. Part II preserves the broader Annals development. Its additional proof records are prose mathematics. They do not enter the twenty-result formalization contract. Section 10 gives a separate crosswalk.

The two edge recovery conjectures remain open. Known-edge cancellation is established, measurable-mask perturbation is established, and finite adaptive selection is established within its hypotheses. Those results do not provide an estimator with the full statistical and approximation guarantees requested by the conjectures.

1.2 Evidence available in this edition

Evidence	Meaning
Source reconciliation	Current hashes, 225 declaration locations, 225 audit directives, and 225 inherited axiom-output records agree.
Historical Lean execution	The September 6 UTC release records the pinned build and namespace audit passing. The raw logs are preserved.
Manual correspondence	Manuscript statements have a reviewed, explicit relationship to formal declarations. Lean does not parse or certify the English PDF automatically.
Exact model tests	Finite independent models exercise indexing, masks, trees, lifting, filters, and boundary cases. They are computational evidence on their tested ranges.

Evidence	Meaning
Companion numerics	Signal experiments and figure data are empirical demonstrations. Their floating-point implementation is outside the Lean theorem claim.

The guide's current build performs no Lean compilation. Reproduction commands appear in Section 8, with the missing-tool case treated as a failure to establish a fresh kernel result. The final package's verification records distinguish current reruns from inherited evidence.

1.3 Read a theorem as a typed contract

The name after `theorem` is a reusable declaration. Parenthesized arguments are explicit inputs. Braced arguments are generally inferred. A hypothesis such as `(hN : 0 < N)` is an actual input to the theorem. A typeclass argument such as `[AddCommMonoid E]` specifies the algebraic laws available to its proof.

The expression after the colon is the proposition. The proof after `:= by` constructs evidence of that proposition. Common commands here have small, specific jobs: `ext` reduces equality to coordinates, `rw` rewrites using an equality, `simp` uses declared simplification rules, `ring` normalizes polynomial identities, `omega` handles integer arithmetic, and `linarith` combines linear inequalities. Their successful use belongs to the recorded compilation, not to a textual count of tactic names.

A definition marked `noncomputable` may use classical choice or real-number constructions that do not give executable numerical code. Its theorems can still be checked by the kernel. In particular, the pruning evaluator over exact real costs specifies a mathematical algorithm and a count of unit operations. Lean evaluation speed is outside that count.

1.4 Four kinds of signal space

Object	Lean representation	What it provides
Arbitrary sequence	`ℤ → K`	Every integer sample exists. No norm or convergence is assumed.
Finite filter	`ℤ →₀ K`	A finitely supported coefficient function. Convolution sums are finite.
Square-summable signal	`lp (fun _ : I => ℂ) 2`	The joint ℓ^2 norm, infinite norm-convergent expansions, bounded maps and adjoints.
Formal series	`R⟦X⟧`	Coefficients with formal addition and Cauchy multiplication. Analytic convergence is not part of the type.

The same informal word "packet" appears in all four settings. The guide always identifies the ambient type before using an identity across them.

1.5 Recommended reading routes

For the signal processing route, read integer addresses, the Hilbert bridge, FIR transfer, faithful filter banks, and Haar packets. For the adaptive route, read known edges, mask perturbations, complete prunings, the dynamic program, and its stability envelope. For the algebraic route, read formal cyclic products, homogeneous Smith actions, and CRT masks.

Every displayed code excerpt is copied from an immutable source declaration. The raw excerpt, its source line range, and its SHA256 hash accompany the PDF. Excerpts rely on the imports, namespaces, variables, and definitions in their source files. They are not advertised as independently compilable snippets. The complete project remains the authoritative executable object.

2 Partitions, signed addresses, and the Hilbert bridge

2.1 A packet first means a coordinate mask

Take an index set I, a label set R, and a phase map $\phi : I \to R$. The packet with label r keeps coordinate i exactly when $\phi(i) = r$:

$$(P_r x)(i) = \begin{cases} x(i), & \phi(i) = r, \\ 0, & \phi(i) \neq r. \end{cases}$$

The definition requires a zero in the value type and a decidable label equality. It does not require a Fourier transform, a measure, or an inner product.

Source: `KomorebiPaperV/Packets.lean`.

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

Applying the same mask twice changes nothing. Distinct labels cannot both be retained at the same coordinate. If the label type is finite and values can be added commutatively, the finite sum of packets returns the original value. At any one coordinate exactly one summand survives.

Source: `KomorebiPaperV/Packets.lean`.

```
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]
```

The proof uses `Finset.sum_eq_single` to isolate the surviving label. This simple mechanism is reused after stronger ambient structures are introduced.

2.2 Signed polyphase coordinates are an equivalence

For $N > 0$, every integer has the unique decomposition

$$n = Nq + r, \qquad 0 \le r < N, \quad q \in \mathbb{Z}.$$

The integer quotient is Euclidean: for $N = 3$ and $n = -1$, the address is $(r, q) = (2, -1)$, not a negative residue. Lean represents the bounded residue by `Fin N`. Its stored value has a proof that it is smaller than N.

Source: `KomorebiPaperV/IntegerPolyphase.lean`.

```
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
    · exact quotient_address N hN p.1 p.2
```

The two inverse laws are mathematical content. They ensure both that analyzing then synthesizing recovers a signal and that synthesizing then analyzing recovers an arbitrary channel family. Merely checking the first direction would not establish that all channel coordinates are reachable.

Writing $x_r[q] = x[Nq + r]$, the pointwise maps are

$$Ax(r, q) = x[Nq + r], \qquad Sc(n) = c(n \bmod N, \lfloor n/N \rfloor).$$

These are meaningful for any value type. Energy is a later statement.

2.3 The two units of delay

A delay by d original samples is $(D_d x)[n] = x[n - d]$. A delay by d inside each channel is an original delay by Nd. For a general original delay, the residue changes and a signed carry enters the channel coordinate:

$$(AD_d x)_r[q] = x_{(r-d) \bmod N}\left[q + \left\lfloor \frac{r-d}{N} \right\rfloor\right].$$

Source: `KomorebiPaperV/IntegerPolyphase.lean`.

```
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 $N = 2$ and a one-sample delay, the new even channel is the old odd channel delayed once, while the new odd channel is the old even channel at the same channel index:

$$(AD_1x)_0[q] = x_1[q-1], \qquad (AD_1x)_1[q] = x_0[q].$$

For a two-sample delay both channels are delayed once. This is why a filter-bank identity with channel delay d reconstructs $x[n - Nd]$.

2.4 Square summability is built into the type

Source: `KomorebiPaperV/HilbertPackets.lean`.

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

The bounded projector is obtained by first proving that masking cannot increase the norm. The proof works coordinate by coordinate and invokes monotonicity of the ℓ^2 norm.

Source: `KomorebiPaperV/HilbertPackets.lean`.

```
theorem projection_norm_le [DecidableEq R] (phase : I → R) (r : R)
    (x : Signal I) : ‖projection phase r x‖ ≤ ‖x‖ := by
  apply lp.norm_mono (by norm_num)
  intro i
  by_cases h : phase i = r <;> simp [projection_apply, h]
```

The same mask is made into a linear map and then promoted to a continuous linear map with bound one.

Source: `KomorebiPaperV/HilbertPackets.lean`.

```
noncomputable def projector [DecidableEq R] (phase : I → R) (r : R) :
    Signal I →L[ℂ] Signal I :=
  (projectionLinear phase r).mkContinuous 1 (by
    intro x
    change ‖projection phase r x‖ ≤ 1 * ‖x‖
    simpa using projection_norm_le phase r x)
```

The notation `→L[ℂ]` records continuous complex linearity. Idempotence alone would not establish boundedness, symmetry, or orthogonality. The file proves those separately. With finitely many labels, the sum is the identity as a continuous linear operator, not just a formula on finite test vectors.

2.5 The channel norm is the joint norm

The Hilbert space analysis target is $\ell^2(\mathrm{Fin}\, N \times \mathbb{Z})$:

$$\|Ax\|^2 = \sum_{r=0}^{N-1} \sum_{q \in \mathbb{Z}} |x[Nq + r]|^2 = \|x\|^2.$$

It is not an unqualified function space of channels carrying a supremum norm. The source packages the address equivalence into a complex linear isometric equivalence.

Source: KomorebiPaperV/HilbertPackets.lean.

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

Its inverse laws are inherited from that equivalence, and its energy law follows from norm preservation. The proof of reindexing uses summability over the bijective address map. This is the actual bridge from arbitrary pointwise coordinates to the Hilbert statement.

2.6 Impulse expansion means norm convergence

The impulse at i with amplitude x_i is represented by `lp.single 2 i (x i)`. The full theorem is a `HasSum` statement in the ℓ^2 space.

Source: KomorebiPaperV/HilbertPackets.lean.

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

Thus the series of impulses converges to x in the ambient norm. The finite support version is an exact finite sum. The shift is an isometric equivalence with formula $S_d x[n] = x[n-d]$, and shifting the impulse at zero gives the impulse at d. Density of the impulse basis yields the dense shift-generated span.

2.7 Refinement uses divisibility

If $N \mid M$, reducing modulo M and then modulo N agrees with reducing directly modulo N. Consequently one coarse residue projector is the sum of the fine residue projectors lying above it.

Source: KomorebiPaperV/HilbertPackets.lean.

```
theorem residue_projector_refinement (N M : ℕ) (hN : 0 < N) (hM : 0 < M)
    (hNM : N ∣ M) (r : Fin N) :
    projector (IntegerPolyphase.phase N hN) r =
      ∑ s : Fin M, if IntegerPolyphase.phase N hN (s.val : ℤ) = r
        then projector (IntegerPolyphase.phase M hM) s else 0 := by
  have hc : (fun j : ℤ => IntegerPolyphase.phase N hN
      ((IntegerPolyphase.phase M hM j).val : ℤ)) = IntegerPolyphase.phase N hN := by
    funext j
    apply Fin.ext
    change (((IntegerPolyphase.phase M hM j).val : ℤ) % (N : ℤ)).toNat =
      (j % (N : ℤ)).toNat
    rw [IntegerPolyphase.phase_val, IntegerPolyphase.emod_coarsen N M hNM]
  have h := projector_refinement (IntegerPolyphase.phase M hM)
    (fun s : Fin M => IntegerPolyphase.phase N hN (s.val : ℤ)) r
  simpa only [Function.comp_def, hc] using h
```

For example, the even packet modulo 2 is the sum of packets 0, 2, and 4 modulo 6. The theorem is an equality of bounded operators on the full complex ℓ^2 space. The hypotheses $N > 0$, $M > 0$, and $N \mid M$ remain explicit. A relation between arbitrary unrelated moduli would require a different construction.

3 FIR transfer and faithful perfect reconstruction

3.1 Finite support makes time domain transfer unconditional

An FIR filter is a finitely supported function $h : \mathbb{Z} \to K$. Negative tap addresses are allowed, so the formal theorem includes advances as well as delays. For an arbitrary bilateral sequence,

$$(h * x)[n] = \sum_{t \in \operatorname{supp} h} h[t]x[n-t].$$

No convergence hypothesis is needed because this sum is finite.

Source: `KomorebiPaperV/FIRTransfer.lean`.

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

Fix output phase k and input phase s. The tap carrying an input channel delay m is

$$t = Nm + k - s, \qquad h_{k,s}[m] = h[Nm + k - s].$$

The address identity is

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

It says exactly which original tap contributes to which channel and delay. The source's `tapEquiv` proves the relevant reindexing is bijective, then partitions the finite support by input phase. It does not enlarge the finite sum to an arbitrary infinite series.

Source: `KomorebiPaperV/FIRTransfer.lean`.

```
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) = _
  rw [fir_reindex N hN h x k n]
  calc
    _ = ∑ s : Fin N, ∑ p ∈ (transferSupport N hN h k).filter (fun p => p.1 = s),
        kernel N h k p.1 p.2 * analysis N x p.1 (n - p.2) :=
      (Finset.sum_fiberwise (transferSupport N hN h k) Prod.fst
        (fun p => kernel N h k p.1 p.2 * analysis N x p.1 (n - p.2))).symm
    _ = _ := by
      apply Finset.sum_congr rfl
      intro s _
      exact sum_pair_fiber (transferSupport N hN h k) s
        (fun r m => kernel N h k r m * analysis N x r (n - m))
```

3.2 A two channel calculation

Three taps and their channel delays

Take $h[0] = a$, $h[1] = b$, $h[2] = c$, with all other taps zero. For $N = 2$ the four channel kernels are

$$y_0[n] = ax_0[n] + cx_0[n-1] + bx_1[n-1],$$
$$y_1[n] = bx_0[n] + ax_1[n] + cx_1[n-1].$$

The tap at one has different channel delays in the two off diagonal positions. In the transform convention used by the paper, the transfer matrix is

$$H(z) = \begin{pmatrix} a + cz^{-1} & bz^{-1} \\ b & a + cz^{-1} \end{pmatrix}.$$

The time domain equations hold for every sequence. The transform equation $Y(z) = H(z)X(z)$ has the additional analytic hypotheses described next.

3.3 A transform is a convergent weighted sum

The source defines a totalized `tsum` expression, but the meaningful convergence predicate is `HasTransform`.

Source: `KomorebiPaperV/ZTransform.lean`.

```
def HasTransform (x : ℤ → ℂ) (z X : ℂ) : Prop := HasSum (fun n => x n * z ^ (-n)) X
```

It states that the bilateral weighted series $\sum_n x[n]z^{-n}$ has the specified sum. The delay theorem requires both a nonzero transform parameter and this convergence input.

Source: `KomorebiPaperV/ZTransform.lean`.

```
theorem delay_transform (x : ℤ → ℂ) (z X : ℂ) (hz : z ≠ 0)
    (h : HasTransform x z X) (d : ℤ) :
    HasTransform (fun n => x (n-d)) z (z ^ (-d) * X) := by
  have hh := h.mul_left (z ^ (-d))
  have he : (fun n : ℤ => z ^ (-d) * (x n * z ^ (-n))) =
      (fun n : ℤ => x (n-d) * z ^ (-n)) ∘ Equiv.addRight d := by
    funext n
    change z ^ (-d) * (x n * z ^ (-n)) = x (n+d-d) * z ^ (-(n+d))
    rw [add_sub_cancel_right, neg_add, zpow_add₀ hz]
    ring
  rw [he] at hh
  exact (Equiv.addRight d).hasSum_iff.mp hh
```

Finitely many delayed inputs can then be combined using finite sums of `HasSum` proofs. The matrix version assumes weighted summability separately for every input channel.

Source: KomorebiPaperV/ZTransform.lean.

```
theorem fir_matrix_transform (N : ℕ) (hN : 0 < N) (h : ℤ →₀ ℂ) (x : ℤ → ℂ)
    (z : ℂ) (hz : z ≠ 0)
    (hx : ∀ s, Summable (fun n => analysis N x s n * z ^ (-n))) :
    (fun k => transform (analysis N (fir h x) k) z) =
      (transferMatrix N hN h z).mulVec (fun s => transform (analysis N x s) z) := by
  funext k
  exact (fir_channel_transform N hN h x z hz _ (fun s => (hx s).hasSum) k).tsum_eq
```

An arbitrary ℓ^2 signal need not have an absolutely convergent bilateral transform at a chosen complex z. An L^2 Fourier boundary-value interpretation on the unit circle is a separate analytic interpretation. Off the unit circle, the displayed weighted summability conditions must be supplied. Neither the definition of `tsum` nor an FIR input filter removes them.

3.4 Laurent polynomials act as finite filters

The type `AddMonoidAlgebra ℂ ℤ` represents finite Laurent coefficients indexed by integer delays. Its multiplication is finite convolution. A monomial at index d acts by $x[n] \mapsto x[n-d]$, with transform symbol z^{-d}. The formal action is an algebra homomorphism into complex linear endomorphisms of all bilateral sequences.

Source: KomorebiPaperV/FilterBank.lean.

```
noncomputable def act : Laurent →ₐ[ℂ] Module.End ℂ Signal :=
  AddMonoidAlgebra.lift ℂ (Module.End ℂ Signal) ℤ shiftHom
```

The source proves `act_is_fir`, making the abstract algebra homomorphism coincide with the literal finite filter sum. It then defines a matrix bank by summing the action of each matrix entry over the input channels. The theorem `bank_mul` identifies matrix multiplication with composition of banks in the correct order.

3.5 Why the matrix criterion is an equivalence

The reverse direction of a perfect-reconstruction criterion needs faithfulness. If two matrix banks have the same action on every sequence, why must their Laurent coefficients agree? The source feeds a unit impulse into one channel at a time. Reading output channel k at sample n recovers exactly the nth Laurent coefficient of the (k, s) entry.

Source: KomorebiPaperV/FilterBank.lean.

```
theorem bank_faithful (A B : Matrix I I Laurent) :
    (∀ x, bank A x = bank B x) ↔ A = B := by
  constructor
  · intro h
    funext k s
    apply AddMonoidAlgebra.ext
    apply Finsupp.ext
    intro n
    have he := congrArg (fun y : I → Signal => y k n) (h (channelImpulse s))
```

```
    simpa only [bank_impulse] using he
  · rintro rfl
    intro x
    rfl
```

This eliminates a possible gap between an operator statement and a formal matrix identity. It relies on equality on every input, rather than agreement on a bounded collection of numerical examples.

3.6 Gain and delay

The target matrix is diagonal, with each diagonal entry the monomial of amplitude c at delay d. The theorem gives the equivalence between matrix equality and reconstruction of every channel sequence.

Source: `KomorebiPaperV/FilterBank.lean`.

```
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]
  constructor
  · intro h x
    funext k n
    rw [bank_mul, gain_delay_action]
    exact h x k n
  · intro h x k n
    have he := congrArg (fun y : I → Signal => y k n) (h x)
    simpa only [bank_mul, gain_delay_action] using he
```

The formula allows any complex c. To interpret the result as invertible reconstruction up to gain, one additionally requires $c \neq 0$. At $c = 0$, the theorem remains a valid zero-output identity, but it is not an invertibility statement.

Returning from channel coordinates to original coordinates yields the literal sample formula.

Source: `KomorebiPaperV/FilterBank.lean`.

```
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
  rw [hr]
  congr 1
  unfold KomorebiPaperV.IntegerPolyphase.analysis
  congr 1
  have ha := KomorebiPaperV.IntegerPolyphase.address_phase_quotient N hN n
  dsimp [KomorebiPaperV.IntegerPolyphase.address] at ha ⊢
  linarith
```

This result is independent of numerical transform convergence. The transform representation of the bank is a further theorem with `HasTransform` hypotheses. The distinction is practical: an exact finite

reconstruction identity can be checked algebraically even when a particular signal's off-circle transform does not exist.

4 Haar, lifting, and finite depth packet synthesis

4.1 Interleaving is part of the theorem

For a bilateral signal, write $e[n] = x[2n]$ and $o[n] = x[2n+1]$. The source first proves that the squared norms of these two channels sum to the original squared norm. It then forms the normalized Haar channels

$$a[n] = \frac{e[n] + o[n]}{\sqrt{2}}, \qquad d[n] = \frac{e[n] - o[n]}{\sqrt{2}}.$$

The sample formula establishes that the abstract construction really acts on these literal interleaved samples.

Source: `KomorebiPaperV/HaarChannels.lean`.

```
theorem haar_low_samples (x : Signal ℤ) (n : ℤ) :
    analysisLow x n = ((Real.sqrt 2 : ℝ) : ℂ)⁻¹ * (x (2*n) + x (2*n+1)) := by
  rw [analysisLow, low_apply, even_apply, odd_apply]
```

The energy identity combines the Hilbert space parallelogram law with the joint channel norm.

Source: `KomorebiPaperV/HaarChannels.lean`.

```
theorem haar_energy (x : Signal ℤ) :
    ‖x‖ ^ 2 = ‖analysisLow x‖ ^ 2 + ‖analysisHigh x‖ ^ 2 := by
  rw [analysisLow, analysisHigh, energy, interleaved_energy]
```

This theorem holds for complex ℓ^2 signals, not only finite real arrays. A finite numerical check can expose normalization mistakes but cannot replace the infinite-space proof.

4.2 Lifting is a different normalization

The lifting analysis sends (e, o) to (s, d) with

$$d = o - e, \qquad s = e + \frac{d}{2} = \frac{e + o}{2}.$$

The inverse is $e = s - d/2$, $o = d + e$. Its determinant is one. The theorem is stated over a characteristic zero field, which makes division by two meaningful in the required way.

Source: `KomorebiPaperV/Lifting.lean`.

```
theorem lifting_synthesis_analysis (x : K × K) :
    liftingSynthesis (liftingAnalysis x) = x := by
  apply Prod.ext <;> dsimp [liftingAnalysis, liftingSynthesis] <;> ring
```

The lifting pair is invertible but is not the normalized Haar pair. For example, $(e, o) = (2, 4)$ gives lifting coordinates $(s, d) = (3, 2)$, whose squared Euclidean norm is 13, whereas the input norm squared is 20. Normalized Haar gives $(3\sqrt{2}, -\sqrt{2})$ with squared norm 20 under the source's high-channel sign convention. Invertibility and isometry are separate claims.

4.3 Flat finite signals have both inverse laws

The finite transform file builds a pairing equivalence between a flat vector of length $2N$ and N pairs. It transports lifting through that equivalence. The result is a theorem about the actual flattened signal, not an unconnected pairwise calculation.

Source: `KomorebiPaperV/FiniteTransforms.lean`.

```
theorem unlift_lift_flat (n : ℕ) (x : Fin (2 * n) → K) :
    unliftFlat n (liftFlat n x) = x := by
  simp only [unliftFlat, liftFlat, flat_analysis_synthesis,
    KomorebiPaperV.lifting_synthesis_analysis, flat_synthesis_analysis]
```

The companion direction, `lift_unlift_flat`, establishes reconstruction of arbitrary transformed coordinates as well. The finite Haar statements similarly include pair energy, finite energy, and a flat involution. These finite constructions are useful for matching array implementations, while the Hilbert files establish the infinite ℓ^2 statements.

4.4 A packet is an ordered composition

At depth zero there is one empty word and its operator is the identity. A depth-$n+1$ word consists of a first branch and a remaining word. The first branch acts on the signal before the remaining packet.

Source: `KomorebiPaperV/WaveletPackets.lean`.

```
noncomputable def packet : (n : ℕ) → Word n → H →L[ℂ] H
  | 0, _ => ContinuousLinearMap.id ℂ H
  | n+1, (b,w) => (packet n w).comp (childOp b)
```

Here `Word n` is a recursively finite type. Each child map is a bounded continuous linear operator. The low and high norm bounds come from the two channel energy identity, so their adjoints are legitimate Hilbert adjoints.

4.5 Induction preserves total energy

The proof at depth $n+1$ first sums over the remaining word for each child. The induction hypothesis reduces that sum to the child's norm squared. Summing over the two children returns the parent's norm squared.

Source: `KomorebiPaperV/WaveletPackets.lean`.

```
theorem packet_energy (n : ℕ) (x : H) : ∑ w : Word n, ‖packet n w x‖ ^ 2 = ‖x‖ ^ 2 := by
  induction n generalizing x with
  | zero =>
    change (∑ w : Unit, ‖x‖ ^ 2) = _
    simp
  | succ n ih =>
    change (∑ w : Fin 2 × Word n, ‖(packet n w.2) (childOp w.1 x)‖ ^ 2) = _
    rw [Fintype.sum_prod_type]
    simp_rw [ih]
    exact child_energy x
```

For depth two, this is the four-packet identity

$$\|W_{00}x\|^2 + \|W_{01}x\|^2 + \|W_{10}x\|^2 + \|W_{11}x\|^2 = \|x\|^2.$$

The theorem quantifies over every natural depth. Each individual depth has a finite packet family. It is not a theorem about convergence of an infinite depth packet expansion.

4.6 Energy gives an operator reconstruction identity

For finitely many bounded maps W_i on the complex Hilbert space, an energy identity valid for every x implies

$$\sum_i W_i^* W_i = I.$$

The formal proof uses inner product extensionality, the adjoint relation, and complex polarization through the library theorem it invokes.

Source: KomorebiPaperV/WaveletPackets.lean.

```
theorem energy_implies_adjoint_identity {I : Type*} [Fintype I]
    (W : I → H →L[ℂ] H) (h : ∀ x, ∑ i, ‖W i x‖ ^ 2 = ‖x‖ ^ 2) :
    (∑ i, (W i).adjoint.comp (W i)) = ContinuousLinearMap.id ℂ H := by
  have he : (∑ i, (W i).adjoint.comp (W i)).toLinearMap = LinearMap.id := by
    apply (ext_inner_map _ _).mp
    intro x
    change inner ℂ ((∑ i, (W i).adjoint.comp (W i)) x) x = inner ℂ x x
    simp only [sum_apply, ContinuousLinearMap.comp_apply, sum_inner,
      ContinuousLinearMap.adjoint_inner_left, inner_self_eq_norm_sq_to_K]
    exact_mod_cast h x
  apply ContinuousLinearMap.ext
  intro x
  exact LinearMap.congr_fun he x
```

This is a genuine adjoint identity. Writing a reconstruction map with a similar-looking formula would not suffice without showing that it is the adjoint of the analysis map.

Applying the resulting operator equality to a signal gives the source level synthesis theorem.

Source: KomorebiPaperV/WaveletPackets.lean.

```
theorem packet_reconstruction (n : ℕ) (x : H) :
    (∑ w : Word n, (packet n w).adjoint (packet n w x)) = x := by
  have h := congrArg (fun T : H →L[ℂ] H => T x) (packet_operator_reconstruction n)
  simpa using h
```

The proof does not need every W_i to be invertible. It needs the collective energy identity. The family can divide information among its channels while the sum of adjoint synthesis terms recovers the original signal exactly.

5 Known edges and measurable perturbations

5.1 Exact retention separates the error algebraically

Write $f = s + j$, where s is the background and j is a known jump packet. If the reconstruction is $Qs + j$, the retained jump cancels:

$$f - (Qs + j) = s - Qs.$$

Source: `KomorebiPaperV/KnownEdge.lean`.

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

The cancellation theorem needs only an additive commutative group and an arbitrary function Q. It does not require Q to be linear. This is sufficient for cancellation because Q is applied to s before the same j is restored. A separate claim that commutes Q with addition, or estimates an operator commutator, would need the appropriate linearity hypotheses.

Taking norms yields exact equality of the full reconstruction error with the background error. A finite sum of known packets cancels in the same way. None of these statements estimates an unknown edge or proves that a data driven algorithm finds the correct mask.

5.2 Estimated packets have two errors

For $j = a\mathbf{1}_S$ and $\hat{j} = b\mathbf{1}_T$,

$$j - \hat{j} = (a - b)\mathbf{1}_S + b(\mathbf{1}_S - \mathbf{1}_T).$$

The first term changes the amplitude on the original mask. The second changes the mask while holding the estimated amplitude fixed. The symmetric difference $S \triangle T$ contains exactly the points where the two masks disagree.

For measurable sets and $1 \le p < \infty$, the resulting bound is

$$\|a\mathbf{1}_S - b\mathbf{1}_T\|_p \le |a - b|\mu(S)^{1/p} + |b|\mu(S \triangle T)^{1/p}.$$

The formal theorem lives first in the extended nonnegative reals, so infinite measures are handled without silently converting infinity to an ordinary real number.

Source: `KomorebiPaperV/MaskPerturbation.lean`.

```
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 Δ T) ^ (1 / p.toReal) := by
  have hp0 : p ≠ 0 := ne_of_gt (lt_of_lt_of_le (by norm_num) hp)
  rw [jump_error_decomposition]
  calc
    _ ≤ eLpNorm (jump S (a-b)) p μ + eLpNorm (jump S b-jump T b) p μ :=
      eLpNorm_add_le (jump_measurable S hS (a-b) μ)
        ((jump_measurable S hS b μ).sub (jump_measurable T hT b μ)) hp
```

```
    _ = _ := by
      rw [symmetric_difference_norm μ S T hS hT p hp0 hpTop,
        jump, eLpNorm_indicator_const hS hp0 hpTop]
```

The notation `ENNReal` permits ∞, and `eLpNorm` is an extended seminorm. The hypotheses `hp : 1 ≤ p` and `hpTop : p ≠ ∞` govern the triangle inequality and finite-exponent formula. A finite real valued corollary requires the displayed terms to be finite, for example through finite mask and symmetric-difference measures.

5.3 A concrete perturbation budget

An amplitude and mask budget

Suppose $p = 2$, $\mu(S) = 1$, $\mu(S \triangle T) = 0.01$, $a = 2$, and $b = 1.9$. The bound is

$$0.1\sqrt{1} + 1.9\sqrt{0.01} = 0.29.$$

If the background approximation error is at most 0.04, the full error is at most 0.33 by the additional triangle inequality in `full_signal_perturbation`. The numbers illustrate a deterministic budget conditional on the amplitude, mask, and background estimates. A statistical procedure achieving those estimates requires additional work.

5.4 Pointwise and essential supremum

The source first proves a bound at every point, then transfers it to the essential supremum seminorm.

Source: `KomorebiPaperV/MaskPerturbation.lean`.

```
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 Δ 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 displayed bound uses whether $S \triangle T$ is empty as an actual set. An essential-supremum argument may ignore a null set, but a pointwise supremum cannot. This particular theorem remains a valid, sometimes coarse, bound when the sets differ only on a null set. Its statement should not be silently replaced by an assertion that pointwise and almost-everywhere equality coincide.

5.5 The open obligations remain identifiable

The stable edge-packet recovery conjecture asks for specified signal classes, contrast and separation assumptions, and a noise model under which a multiscale estimator controls both mask error and amplitude error at rates separated from smooth approximation. The formal perturbation bound explains how such estimates would enter reconstruction. It does not construct that estimator or prove those rates.

The edge-packet approximation-class conjecture asks for natural function spaces whose norms distinguish smooth approximation, boundary geometry, packet amplitude, and packet identification. The

current declarations do not define and close that proposed theory. Infinite-tree consistency, minimax claims, and optimal Smith sampling also remain outside the twenty-result formal scope.

These boundaries matter when reading Part II. Its broader deterministic and geometric theorems can develop the mathematics without turning every contextual statement into a Lean-certified target. The final traditional record and the separate Part II crosswalk identify that distinction explicitly.

6 Complete prunings, global selection, and stability

6.1 A finite tree carries positive finite branching

The tree type permits a leaf or a branch with $n + 1$ children. Thus every branch has at least one child, and unary branches are included.

Source: `KomorebiPaperV/FiniteAdaptive.lean`.

```
inductive Tree (A : Type*) where
  | leaf : A → Tree A
  | branch : (a : A) → (n : ℕ) → (Fin (n + 1) → Tree A) → Tree A
```

The tree itself is a finite inductive object. A cut is another typed inductive object: stop at the current root, or choose one cut in each child. The type prevents selecting an arbitrary collection that fails to cover a terminal path.

Source: `KomorebiPaperV/FiniteAdaptive.lean`.

```
inductive Cut : Tree A → Type _ where
  | stop (t : Tree A) : Cut t
  | split {a : A} {n : ℕ} {children : Fin (n + 1) → Tree A} :
      (∀ i, Cut (children i)) → Cut (.branch a n children)
```

A singleton leaf has only its stop cut. At a unary branch, stopping at the root and descending to its child are distinct cuts even if node labels happen to coincide. Formal nodes carry their positions in the tree, so equal labels do not collapse different nodes.

6.2 Conservation is an explicit hypothesis

For an additive quantity p at each node, conservation says that the parent's value is the sum of the child-root values and that this continues recursively.

Source: `KomorebiPaperV/FiniteAdaptive.lean`.

```
def conserves [AddCommMonoid E] (p : A → E) : Tree A → Prop
  | .leaf _ => True
  | .branch a _ children =>
      p a = ∑ i, p (label (children i)) ∧ ∀ i, conserves p (children i)
```

By induction on the cut, its sum equals the root value. The quantity may be a scalar, a vector, or an operator. The argument uses addition and the conservation identity. It does not derive conservation from an arbitrary tree topology.

For projector valued packets, one supplies both conservation and a root equal to the identity operator. Every complete pruning then reconstructs the signal. This makes selection compatible with exact synthesis: changing the cut changes the representation while preserving the total operator.

6.3 From recursive cuts to covering antichains

The manuscript describes a complete pruning as a node set that meets each root to terminal path exactly once. The source builds a positional node type, terminal paths, and a membership predicate for the nodes selected by a recursive cut. It proves the equivalence with the set based covering-antichain description.

Source: `KomorebiPaperV/CompletePrunings.lean`.

```
theorem complete_iff_covering_antichain {t : FiniteAdaptive.Tree A} (S : Set (Node t)) :
    Complete S ↔ CoveringAntichain S := by
  constructor
  · intro h
    constructor
    · intro p
      obtain ⟨v, hv, _⟩ := h p
      exact ⟨v, hv.2, hv.1⟩
    · intro u hu v hv huv
      obtain ⟨p, hp⟩ := node_on_terminal v
      obtain ⟨w, _, hw⟩ := h p
      exact (hw u ⟨ancestor_onPath huv hp, hu⟩).trans (hw v ⟨hp, hv⟩).symm
  · rintro ⟨hcover, hanti⟩ p
    obtain ⟨v, hv, hp⟩ := hcover p
    refine ⟨v, ⟨hp, hv⟩, ?_⟩
    intro u hu
    rcases path_nodes_comparable hu.1 hp with h | h
    · exact hanti u hu.2 v hv h
    · exact (hanti v hv u hu.2 h).symm
```

The file also proves that every complete set comes from a cut and that the cut is determined by its selected nodes. Together these results give `pruningEquiv`. This is why later theorems apply to every complete covering antichain, not just a preselected list of recursive examples.

6.4 Literal sums over selected nodes

`PruningSums` connects the recursive cut sum with the actual finite sum over the selected node set. Its operator theorem provides both the operator equality and the applied-signal equality.

Source: `KomorebiPaperV/PruningSums.lean`.

```
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 hypotheses `h : conserves p t` and `hr : p (label t) = ...id...` are inputs. There is no assertion that an arbitrary family of packet operators indexed by a tree automatically satisfies them.

6.5 Existence of a minimum allows nonadditive objectives

The manuscript's general criterion has the form

$$J(S) = R(S) + \eta L(S) + \tau K(S),$$

where R, L, and K can depend on the complete pruning as a whole. Finiteness alone ensures a minimum is attained.

Source: `KomorebiPaperV/PruningSums.lean`.

```
theorem objective_minimum (t : FiniteAdaptive.Tree A) (R L K : Pruning t → ℝ) (η τ : ℝ) :
    ∃ S : Pruning t, ∀ T, objective R L K η τ S ≤ objective R L K η τ T :=
  complete_pruning_minimum t _
```

No positivity or additivity of these functions is needed for the finite attainment theorem. Such assumptions can be useful in applications, but the logical reason for attainment here is a nonempty finite search space.

The dynamic program below solves an additive criterion $\sum_{v \in S} c(v)$. The general nonadditive existence result does not imply that the same recurrence efficiently minimizes an arbitrary coupled objective. A user of the recurrence must verify the required additive decomposition.

6.6 Strict gaps require competitors

If S_* is a strict minimizer and there is another pruning, the minimum positive objective difference is attained by a real competitor.

Source: `KomorebiPaperV/PruningSums.lean`.

```
theorem objective_gap_attained (t : FiniteAdaptive.Tree A) (J : Pruning t → ℝ) (S : Pruning t)
    (ho : ∃ T, T ≠ S) (hu : ∀ T, T ≠ S → J S < J T) :
    ∃ T : Pruning t, T ≠ S ∧ 0 < J T-J S ∧ ∀ U, U ≠ S → J T-J S ≤ J U-J S := by
  let : Fintype (Pruning t) := Fintype.ofEquiv (Cut t) (pruningEquiv t)
  classical
  obtain ⟨U,hU⟩ := ho
  obtain ⟨T,hT,hmin⟩ := KomorebiPaperV.finite_minimum (fun T => J T-J S)
    (Finset.univ.erase S) ⟨U, Finset.mem_erase.mpr ⟨hU, Finset.mem_univ _⟩⟩
  have hTS := (Finset.mem_erase.mp hT).1
  exact ⟨T,hTS,sub_pos.mpr (hu T hTS),fun V hV => hmin V
    (Finset.mem_erase.mpr ⟨hV, Finset.mem_univ _⟩)⟩
```

The explicit `ho` hypothesis excludes the empty competitor family. For a singleton pruning space, the selected pruning is unchanged for every objective, so a separate singleton theorem is the natural statement. A tie between distinct prunings gives zero separation and does not satisfy a positive strict-gap hypothesis.

6.7 The bottom up evaluator

For an additive node cost, the optimal value obeys

$$V(v) = \min\left(c(v), \sum_{u \text{ child of } v} V(u)\right).$$

The selector stops at the node when the two alternatives tie. The formal evaluator returns the cut, its cost, and three operation counters together.

Source: `KomorebiPaperV/PruningAlgorithm.lean`.

```
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⟩
```

The child results are introduced once as a family, and their stored costs are summed. The correctness theorem identifies both outputs with the specification: cost equals `value`, and cut equals `best`.

Source: `KomorebiPaperV/PruningAlgorithm.lean`.

```
theorem solve_correct (c : A → ℝ) (t : FiniteAdaptive.Tree A) :
    (solve c t).cost = value c t ∧ (solve c t).cut = best c t := by
  induction t with
  | leaf a => exact ⟨rfl,rfl⟩
  | branch a n ch ih =>
    have hc : (∑ i, (solve c (ch i)).cost) = ∑ i, value c (ch i) :=
      Finset.sum_congr rfl (fun i _ => (ih i).1)
    constructor
    · simp only [solve, value, hc]
    · simp only [solve, best, hc]
      split_ifs
      · rfl
      · congr 1
        funext i
        exact (ih i).2
```

The proof uses tree induction. In a branch, the induction hypotheses identify all cached child values with their specification values. The same comparison then controls both the evaluator and the specification, including equality.

6.8 An explicit small tree

Consider a root r with two leaf children a and b. Its complete prunings are $\{r\}$ and $\{a, b\}$. Set $c(r) = 7$, $c(a) = 2$, and $c(b) = 3$. The split costs 5 and is selected. The gap to the other pruning is 2.

Changing $c(r)$ to 5 creates a tie. The selector now stops at the root by convention. This deterministic choice is defined, but it is not protected by a positive separation margin. An arbitrarily small perturbation can change which pruning has smaller cost. Tie handling and strict stability address different questions.

If $p(r) = p(a) + p(b) = I$, both prunings synthesize exactly. At a tie, two representations can exchange preference while both preserve the full signal through the same synthesis identity.

6.9 What the linear-work count charges

The source proves one visit per node, one comparison per internal node, and one addition per edge. The last identity is expressed as additions plus one equals the node count. Combining them yields

Source: `KomorebiPaperV/PruningAlgorithm.lean`.

```
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
```

Node costs are supplied inputs. Their estimation from data, the cost of exact real arithmetic, output serialization, and the runtime behavior of a particular compiler are not charged by this theorem. The claimed bound is precise for its stated unit operation model. It is not a wall clock performance certificate or a cost bound for a nonadditive objective.

6.10 The exact worst pruning error envelope

Suppose local perturbations obey $|d(v) - c(v)| \le e(v)$. The maximum possible sum of the error weights along a complete pruning has the recurrence

$$E(v) = \max\left(e(v), \sum_{u \text{ child of } v} E(u)\right).$$

Source: `KomorebiPaperV/FiniteAdaptive.lean`.

```
noncomputable def envelope (e : A → ℝ) : Tree A → ℝ
  | .leaf a => e a
  | .branch a _ children => max (e a) (∑ i, envelope e (children i))
```

The source proves both that every cut weight is at most this envelope and that some cut attains it. It is therefore the exact maximum of the additive weights over complete prunings, rather than a loose sum over all nodes.

For the two-leaf example, if $e(r) = 0.3$ and $e(a) = e(b) = 0.2$, then $E(r) = \max(0.3, 0.4) = 0.4$. Both candidate objective values are perturbed by at most 0.4. The original gap is 2, and $2E(r) = 0.8 < 2$, so the selected split is protected.

6.11 Why the margin contains a factor of two

When comparing a competitor with the minimizer, the competitor can move down by the full error budget while the minimizer moves up by the same budget. Their separation can shrink by twice the budget. The antichain theorem preserves the selected pruning under the strict inequality.

Source: `KomorebiPaperV/PruningSums.lean`.

```
theorem antichain_selector_stable (c d e : A → ℝ) (h : ∀ a, |d a-c a| ≤ e a)
    (t : FiniteAdaptive.Tree A) (γ : ℝ)
    (hg : ∀ S : Pruning t, S ≠ selected c t → γ ≤ nodeSum c S-nodeSum c (selected c t))
    (hm : 2*envelope e t < γ) : selected d t = selected c t := by
  apply congrArg (pruningEquiv t)
  apply best_unchanged_of_gap c d e h t γ _ hm
  intro q hq
  have hne : pruningEquiv t q ≠ selected c t := fun he => hq ((pruningEquiv t).injective he)
  simpa only [selected, pruning_nodeSum] using hg (pruningEquiv t q) hne
```

Equality at the boundary is insufficient in general. With an original gap $\gamma = 2$ and budget $E = 1$, the winner can move up by one and the competitor down by one, producing a tie. A deterministic tie rule can then change the selected object. The theorem's strict inequality is part of its content.

6.12 Probability is transferred from a joint event

The probabilistic statement assumes one event on which all required local inequalities hold simultaneously. Deterministic stability then makes that event a subset of the event that the selector is unchanged.

Source: `KomorebiPaperV/PruningAlgorithm.lean`.

```
theorem antichain_probability {Ω : Type*} [MeasurableSpace Ω]
    (μ : MeasureTheory.Measure Ω) (good : Set Ω) (δ : ENNReal)
    (c e : A → ℝ) (d : Ω → A → ℝ) (t : FiniteAdaptive.Tree A) (γ : ℝ)
    (hgood : ∀ ω ∈ good, ∀ a, |d ω a-c a| ≤ e a)
    (hg : ∀ S : Pruning t, S ≠ selected c t → γ ≤ nodeSum c S-nodeSum c (selected c t))
    (hm : 2*envelope e t < γ) (hprob : 1-δ ≤ μ good) :
    1-δ ≤ μ {ω | selected (d ω) t = selected c t} := by
  apply hprob.trans
  apply MeasureTheory.measure_mono
  intro ω hω
  exact antichain_selector_stable c (d ω) e (hgood ω hω) t γ hg hm
```

Separate marginal statements for each node do not automatically give the same probability for their intersection. One needs a supplied joint event or an additional argument, such as a correctly budgeted union bound. No independence assumption or tail concentration theorem is inferred by the source.

The theorem is written for a measure with an explicit lower bound `1-δ ≤ μ good`. Lean's measure notation also evaluates arbitrary sets through the associated outer measure, so the set-inclusion proof does

not itself establish event measurability. For ordinary success probabilities, use a probability measure, measurable random node costs, and a measurable joint good event. The selector on a finite tree is then determined by finitely many measurable comparisons, making its success event measurable. Finite tree structure alone cannot make arbitrary random cost functions measurable. The final traditional edition makes these probability hypotheses explicit. In Part II, consistency statements based only on probabilities tending to one are interpreted as convergence in probability. Almost-sure conclusions need additional hypotheses and an argument such as summable failure probabilities.

7 Formal cyclic products and projective Smith actions

7.1 A formal residue projector filters coefficients

For a formal power series $f = \sum_{n \geq 0} f_n X^n$, the jth cyclic projector retains the degrees congruent to j modulo N.

Source: `KomorebiPaperV/CyclicSeries.lean`.

```
def cyclicProjector (N j : ℕ) (f : R⟦X⟧) : R⟦X⟧ :=
  PowerSeries.mk fun n => if n % N = j % N then PowerSeries.coeff n f else 0
```

The ambient coefficient ring is a commutative semiring. There is no evaluation point and no convergence assumption in this definition. Multiplication is the formal Cauchy product, whose individual coefficients are finite sums.

The projector laws express idempotence, disjointness of different residue classes, and resolution of the identity for $N > 0$. A pure packet in residue i multiplied by one in residue j lies in residue $i+j$ modulo N.

7.2 The complete cyclic product law

Decompose both factors into packets, distribute multiplication, and retain only pairs whose residue labels add to the requested output label.

Source: `KomorebiPaperV/CyclicSeries.lean`.

```
theorem cyclic_product_law {N : ℕ} (hN : 0 < N) (k : ℕ) (f g : R⟦X⟧) :
    cyclicProjector N k (f * g) =
      ∑ i ∈ Finset.range N, ∑ j ∈ Finset.range N,
        if (i + j) % N = k % N then
          cyclicProjector N i f * cyclicProjector N j g
        else 0 := by
  calc
    cyclicProjector N k (f * g) =
        cyclicProjector N k
          ((∑ i ∈ Finset.range N, cyclicProjector N i f) *
            (∑ j ∈ Finset.range N, cyclicProjector N j g)) := by
      rw [sum_cyclicProjector hN, sum_cyclicProjector hN]
    _ = ∑ i ∈ Finset.range N, ∑ j ∈ Finset.range N,
          if (i + j) % N = k % N then
            cyclicProjector N i f * cyclicProjector N j g
          else 0 := by
```

```
      rw [Finset.sum_mul]
      simp_rw [Finset.mul_sum]
      rw [cyclicProjector_sum]
      apply Finset.sum_congr rfl
      intro i hi
      rw [cyclicProjector_sum]
      apply Finset.sum_congr rfl
      intro j hj
      by_cases hlabel : (i + j) % N = k % N
      · simp only [hlabel, ↓reduceIte]
        rw [cyclicProjector_eq_of_mod_eq (i := k) (j := (i + j) % N)
          (by simpa using hlabel.symm)]
        exact cyclicProjector_product_packet N i j f g
      · simp only [hlabel, ↓reduceIte]
        exact cyclicProjector_product_other (R := R) (fun h => hlabel h.symm) f g
```

For $N = 2$, write $f = f_0 + f_1$ and $g = g_0 + g_1$. Then

$$(fg)_0 = f_0 g_0 + f_1 g_1, \qquad (fg)_1 = f_0 g_1 + f_1 g_0.$$

The subscripts label degree parity. They are not independent scalar coefficients. This formula explains nonlinear coupling between packet labels without leaving the formal-series setting.

7.3 Powers include the empty product

The power theorem sums over all q-tuples of residue labels, retaining those whose sum has the requested residue.

Source: `KomorebiPaperV/CyclicSeries.lean`.

```
theorem cyclic_power_law {N : ℕ} (hN : 0 < N) (k q : ℕ) (f : R⟦X⟧) :
    cyclicProjector N k (f ^ q) =
      ∑ i : Fin q → Fin N, if (∑ a, (i a).val) % N = k % N then
        ∏ a, cyclicProjector N (i a).val f else 0 := by
  classical
  have hex : f ^ q = ∑ i : Fin q → Fin N, ∏ a, cyclicProjector N (i a).val f := by
    have h := Fintype.sum_pow (fun i : Fin N => cyclicProjector N i.val f) q
    rw [sum_fin_projector hN] at h
    exact h
  rw [hex, cyclicProjector_sum]
  apply Finset.sum_congr rfl
  intro i _
  exact project_finite_product N k Finset.univ (fun a => (i a).val)
    (fun a => cyclicProjector N (i a).val f)
    (fun a _ => cyclicProjector_idempotent N (i a).val f)
```

At $q = 0$, the type `Fin 0 → Fin N` has one function, the empty assignment. Its sum of labels is zero and its product is one. Thus the formula returns the constant series one for the zero residue and zero otherwise. This boundary case is part of the statement, not an unstated convention added afterward.

The foundational coefficient-projector proofs identify their adaptation from Paper IV in source comments and in the cumulative census. Reusing them in a new application does not make them independent additional discoveries. Analytic root-of-unity formulas require their own convergence domain when formal series are evaluated.

7.4 Use homogeneous coordinates at poles

For a nonzero complex parameter a, the normalized homogeneous Smith action is

$$A_a \begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1+a & 1-a \\ 1-a & 1+a \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}.$$

Source: KomorebiPaperV/SmithAction.lean.

```
noncomputable def action (a : ℂ) (p : ℂ × ℂ) : ℂ × ℂ :=
  (((1 + a) * p.1 + (1 - a) * p.2) / 2,
   ((1 - a) * p.1 + (1 + a) * p.2) / 2)
```

The source proves $A_a A_b = A_{ab}$, $A_1 = I$, and preservation of nonzero pairs when $a \neq 0$. Multiplying both input coordinates by the same nonzero scalar multiplies both output coordinates by that scalar.

A projective point is a nonzero pair modulo this common scaling. The source constructs the equivalence relation explicitly.

Source: KomorebiPaperV/SmithCyclic.lean.

```
noncomputable def pointSetoid : Setoid Point where
  r := same
  iseqv := {
    refl := by intro p; exact ⟨1, one_ne_zero, by simp⟩
    symm := by
      rintro p q ⟨c, hc, hq⟩
      refine ⟨c⁻¹, inv_ne_zero hc, ?_⟩
      rw [hq]
      simp [hc]
    trans := by
      rintro p q r ⟨c, hc, hq⟩ ⟨d, hd, hr⟩
      exact ⟨d*c, mul_ne_zero hd hc, by rw [hr, hq]; simp [mul_assoc]⟩ }
```

The quotient is the full projective line in this coordinate model. It includes $[0 : 1]$, representing zero impedance, and $[1 : 0]$, representing infinity. The action respects scaling, so it descends to the quotient.

7.5 Cyclic composition

The parameter $\omega_N = \exp(2\pi i/N)$ is nonzero and satisfies $\omega_N^N = 1$ for $N > 0$. The resulting projective actions compose by addition of exponents modulo N.

Source: KomorebiPaperV/SmithCyclic.lean.

```
theorem cyclic_composition (N : ℕ) (hN : 0 < N) (k l : ℕ) (p : Projective) :
    cyclicAction N k (cyclicAction N l p) = cyclicAction N ((k+l)%N) p := by
  unfold cyclicAction
  rw [projective_mul]
  have he : omega N ^ k * omega N ^ l = omega N ^ ((k+l)%N) := by
    rw [← pow_add, pow_eq_pow_mod (k+l) (omega_pow N hN)]
  congr 1
```

This source target establishes the modulo-N composition law. It should not be described as a separate formal proof that the action has exact order N unless the required primitivity and nontriviality converse are established as additional formal declarations. The broader traditional treatment contains further action analysis; its prose scope is separately recorded.

7.6 Cayley coordinates explain the action

In the pair coordinates $(u-v, u+v)$, the action multiplies the first coordinate by a and leaves the second fixed.

Source: `KomorebiPaperV/SmithCyclic.lean`.

```
theorem cayley_conjugacy (a : ℂ) (p : ℂ × ℂ) :
    ((action a p).1 - (action a p).2, (action a p).1 + (action a p).2) =
      (a * (p.1-p.2), p.1+p.2) := by
  apply Prod.ext <;> dsimp [action] <;> ring
```

On a finite chart where the denominators are nonzero, this corresponds to rotation or multiplication of the reflection coefficient. The rational impedance formula is

$$z \longmapsto \frac{(1+a)z + (1-a)}{(1-a)z + (1+a)}.$$

A vanishing denominator means the image is the projective point at infinity. Lean's field division is totalized, so a bare identity of division expressions does not by itself assert that a displayed chart exists. The homogeneous action carries the geometry through such poles, while chart composition theorems carry explicit denominator conditions.

7.7 The half-turn swaps zero and infinity

For $N = 2$, $\omega_2 = -1$ and the homogeneous action swaps the two coordinates.

Source: `KomorebiPaperV/SmithCyclic.lean`.

```
theorem two_phase_swap (p : ℂ × ℂ) : action (omega 2 ^ 1) p = (p.2,p.1) := by
  rw [pow_one, omega_two, half_turn]
```

On the finite nonzero chart this gives $z \mapsto z^{-1}$. At $z = 0$ it maps $[0 : 1]$ to $[1 : 0]$. At infinity it does the reverse. The reciprocal formula is therefore a chart description of a globally defined projective transformation, not a rule that discards poles.

7.8 Dyadic and odd masks factor by CRT

If $N = 2^a M$ with M odd, the two factors are coprime. An integer satisfies a residue condition modulo their product precisely when it satisfies both factor conditions. The source proves the corresponding equality of integer-residue masks.

Source: `KomorebiPaperV/CRT.lean`.

```
theorem dyadic_odd_packet_factorization (a d : ℕ) (hd : Odd d)
    (r : ZMod (2 ^ a * d)) (x : ℤ → V) :
    KomorebiPaperV.packet (fun i : ℤ => (i : ZMod (2 ^ a * d))) r x =
      KomorebiPaperV.packet (fun i : ℤ => (i : ZMod (2 ^ a)))
        (ZMod.chineseRemainder (dyadic_odd_coprime a d hd) r).1
        (KomorebiPaperV.packet (fun i : ℤ => (i : ZMod d))
          (ZMod.chineseRemainder (dyadic_odd_coprime a d hd) r).2 x) :=
  residue_mask_factorization _ _ (dyadic_odd_coprime a d hd) r x
```

This is an exact mask factorization. It does not prove that a particular nonuniform sample allocation is optimal, nor does it provide a minimax sampling theorem. It supplies the arithmetic structure on which more specialized sampling constructions can be built.

8 Trust boundaries and reproduction

8.1 What the inherited audit actually checks

The preserved compiler log records 225 explicit theorem targets and 827 namespace declarations. The audit driver first requires each target to be present as a theorem. It then traverses the `KomorebiPaperV` namespace, rejects unsafe declarations, rejects local axiom and opaque declarations, and collects transitive axiom dependencies.

Only `propext`, `Classical.choice`, and `Quot.sound` are permitted. The audit checks the dependencies of the compiled declarations. It does not prove that English prose is a correct paraphrase of those declarations, that a Python implementation matches a Lean definition, or that an application has supplied a theorem's hypotheses.

The current guide build reconciles each target name with its immutable source location and inherited axiom-output record. It verifies pinned source hashes and performs a lexical scan for proof placeholders and prohibited source constructs. These checks protect provenance and locate obvious source-policy violations. They do not replace elaboration, type checking, or a new kernel traversal.

8.2 The two dates of evidence

The release's raw kernel verification timestamp is 6 September 2026 at approximately 03:18 UTC. This monograph companion is dated 9 October 2026. The compiler transcript describes the preserved v1.1.0 sources and their pinned dependencies. Its date remains visible beside the current checks.

This edition freshly ran the twenty exact model tests, comprising 305,933 counted assertions, and all sixteen verifier control methods. It ran the 33 current companion pytest tests and the 27 overlapping historical tests with no skips. These are finite tests of implementations and models, with separate contracts for their tested scope.

All 52 stored floating numerical metrics and eighteen discrete result leaves were reproduced exactly in the available environment. Eighteen companion PDF/PNG figure pairs were regenerated. The broader Annals companion separately passed 11,052 mixed exact and numerical checks across thirteen groups. Six bounded computational manifests were validated against their inputs. The symbolic visual suite passed 181 checks and rejected seven false controls. The extension examples separately passed 38 exact checks and rejected four false controls.

These counts refer to the recorded tests and their declared ranges. Overlapping suites are retained as provenance and are not combined into a theorem denominator. In particular, floating computations and bounded models do not extend the twenty-result formal contract or close either edge recovery conjecture.

No fresh Lean process ran because `lean`, `lake`, and `elan` were unavailable. Current static reconciliation checks the actual source bytes, 225 declaration locations, 225 audit directives, nine dependency revisions, and 225 saved axiom records. It does not run Lean's elaborator or kernel. The complete package's current computational review and per-run manifests record the fresh tests and their limits.

The recovered October 5 edition lacks seventeen verification artifacts, including its `verify_available.py` runner and some report and figure files. The surviving formal sources, September compiler logs, and saved axiom audit are preserved. Current scoped replays have their own manifests and adapters. The October 5 available-checks gate was not rerun or authenticated from a complete historical inventory. The inventory appears in `provenance/HISTORICAL_RECOVERY_GAPS.json`. Missing historical evidence is not recreated by a current passing test.

8.3 Reproduce from a working copy

Use a writable copy of the original release to run its verification entry points, because they write status and log files beside the sources. Keep the packaged original snapshot unchanged as the provenance reference. The following commands start in that copied release root, which contains `VERIFY_PAPER.py`, `formalization`, `companion`, and `verification`.

The retained commands below describe the September formal release entry points. They do not reproduce the missing October 5 available-checks runner. The original formal-release pipeline expects the pinned Lean toolchain, its pinned dependencies, the Python requirements in `companion/requirements.txt`, `pypdf`, pdfLaTeX, and Poppler tools. Once those prerequisites are available, obtain the mathlib cache in the Lean project:

```
cd formalization/lean
lake exe cache get
cd ../..
python3 VERIFY_PAPER.py
```

This full entry point checks the reviewed source hashes, calls the Lean verifier, runs the original companion tests, repeats the optimized static checks, replays numerical experiments, regenerates figure pairs, and rebuilds the selected 39-page source manuscript. It reproduces the original integrated formal-core release, not the later combined two-part traditional PDF.

For the formal source and exact-model subset, use:

```
python3 verification/verify.py
```

This command checks the actual toolchain version, builds the local Lean library, checks pinned dependency revisions and tracked-file cleanliness, and runs `KernelAudit.lean` with warnings treated as errors. If Lake is unavailable, the verifier reports `BLOCKED_NO_LAKE` and exits with code two. It does not turn passing Python tests into a successful Lean result.

For a deliberately static and exact-model run with no Lean claim:

```
python3 -O verification/verify.py --static-only
```

The `STATIC_AND_EXACT_ONLY` status and `lean_kernel_checked=false` are expected for that route. Python optimization is an intentional verifier control; the program's critical requirements are explicit checks rather than removable Python `assert` statements.

8.4 Manual inspection of the formal build

To inspect the principal operations separately, work inside `formalization/lean`:

```
lake env lean --version
lake build KomorebiPaperV
lake env lean -DwarningAsError=true KernelAudit.lean
```

A successful build without the audit is incomplete evidence for the release's declared axiom policy. A successful audit of changed sources is a new result that should be recorded with new source hashes. Do not reuse an old transcript as though it described a modified source tree.

The historical audit's final success line is:

```
PAPER_V_CANDIDATE_TARGET_AUDIT_PASS: 225 theorem targets; 827 namespace declarations
```

The whole transcript, compiler version, source pins, and exit status matter. Searching for this line in a file is provenance reconciliation, not a substitute for executing the audit.

8.5 The historical runtime adapter

The original environment used a narrow executable self-location adapter. Its ordinary process-specific executable path returned a permission error, while `/proc/self/exe` was accessible. The source adapter redirected only the current process's own executable-link query. The bundled documentation records that the distributed compiler and kernel were unchanged.

The retained positive control accepted a valid arithmetic theorem and the negative control rejected an invalid proof with exit code one. These are useful runtime controls, but the full source build and namespace audit remain the decisive historical evidence. The adapter is environment-specific and unnecessary on a normal installation. Toolchain binaries and compiled dependency caches are not included in this package.

8.6 Rebuild this guide

The guide itself needs XeLaTeX and the TeX packages named in `preamble.tex` and `komorebi-monograph.sty`. Its P052, Nimbus Sans, DejaVu Mono, and Latin Modern fonts must be installed externally. The builder resolves them with `kpsewhich`, verifies the staged bytes, and records their hashes. It never installs dependencies or bundles font binaries. From its editable directory:

```
python3 build.py
```

This copies the complete authored guide tree to an isolated directory and compiles the supplied TeX and generated catalog files in four passes. It executes no Lean code and does not regenerate mathematics. The separate `build_guide_data.py` script can regenerate the catalogs when supplied the immutable release root and the traditional paper's final crosswalk. It verifies hashes and rejects mismatched contracts before writing the guide's data products.

The raw excerpt files preserve source text. Their rendering files escape TeX-special characters and route rare mathematical glyphs to a suitable font. The PDF is a reading surface. The full source files retain the actual namespace and typeclass context needed for Lean compilation.

8.7 A theorem-use checklist

Before using a result, identify its ambient type and explicit hypotheses, locate the exact declaration, and check how the application supplies each hypothesis. For an FIR bank, confirm finite Laurent support and delay units. For a transform equality, supply weighted summability. For pruning synthesis, supply conservation. For selector stability, establish the strict gap and joint local-error bound. For a Smith chart, check denominator nonvanishing or use the projective action.

This is the recurring pattern of the formal development: construct the object in the correct space, prove the bridge to the intended model, then use the identity whose hypotheses have actually been supplied.

9 The complete twenty-result correspondence

Each entry gives the scoped formal statement, its principal declarations, and the traditional manuscript locator. The 225-target catalog includes the supporting theorems as well. Principal targets may recur because one synthesis fact supports more than one manuscript statement. These are navigation relationships, not additive theorem counts. Traditional locators use printed Arabic pages. The unnumbered cover and Roman opening pages have separate physical PDF indices, preserved in the companion crosswalk.

9.1 1.1 Impulse seed theorem

Traditional locator. Theorem 1.1, final paper p. 2; `result:01`. **Exact reviewed scope.** Complex l2 impulse expansion, finite-support identity, unitary shifts and dense shift-generated span. `KomorebiPaperV.HilbertPackets.impulse_expansion`

`KomorebiPaperV/HilbertPackets.lean`, line 135.

`KomorebiPaperV.HilbertPackets.finite_impulse_expansion`

`KomorebiPaperV/HilbertPackets.lean`, line 207.

`KomorebiPaperV.HilbertPackets.shift_generated_dense_span`

`KomorebiPaperV/HilbertPackets.lean`, line 203.

`KomorebiPaperV.HilbertPackets.shift_impulse`

`KomorebiPaperV/HilbertPackets.lean`, line 183.

9.2 2.1 Polyphase projector theorem

Traditional locator. Theorem 2.1, final paper p. 3; `result:02`. **Exact reviewed scope.** Bounded orthogonal residue projectors on complex l2. `KomorebiPaperV.HilbertPackets.projector_idempotent`

`KomorebiPaperV/HilbertPackets.lean`, line 56.

`KomorebiPaperV.HilbertPackets.projector_annihilate`

`KomorebiPaperV/HilbertPackets.lean`, line 62.

`KomorebiPaperV.HilbertPackets.projector_complete`

`KomorebiPaperV/HilbertPackets.lean`, line 69.

`KomorebiPaperV.HilbertPackets.projector_symmetric`

`KomorebiPaperV/HilbertPackets.lean`, line 139.

`KomorebiPaperV.HilbertPackets.projector_orthogonal`

`KomorebiPaperV/HilbertPackets.lean`, line 147.

9.3 2.2 Analysis–synthesis identity

Traditional locator. Corollary 2.2, final paper p. 3; `result:03`. **Exact reviewed scope.** Signed integer polyphase linear isometry; channels represented by joint l2 coordinates. `KomorebiPaperV.HilbertPackets.polyphase_synthesis_analysis`

`KomorebiPaperV/HilbertPackets.lean`, line 123.

`KomorebiPaperV.HilbertPackets.polyphase_analysis_synthesis`

`KomorebiPaperV/HilbertPackets.lean`, line 127.

`KomorebiPaperV.HilbertPackets.polyphase_energy`

`KomorebiPaperV/HilbertPackets.lean`, line 131.

9.4 2.3 Exact nested refinement

Traditional locator. Theorem 2.3, final paper p. 5; `result:04`. **Exact reviewed scope.** Literal divisible-modulus bounded projector refinement. `KomorebiPaperV.HilbertPackets.residue_projector_refinement`

`KomorebiPaperV/HilbertPackets.lean`, line 187.

9.5 3.1 Exact FIR polyphase transfer law

Traditional locator. Theorem 3.1, final paper p. 5; `result:05`. **Exact reviewed scope.** Time-domain transfer for every bilateral sequence; channel z-transform identity for z nonzero wherever each weighted input series is summable. `KomorebiPaperV.ZTransform.fir_channel_transform`

`KomorebiPaperV/ZTransform.lean`, line 49.

`KomorebiPaperV.ZTransform.fir_matrix_transform`

`KomorebiPaperV/ZTransform.lean`, line 64.

`KomorebiPaperV.FIRTransfer.fir_polyphase_transfer`

`KomorebiPaperV/FIRTransfer.lean`, line 96.

9.6 4.1 Perfect-reconstruction criterion

Traditional locator. Theorem 4.1, final paper p. 7; `result:06`. **Exact reviewed scope.** Concrete finite Laurent/FIR banks on every bilateral sequence, faithful matrix/operator correspondence, gain and channel delay; original-sample delay is N*d. Arbitrary scalar-matrix criterion is also retained in FiniteTransforms. `KomorebiPaperV.FilterBank.act_is_fir`

`KomorebiPaperV/FilterBank.lean`, line 36.

`KomorebiPaperV.FilterBank.bank_mul`

`KomorebiPaperV/FilterBank.lean`, line 84.

`KomorebiPaperV.FilterBank.bank_faithful`

`KomorebiPaperV/FilterBank.lean`, line 104.

`KomorebiPaperV.FilterBank.perfect_reconstruction_iff`

`KomorebiPaperV/FilterBank.lean`, line 133.

`KomorebiPaperV.FilterBank.bank_transform`

`KomorebiPaperV/FilterBank.lean`, line 147.

`KomorebiPaperV.FilterBank.original_sample_delay`

`KomorebiPaperV/FilterBank.lean`, line 155.

9.7 4.2 Haar energy identity

Traditional locator. Corollary 4.2, final paper p. 8; `result:07`. **Exact reviewed scope.** Literal interleaved input and normalized low/detail channels in complex l2. `KomorebiPaperV.HaarChannels.haar_energy`

`KomorebiPaperV/HaarChannels.lean`, line 40.

`KomorebiPaperV.HaarChannels.haar_low_samples`

`KomorebiPaperV/HaarChannels.lean`, line 44.

`KomorebiPaperV.HaarChannels.haar_high_samples`

`KomorebiPaperV/HaarChannels.lean`, line 48.

9.8 4.3 Exact lifting reconstruction

Traditional locator. Theorem 4.3, final paper p. 10; `result:08`. **Exact reviewed scope.** Pairwise predict/update inverse and determinant one, with both flat finite-signal inverse laws. `KomorebiPaperV.FiniteTransforms.unlift_lift_flat`

`KomorebiPaperV/FiniteTransforms.lean`, line 78.

`KomorebiPaperV.FiniteTransforms.lift_unlift_flat`

`KomorebiPaperV/FiniteTransforms.lean`, line 83.

`KomorebiPaperV.lifting_synthesis_analysis`

`KomorebiPaperV/Lifting.lean`, line 21.

`KomorebiPaperV.lifting_analysis_synthesis`

`KomorebiPaperV/Lifting.lean`, line 25.

`KomorebiPaperV.lifting_determinant`

`KomorebiPaperV/Lifting.lean`, line 29.

9.9 5.1 Wavelet-packet reconstruction and energy

Traditional locator. Theorem 5.1, final paper p. 10; `result:09`. **Exact reviewed scope.** All finite depths of literal Haar packet operators; sums of true Hilbert adjoints reconstruct, and packet norms preserve energy. `KomorebiPaperV.WaveletPackets.packet_energy`

`KomorebiPaperV/WaveletPackets.lean`, line 68.

`KomorebiPaperV.WaveletPackets.packet_operator_reconstruction`

`KomorebiPaperV/WaveletPackets.lean`, line 94.

`KomorebiPaperV.WaveletPackets.packet_reconstruction`

`KomorebiPaperV/WaveletPackets.lean`, line 98.

9.10 6.1 Exact known-edge theorem

Traditional locator. Theorem 6.1, final paper p. 12; `result:10`. **Exact reviewed scope.** Exact known-edge cancellation in arbitrary seminormed additive groups. `KomorebiPaperV.known_edge_error`

`KomorebiPaperV/KnownEdge.lean`, line 17.

`KomorebiPaperV.known_edge_norm`

`KomorebiPaperV/KnownEdge.lean`, line 34.

9.11 6.2 Several exact edge packets

Traditional locator. Corollary 6.2, final paper p. 12; `result:11`. **Exact reviewed scope.** Finite families of exactly retained jump packets. `KomorebiPaperV.several_known_edges_error`

`KomorebiPaperV/KnownEdge.lean`, line 25.

`KomorebiPaperV.several_known_edges_norm`

`KomorebiPaperV/KnownEdge.lean`, line 38.

9.12 7.1 Packet perturbation bound

Traditional locator. Theorem 7.1, final paper p. 13; `result:12`. **Exact reviewed scope.** Measurable scalar masks in arbitrary measure spaces; extended finite-p norms, and pointwise plus essential supremum bounds. Finite mask and symmetric-difference measures yield the finite real-valued statement. `KomorebiPaperV.MaskPerturbation.indicator_difference_norm`

`KomorebiPaperV/MaskPerturbation.lean`, line 25.

`KomorebiPaperV.MaskPerturbation.finite_p_perturbation`

`KomorebiPaperV/MaskPerturbation.lean`, line 43.

`KomorebiPaperV.MaskPerturbation.pointwise_sup_bound`

`KomorebiPaperV/MaskPerturbation.lean`, line 59.

`KomorebiPaperV.MaskPerturbation.infinity_perturbation`

`KomorebiPaperV/MaskPerturbation.lean`, line 73.

`KomorebiPaperV.MaskPerturbation.full_signal_perturbation`

`KomorebiPaperV/MaskPerturbation.lean`, line 79.

9.13 11.2 Exact synthesis from every complete pruning

Traditional locator. Theorem 11.2, final paper p. 20; `result:13`. **Exact reviewed scope.** Literal finite node sums over covering antichains, with exact operator and applied-signal synthesis under the manuscript conservation hypotheses. `KomorebiPaperV.PruningSums.cutSum_eq_nodeSum`

`KomorebiPaperV/PruningSums.lean`, line 54.

`KomorebiPaperV.PruningSums.antichain_synthesis`

`KomorebiPaperV/PruningSums.lean`, line 81.

`KomorebiPaperV.PruningSums.packet_projector_synthesis`

`KomorebiPaperV/PruningSums.lean`, line 88.

9.14 11.3 Finite globally optimal stable selector

Traditional locator. Theorem 11.3, final paper p. 20; `result:14`. **Exact reviewed scope.** Named non-additive representation/leakage/complexity criterion; attained positive gap when competitors exist, uniform perturbation stability, and singleton case. Every pruning preserves synthesis. `KomorebiPaperV.PruningSums.objective_minimum`

`KomorebiPaperV/PruningSums.lean`, line 134.

`KomorebiPaperV.PruningSums.objective_gap_attained`

`KomorebiPaperV/PruningSums.lean`, line 165.

`KomorebiPaperV.PruningSums.objective_stability`

`KomorebiPaperV/PruningSums.lean`, line 148.

`KomorebiPaperV.PruningSums.singleton_selector`

`KomorebiPaperV/PruningSums.lean`, line 158.

`KomorebiPaperV.PruningSums.packet_projector_synthesis`

`KomorebiPaperV/PruningSums.lean`, line 88.

9.15 11.4 Dynamic-programming best packet basis

Traditional locator. Theorem 11.4, final paper p. 21; `result:15`. **Exact reviewed scope.** Cached bottom-up evaluator equals the global selector; stop on ties. One comparison per internal node; at most 3*node-count unit operations for visits/comparisons/value additions with node costs supplied. Not a Lean runtime or scalar-cost evaluation bound. `KomorebiPaperV.PruningAlgorithm.solve_correct`

`KomorebiPaperV/PruningAlgorithm.lean`, line 36.

`KomorebiPaperV.PruningAlgorithm.solve_comparisons`

`KomorebiPaperV/PruningAlgorithm.lean`, line 52.

`KomorebiPaperV.PruningAlgorithm.solve_visits`

`KomorebiPaperV/PruningAlgorithm.lean`, line 61.

`KomorebiPaperV.PruningAlgorithm.solve_additions`

`KomorebiPaperV/PruningAlgorithm.lean`, line 69.

`KomorebiPaperV.PruningAlgorithm.solve_linear_work`

`KomorebiPaperV/PruningAlgorithm.lean`, line 89.

`KomorebiPaperV.PruningAlgorithm.computed_antichain_optimal`

`KomorebiPaperV/PruningAlgorithm.lean`, line 97.

9.16 11.5 Local perturbation certificate

Traditional locator. Corollary 11.5, final paper p. 21; `result:16`. **Exact reviewed scope.** Exact maximum antichain error envelope and strict-gap stability; probability transfer from a joint local-error event is proved in PruningAlgorithm. `KomorebiPaperV.PruningSums.antichain_perturbation`

`KomorebiPaperV/PruningSums.lean`, line 108.

`KomorebiPaperV.PruningSums.antichain_envelope_attained`

`KomorebiPaperV/PruningSums.lean`, line 115.

`KomorebiPaperV.PruningSums.antichain_selector_stable`

KomorebiPaperV/PruningSums.lean, line 120.

KomorebiPaperV.PruningAlgorithm.antichain_probability

KomorebiPaperV/PruningAlgorithm.lean, line 102.

9.17 12.1 Cyclic product law

Traditional locator. Theorem 12.1, final paper p. 24; result:17. **Exact reviewed scope.** Formal power-series cyclic product and literal q-tuple power formula over commutative semirings, including q=0. Analytic interpretations apply on their convergence domains; no arbitrary divergent evaluation is asserted. KomorebiPaperV.CyclicSeries.cyclic_product_law

KomorebiPaperV/CyclicSeries.lean, line 126.

KomorebiPaperV.CyclicSeries.cyclic_power_law

KomorebiPaperV/CyclicSeries.lean, line 195.

9.18 13.2 Polyphase Möbius action

Traditional locator. Theorem 13.2, final paper p. 26; result:18. **Exact reviewed scope.** Explicit exponential root of unity and modulo-N action on the projective quotient of nonzero homogeneous pairs; Cayley conjugacy and the finite-chart formula. KomorebiPaperV.SmithCyclic.omega_pow

KomorebiPaperV/SmithCyclic.lean, line 12.

KomorebiPaperV.SmithCyclic.cyclic_composition

KomorebiPaperV/SmithCyclic.lean, line 78.

KomorebiPaperV.SmithCyclic.cayley_conjugacy

KomorebiPaperV/SmithCyclic.lean, line 87.

KomorebiPaperV.SmithCyclic.cyclic_chart

KomorebiPaperV/SmithCyclic.lean, line 92.

KomorebiPaperV.SmithCyclic.projective_mul

KomorebiPaperV/SmithCyclic.lean, line 56.

9.19 13.3 Impedance–admittance is the two-phase split

Traditional locator. Corollary 13.3, final paper p. 26; result:19. **Exact reviewed scope.** The N=2 phase is -1; its action swaps homogeneous coordinates, exchanging zero/infinity and giving inverse impedance on the nonzero chart. KomorebiPaperV.SmithCyclic.omega_two

KomorebiPaperV/SmithCyclic.lean, line 19.

KomorebiPaperV.SmithCyclic.two_phase_swap

KomorebiPaperV/SmithCyclic.lean, line 97.

KomorebiPaperV.SmithCyclic.two_phase_impedance_admittance

KomorebiPaperV/SmithCyclic.lean, line 100.

9.20 14.1 Dyadic–odd packet factorization

Traditional locator. Proposition 14.1, final paper p. 29; `result:20`. **Exact reviewed scope.** Signed integer CRT residue masks and dyadic-odd factorization. `KomorebiPaperV.CRT.residue_mask_factorization`

`KomorebiPaperV/CRT.lean`, line 20.

`KomorebiPaperV.CRT.dyadic_odd_packet_factorization`

`KomorebiPaperV/CRT.lean`, line 34.

10 The broader traditional developments

The final traditional edition preserves two substantial source strands. Part I contains the twenty-result formal core. Part II contains sixty-three proof records from the broader Annals development. Together they produce 83 proof records, with overlap in mathematical content. The number 83 counts presented proofs. It does not replace the twenty-result Lean denominator or establish 83 independent new results.

The table below connects groups of Part II developments to nearby formal-core concepts. "Nearby" is a reading aid. It does not assert a formal reduction or that the stated Part II result has been compiled in Lean.

Part II topic	Nearest formal core	Additional boundary
Operator partitions and refinement	Results 1.1–2.3, `HilbertPackets`	Broader operator settings or frames require their own assumptions and proofs.
Polyphase symbols and banks	Results 3.1–4.3, `FilterBank`, `ZTransform`	Unit-circle and L^2 boundary interpretations must be distinguished from off-circle weighted summability.
Frame trees and adaptive objectives	Results 11.2–11.5, `PruningSums`	Local frame bounds are nonnegative. The dynamic-programming result applies to additive costs.
Known edges, masks, and commutators	Results 6.1–7.1, `KnownEdge`, `MaskPerturbation`	A commutator using Q as an operator requires bounded linearity. Cancellation alone does not.
Boundary tubes and edge geometry	Mask perturbation theorem 7.1	Tube estimates use their local range $\delta < \delta_0$. Geometric constants and regularity are additional data.
Continuous Haar edge models	Discrete Haar results 4.2 and 5.1	Half-line step functions need care: their wavelet coefficients can be used even when the steps themselves are not in $L^2(\mathbb{R})$.
Random or asymptotic selection	Finite joint-event transfer 11.5	Probability bounds tending to one establish convergence in probability. Almost sure claims need stronger evidence.
Nonlinear cyclic products	Result 12.1, `CyclicSeries`	Analytic evaluation and stronger functional-analytic claims are additional to formal coefficient identities.

Part II topic	Nearest formal core	Additional boundary
Smith geometry and sampling	Results 13.2–14.1, `SmithCyclic`, `CRT`	The formal composition law alone is not a target proving exact-order converse or optimal sampling.

The final traditional audit records the precise scope repairs applied to Part II. This guide preserves the resulting hypotheses in its discussion. It does not incorporate altered claims into unchanged Lean targets. The following inventory provides a locator for every Part II proof record and explicitly keeps its formal status separate.

10.1 All sixty-three Part II proof records

Every row below has status: prose proof; no additional Lean claim. The final E-prefixed result number and page refer to the combined traditional paper. The nearest-core column identifies related reading only.

Final locator	Part II statement	Nearby core reading
E2.1, p. 42	Theorem: Impulse generation	1.1
E2.3, p. 43	Proposition: Finite periodic signals	1.1
E3.2, p. 43	Theorem: Exact lattice polyphase decomposition	2.1–2.3
E3.3, p. 44	Corollary: One dimensional residue coordinates	2.1–2.3
E3.4, p. 44	Proposition: Translation covariance	2.1–2.3
E3.5, p. 44	Theorem: Exact lattice refinement	2.1–2.3
E4.2, p. 46	Theorem: Exact convolution transfer law	3.1
E4.3, p. 46	Proposition: Transfer norm control	3.1
E4.4, p. 47	Proposition: Composition of transfer matrices	3.1
E4.5, p. 47	Corollary: Finite impulse response closure	3.1
E5.1, p. 48	Theorem: Perfect reconstruction criterion	4.1
E5.2, p. 49	Theorem: Quantitative imperfect reconstruction	4.1
E5.4, p. 49	Theorem: Stability under filter perturbation	4.1
E5.5, p. 50	Theorem: Frame bounds from the polyphase symbol	4.1
E5.6, p. 50	Corollary: Coefficient noise under canonical synthesis	4.1
E5.7, p. 50	Theorem: Perturbation of frame bounds	4.1
E5.9, p. 51	Corollary: Paraunitary bank	4.1
E5.10, p. 51	Corollary: Haar reconstruction and energy	4.2
E5.11, p. 52	Theorem: Exact lifting update	4.3

Final locator	Part II statement	Nearby core reading
E6.2, p. 53	Theorem: Reconstruction on a pruned packet tree	5.1, 11.2
E6.3, p. 53	Corollary: Coefficient perturbations in a unitary tree	5.1, 11.2
E6.4, p. 53	Proposition: Bounds for nonunitary local splits	5.1, 11.2
E6.5, p. 54	Theorem: Exact additive best basis recursion	11.3–11.5
E6.6, p. 55	Theorem: Stability of the optimal cost	11.3–11.5
E6.7, p. 55	Corollary: Stability of the selected basis	11.3–11.5
E7.2, p. 56	Theorem: Higher order singular packet decomposition	6.1–6.2
E7.3, p. 57	Corollary: Uniqueness of the trace packet	6.1–6.2
E7.5, p. 57	Theorem: Exact retained packet identity	6.1–6.2
E7.7, p. 58	Corollary: Sobolev rate after packet separation	6.1–6.2
E7.8, p. 58	Theorem: Reconstruction with an estimated packet	7.1
E8.1, p. 59	Theorem: Exact scalar region formula	7.1
E8.2, p. 60	Corollary: Location error for one step	7.1
E8.3, p. 60	Theorem: Location stability for higher order packets	7.1
E8.4, p. 61	Theorem: Overlap matrix formula	7.1
E8.5, p. 61	Corollary: Background and packet error	7.1
E8.7, p. 62	Theorem: Geometric packet bound	7.1, 11.5
E9.1, p. 63	Lemma: Contrast gap	7.1, 11.5
E9.2, p. 64	Theorem: Deterministic jump cell recovery	7.1, 11.5
E9.3, p. 64	Corollary: Jump count and sign recovery	7.1, 11.5
E9.4, p. 64	Corollary: Packet reconstruction after detection	7.1, 11.5
E9.6, p. 65	Corollary: Deterministic consistency	7.1, 11.5
E9.8, p. 67	Theorem: Probability recovery bound	7.1, 11.5
E9.9, p. 67	Corollary: Repeated measurements	7.1, 11.5
E9.11, p. 67	Corollary: Probability consistency	7.1, 11.5
E9.12, p. 68	Proposition: Vector jump recovery	7.1, 11.5
E10.1, p. 68	Theorem: Smooth Haar coefficient bound	4.2; edge context
E10.2, p. 69	Theorem: Exact Haar coefficient of a step	4.2; edge context
E10.3, p. 69	Corollary: Scale separation	4.2; edge context
E10.4, p. 69	Theorem: Half cell shift guarantee	4.2; edge context

Final locator	Part II statement	Nearby core reading
E10.5, p. 70	Corollary: Alignment free two grid Haar detection	4.2; edge context
E10.6, p. 71	Theorem: Polynomial annihilation and boundary support	6.1–7.1
E10.7, p. 71	Proposition: Noise amplification by differences	6.1–7.1
E10.8, p. 71	Corollary: Robust breakpoint bands	6.1–7.1
E11.2, p. 72	Theorem: Uniform convergence with exact jump packets	6.1–7.1
E11.3, p. 73	Theorem: Estimated packet Gibbs bound	6.1–7.1
E11.4, p. 73	Corollary: Boundary band error	6.1–7.1
E12.1, p. 74	Theorem: Residue law for Cauchy products	12.1
E12.2, p. 75	Theorem: Shifted pointwise monomial law	12.1
E12.3, p. 75	Corollary: Polynomial shift filters	12.1
E14.1, p. 77	Theorem: Cyclic Mobius action	13.2–13.3
E14.2, p. 78	Corollary: Impedance and admittance	13.2–13.3
E14.3, p. 78	Proposition: Radial and phase coordinates	13.2–13.3
E14.4, p. 78	Proposition: Coprime phase factorization	14.1

10.2 Open conjecture locators

Stable edge-packet recovery. Conjecture 7.2, final traditional p. 13; `core:record:17`. Open; excluded from the twenty-result formal contract.

Edge-packet approximation class. Conjecture 15.1, final traditional p. 29; `core:record:35`. Open; excluded from the twenty-result formal contract.

11 All 225 theorem targets

This catalog reproduces every qualified target from the frozen `candidate_contract.json`, grouped by its actual source module. Line numbers refer to the immutable bundled Lean sources. Every target was reconciled with its declaration line, audit directive, and inherited axiom-output record. This reconciliation does not perform Lean elaboration.

11.1 BinaryAdaptive: 6 targets

Source: `formalization/lean/KomorebiPaperV/BinaryAdaptive.lean`.

`KomorebiPaperV.Binary.cutSum_eq_root`

Source line 36; theorem.

`KomorebiPaperV.Binary.value_le_cost`

Source line 59; theorem.

`KomorebiPaperV.Binary.best_cost`

Source line 69; theorem.

`KomorebiPaperV.Binary.best_is_global_minimum`

Source line 79; theorem.

`KomorebiPaperV.Binary.root_le_envelope`

Source line 88; theorem.

`KomorebiPaperV.Binary.cut_perturbation`

Source line 94; theorem.

11.2 CRT: 4 targets

Source: `formalization/lean/KomorebiPaperV/CRT.lean`.

`KomorebiPaperV.CRT.residue_pair_iff`

Source line 13; theorem.

`KomorebiPaperV.CRT.residue_mask_factorization`

Source line 20; theorem.

`KomorebiPaperV.CRT.dyadic_odd_coprime`

Source line 31; theorem.

`KomorebiPaperV.CRT.dyadic_odd_packet_factorization`

Source line 34; theorem.

11.3 CompletePrunings: 15 targets

Source: `formalization/lean/KomorebiPaperV/CompletePrunings.lean`.

`KomorebiPaperV.CompletePrunings.terminal_exists`

Source line 36; theorem.

`KomorebiPaperV.CompletePrunings.node_on_terminal`

Source line 43; theorem.

`KomorebiPaperV.CompletePrunings.selected_stop_iff`

Source line 52; theorem.

`KomorebiPaperV.CompletePrunings.selected_split_root_false`

Source line 58; theorem.

`KomorebiPaperV.CompletePrunings.selected_split_down_iff`

Source line 62; theorem.

`KomorebiPaperV.CompletePrunings.cut_complete`

Source line 69; theorem.

`KomorebiPaperV.CompletePrunings.ancestor_onPath`

Source line 97; theorem.

`KomorebiPaperV.CompletePrunings.path_nodes_comparable`

Source line 108; theorem.

`KomorebiPaperV.CompletePrunings.complete_iff_covering_antichain`

Source line 120; theorem.

`KomorebiPaperV.CompletePrunings.complete_root_unique`

Source line 140; theorem.

`KomorebiPaperV.CompletePrunings.complete_child`

Source line 149; theorem.

`KomorebiPaperV.CompletePrunings.complete_has_cut`

Source line 164; theorem.

`KomorebiPaperV.CompletePrunings.cut_ext`

Source line 192; theorem.

`KomorebiPaperV.CompletePrunings.complete_pruning_finite`

Source line 226; theorem.

`KomorebiPaperV.CompletePrunings.complete_pruning_minimum`

Source line 230; theorem.

11.4 CyclicSeries: 15 targets

Source: `formalization/lean/KomorebiPaperV/CyclicSeries.lean`.

`KomorebiPaperV.CyclicSeries.coeff_cyclicProjector`

Source line 33; theorem.

`KomorebiPaperV.CyclicSeries.cyclicProjector_idempotent`

Source line 38; theorem.

`KomorebiPaperV.CyclicSeries.cyclicProjector_orthogonal`

Source line 43; theorem.

`KomorebiPaperV.CyclicSeries.cyclicProjector_eq_zero_of_residue_ne`

Source line 51; theorem.

`KomorebiPaperV.CyclicSeries.cyclicProjector_eq_of_mod_eq`

Source line 56; theorem.

`KomorebiPaperV.CyclicSeries.cyclicProjector_add`

Source line 62; theorem.

`KomorebiPaperV.CyclicSeries.cyclicProjector_sum`

Source line 67; theorem.

`KomorebiPaperV.CyclicSeries.sum_cyclicProjector`

Source line 78; theorem.

`KomorebiPaperV.CyclicSeries.cyclicProjector_product_packet`

Source line 92; theorem.

`KomorebiPaperV.CyclicSeries.cyclicProjector_product_other`

Source line 118; theorem.

`KomorebiPaperV.CyclicSeries.cyclic_product_law`

Source line 126; theorem.

`KomorebiPaperV.CyclicSeries.pure_finite_product`

Source line 159; theorem.

`KomorebiPaperV.CyclicSeries.project_finite_product`

Source line 177; theorem.

`KomorebiPaperV.CyclicSeries.sum_fin_projector`

Source line 189; theorem.

`KomorebiPaperV.CyclicSeries.cyclic_power_law`

Source line 195; theorem.

11.5 FIRTransfer: 6 targets

Source: `formalization/lean/KomorebiPaperV/FIRTransfer.lean`.

`KomorebiPaperV.FIRTransfer.tapEquiv_symm`

Source line 34; theorem.

`KomorebiPaperV.FIRTransfer.tapIndex_equiv`

Source line 41; theorem.

`KomorebiPaperV.FIRTransfer.transfer_address`

Source line 48; theorem.

`KomorebiPaperV.FIRTransfer.fir_reindex`

Source line 65; theorem.

`KomorebiPaperV.FIRTransfer.sum_pair_fiber`

Source line 82; theorem.

`KomorebiPaperV.FIRTransfer.fir_polyphase_transfer`

Source line 96; theorem.

11.6 FilterBank: 13 targets

Source: `formalization/lean/KomorebiPaperV/FilterBank.lean`.

`KomorebiPaperV.FilterBank.act_apply`

Source line 31; theorem.

`KomorebiPaperV.FilterBank.act_is_fir`

Source line 36; theorem.

`KomorebiPaperV.FilterBank.act_impulse`

Source line 43; theorem.

`KomorebiPaperV.FilterBank.act_single`

Source line 50; theorem.

`KomorebiPaperV.FilterBank.evalZ_apply`

Source line 65; theorem.

`KomorebiPaperV.FilterBank.filter_transform`

Source line 70; theorem.

`KomorebiPaperV.FilterBank.bank_mul`

Source line 84; theorem.

`KomorebiPaperV.FilterBank.bank_impulse`

Source line 93; theorem.

`KomorebiPaperV.FilterBank.bank_faithful`

Source line 104; theorem.

`KomorebiPaperV.FilterBank.gain_delay_action`

Source line 121; theorem.

`KomorebiPaperV.FilterBank.perfect_reconstruction_iff`

Source line 133; theorem.

`KomorebiPaperV.FilterBank.bank_transform`

Source line 147; theorem.

`KomorebiPaperV.FilterBank.original_sample_delay`

Source line 155; theorem.

11.7 FiniteAdaptive: 18 targets

Source: `formalization/lean/KomorebiPaperV/FiniteAdaptive.lean`.

`KomorebiPaperV.FiniteAdaptive.cutSum_eq_root`

Source line 39; theorem.

`KomorebiPaperV.FiniteAdaptive.cutSum_eq_identity`

Source line 53; theorem.

`KomorebiPaperV.FiniteAdaptive.pathHits_one`

Source line 67; theorem.

`KomorebiPaperV.FiniteAdaptive.arbitrary_objective_minimum`

Source line 107; theorem.

`KomorebiPaperV.FiniteAdaptive.strict_minimum_positive_gap`

Source line 114; theorem.

`KomorebiPaperV.FiniteAdaptive.value_le_cost`

Source line 136; theorem.

`KomorebiPaperV.FiniteAdaptive.best_cost`

Source line 146; theorem.

`KomorebiPaperV.FiniteAdaptive.best_is_global_minimum`

Source line 156; theorem.

`KomorebiPaperV.FiniteAdaptive.best_preserves_synthesis`

Source line 161; theorem.

`KomorebiPaperV.FiniteAdaptive.stop_on_tie`

Source line 165; theorem.

`KomorebiPaperV.FiniteAdaptive.root_le_envelope`

Source line 174; theorem.

`KomorebiPaperV.FiniteAdaptive.cost_le_envelope`

Source line 179; theorem.

`KomorebiPaperV.FiniteAdaptive.envelope_attained`

Source line 187; theorem.

`KomorebiPaperV.FiniteAdaptive.cut_perturbation`

Source line 201; theorem.

`KomorebiPaperV.FiniteAdaptive.value_perturbation`

Source line 213; theorem.

`KomorebiPaperV.FiniteAdaptive.best_unchanged_of_gap`

Source line 223; theorem.

`KomorebiPaperV.FiniteAdaptive.one_comparison_per_internal_node`

Source line 243; theorem.

`KomorebiPaperV.FiniteAdaptive.selector_probability`

Source line 252; theorem.

11.8 FiniteSelection: 5 targets

Source: `formalization/lean/KomorebiPaperV/FiniteSelection.lean`.

`KomorebiPaperV.finite_minimum`

Source line 15; theorem.

`KomorebiPaperV.positive_gap`

Source line 48; theorem.

`KomorebiPaperV.gap_stability`

Source line 58; theorem.

`KomorebiPaperV.singleton_minimum`

Source line 69; theorem.

`KomorebiPaperV.singleton_selection_unchanged`

Source line 76; theorem.

11.9 FiniteTransforms: 12 targets

Source: `formalization/lean/KomorebiPaperV/FiniteTransforms.lean`.

`KomorebiPaperV.FiniteTransforms.unpair_pair`

Source line 43; theorem.

`KomorebiPaperV.FiniteTransforms.pair_unpair`

Source line 48; theorem.

`KomorebiPaperV.FiniteTransforms.flat_synthesis_analysis`

Source line 59; theorem.

`KomorebiPaperV.FiniteTransforms.flat_analysis_synthesis`

Source line 64; theorem.

`KomorebiPaperV.FiniteTransforms.unlift_lift_flat`

Source line 78; theorem.

`KomorebiPaperV.FiniteTransforms.lift_unlift_flat`

Source line 83; theorem.

`KomorebiPaperV.FiniteTransforms.haar_involution`

Source line 92; theorem.

`KomorebiPaperV.FiniteTransforms.haar_pair_energy`

Source line 103; theorem.

`KomorebiPaperV.FiniteTransforms.haar_finite_energy`

Source line 110; theorem.

`KomorebiPaperV.FiniteTransforms.haar_flat_involution`

Source line 118; theorem.

`KomorebiPaperV.FiniteTransforms.matrix_eq_of_all_vectors`

Source line 125; theorem.

`KomorebiPaperV.FiniteTransforms.perfect_reconstruction_matrix_iff`

Source line 134; theorem.

11.10 HaarChannels: 8 targets

Source: `formalization/lean/KomorebiPaperV/HaarChannels.lean`.

`KomorebiPaperV.HaarChannels.norm_sq_tsum`

Source line 12; theorem.

`KomorebiPaperV.HaarChannels.channel_energy`

Source line 15; theorem.

`KomorebiPaperV.HaarChannels.even_apply`

Source line 26; theorem.

`KomorebiPaperV.HaarChannels.odd_apply`

Source line 30; theorem.

`KomorebiPaperV.HaarChannels.interleaved_energy`

Source line 32; theorem.

`KomorebiPaperV.HaarChannels.haar_energy`

Source line 40; theorem.

`KomorebiPaperV.HaarChannels.haar_low_samples`

Source line 44; theorem.

`KomorebiPaperV.HaarChannels.haar_high_samples`

Source line 48; theorem.

11.11 HaarHilbert: 5 targets

Source: `formalization/lean/KomorebiPaperV/HaarHilbert.lean`.

`KomorebiPaperV.HaarHilbert.inverse_low`

Source line 15; theorem.

`KomorebiPaperV.HaarHilbert.inverse_high`

Source line 24; theorem.

`KomorebiPaperV.HaarHilbert.energy`

Source line 33; theorem.

`KomorebiPaperV.HaarHilbert.low_apply`

Source line 42; theorem.

`KomorebiPaperV.HaarHilbert.high_apply`

Source line 47; theorem.

11.12 HilbertPackets: 23 targets

Source: `formalization/lean/KomorebiPaperV/HilbertPackets.lean`.

`KomorebiPaperV.HilbertPackets.projection_apply`

Source line 26; theorem.

`KomorebiPaperV.HilbertPackets.projection_norm_le`

Source line 30; theorem.

`KomorebiPaperV.HilbertPackets.projector_apply`

Source line 53; theorem.

`KomorebiPaperV.HilbertPackets.projector_idempotent`

Source line 56; theorem.

`KomorebiPaperV.HilbertPackets.projector_annihilate`

Source line 62; theorem.

`KomorebiPaperV.HilbertPackets.projector_complete`

Source line 69; theorem.

`KomorebiPaperV.HilbertPackets.projector_refinement`

Source line 79; theorem.

`KomorebiPaperV.HilbertPackets.reindex_norm`

Source line 102; theorem.

`KomorebiPaperV.HilbertPackets.polyphase_apply`

Source line 120; theorem.

`KomorebiPaperV.HilbertPackets.polyphase_synthesis_analysis`

Source line 123; theorem.

`KomorebiPaperV.HilbertPackets.polyphase_analysis_synthesis`

Source line 127; theorem.

`KomorebiPaperV.HilbertPackets.polyphase_energy`

Source line 131; theorem.

`KomorebiPaperV.HilbertPackets.impulse_expansion`

Source line 135; theorem.

`KomorebiPaperV.HilbertPackets.projector_symmetric`

Source line 139; theorem.

`KomorebiPaperV.HilbertPackets.projector_orthogonal`

Source line 147; theorem.

`KomorebiPaperV.HilbertPackets.impulse_basis_apply`

Source line 161; theorem.

`KomorebiPaperV.HilbertPackets.impulse_dense_span`

Source line 165; theorem.

`KomorebiPaperV.HilbertPackets.shift_apply`

Source line 175; theorem.

`KomorebiPaperV.HilbertPackets.reindex_single`

Source line 177; theorem.

`KomorebiPaperV.HilbertPackets.shift_impulse`

Source line 183; theorem.

`KomorebiPaperV.HilbertPackets.residue_projector_refinement`

Source line 187; theorem.

`KomorebiPaperV.HilbertPackets.shift_generated_dense_span`

Source line 203; theorem.

`KomorebiPaperV.HilbertPackets.finite_impulse_expansion`

Source line 207; theorem.

11.13 IntegerPolyphase: 11 targets

Source: `formalization/lean/KomorebiPaperV/IntegerPolyphase.lean`.

`KomorebiPaperV.IntegerPolyphase.phase_val`

Source line 26; theorem.

`KomorebiPaperV.IntegerPolyphase.address_phase_quotient`

Source line 31; theorem.

`KomorebiPaperV.IntegerPolyphase.address_divmod`

Source line 37; theorem.

`KomorebiPaperV.IntegerPolyphase.phase_address`

Source line 47; theorem.

`KomorebiPaperV.IntegerPolyphase.quotient_address`

Source line 54; theorem.

`KomorebiPaperV.IntegerPolyphase.synthesis_analysis`

Source line 73; theorem.

`KomorebiPaperV.IntegerPolyphase.analysis_synthesis`

Source line 78; theorem.

`KomorebiPaperV.IntegerPolyphase.channel_delay`

Source line 87; theorem.

`KomorebiPaperV.IntegerPolyphase.sample_delay`

Source line 94; theorem.

`KomorebiPaperV.IntegerPolyphase.emod_coarsen`

Source line 106; theorem.

`KomorebiPaperV.IntegerPolyphase.residue_refinement`

Source line 111; theorem.

11.14 KnownEdge: 6 targets

Source: `formalization/lean/KomorebiPaperV/KnownEdge.lean`.

`KomorebiPaperV.known_edge_error`

Source line 17; theorem.

`KomorebiPaperV.estimated_edge_error`

Source line 21; theorem.

`KomorebiPaperV.several_known_edges_error`

Source line 25; theorem.

`KomorebiPaperV.known_edge_norm`

Source line 34; theorem.

`KomorebiPaperV.several_known_edges_norm`

Source line 38; theorem.

`KomorebiPaperV.estimated_edge_norm_bound`

Source line 43; theorem.

11.15 Lifting: 4 targets

Source: `formalization/lean/KomorebiPaperV/Lifting.lean`.

`KomorebiPaperV.lifting_synthesis_analysis`

Source line 21; theorem.

`KomorebiPaperV.lifting_analysis_synthesis`

Source line 25; theorem.

`KomorebiPaperV.lifting_determinant`

Source line 29; theorem.

`KomorebiPaperV.lifting_coordinates_roundtrip`

Source line 39; theorem.

11.16 MaskPerturbation: 9 targets

Source: `formalization/lean/KomorebiPaperV/MaskPerturbation.lean`.

`KomorebiPaperV.MaskPerturbation.jump_measurable`

Source line 14; theorem.

`KomorebiPaperV.MaskPerturbation.symmetric_difference_norm`

Source line 18; theorem.

`KomorebiPaperV.MaskPerturbation.indicator_difference_norm`

Source line 25; theorem.

`KomorebiPaperV.MaskPerturbation.jump_sub_same_set`

Source line 32; theorem.

`KomorebiPaperV.MaskPerturbation.jump_error_decomposition`

Source line 38; theorem.

`KomorebiPaperV.MaskPerturbation.finite_p_perturbation`

Source line 43; theorem.

`KomorebiPaperV.MaskPerturbation.pointwise_sup_bound`

Source line 59; theorem.

`KomorebiPaperV.MaskPerturbation.infinity_perturbation`

Source line 73; theorem.

`KomorebiPaperV.MaskPerturbation.full_signal_perturbation`

Source line 79; theorem.

11.17 Packets: 8 targets

Source: `formalization/lean/KomorebiPaperV/Packets.lean`.

`KomorebiPaperV.packet_idempotent`

Source line 20; theorem.

`KomorebiPaperV.packet_annihilate`

Source line 25; theorem.

`KomorebiPaperV.packet_complete`

Source line 36; theorem.

`KomorebiPaperV.packet_refinement`

Source line 48; theorem.

`KomorebiPaperV.address_synthesize_analyze`

Source line 74; theorem.

`KomorebiPaperV.address_analyze_synthesize`

Source line 79; theorem.

`KomorebiPaperV.residuePacket_idempotent`

Source line 91; theorem.

`KomorebiPaperV.residuePacket_complete`

Source line 95; theorem.

11.18 PruningAlgorithm: 8 targets

Source: `formalization/lean/KomorebiPaperV/PruningAlgorithm.lean`.

`KomorebiPaperV.PruningAlgorithm.solve_correct`

Source line 36; theorem.

`KomorebiPaperV.PruningAlgorithm.solve_comparisons`

Source line 52; theorem.

`KomorebiPaperV.PruningAlgorithm.solve_visits`

Source line 61; theorem.

`KomorebiPaperV.PruningAlgorithm.solve_additions`

Source line 69; theorem.

`KomorebiPaperV.PruningAlgorithm.internal_le_nodes`

Source line 81; theorem.

`KomorebiPaperV.PruningAlgorithm.solve_linear_work`

Source line 89; theorem.

`KomorebiPaperV.PruningAlgorithm.computed_antichain_optimal`

Source line 97; theorem.

`KomorebiPaperV.PruningAlgorithm.antichain_probability`

Source line 102; theorem.

11.19 PruningSums: 15 targets

Source: `formalization/lean/KomorebiPaperV/PruningSums.lean`.

`KomorebiPaperV.PruningSums.node_sum_branch`

Source line 41; theorem.

`KomorebiPaperV.PruningSums.mem_selectedNodes`

Source line 51; theorem.

`KomorebiPaperV.PruningSums.cutSum_eq_nodeSum`

Source line 54; theorem.

`KomorebiPaperV.PruningSums.pruning_nodeSum`

Source line 78; theorem.

`KomorebiPaperV.PruningSums.antichain_synthesis`

Source line 81; theorem.

`KomorebiPaperV.PruningSums.packet_projector_synthesis`

Source line 88; theorem.

`KomorebiPaperV.PruningSums.selected_global_minimum`

Source line 102; theorem.

`KomorebiPaperV.PruningSums.antichain_perturbation`

Source line 108; theorem.

`KomorebiPaperV.PruningSums.antichain_envelope_attained`

Source line 115; theorem.

`KomorebiPaperV.PruningSums.antichain_selector_stable`

Source line 120; theorem.

`KomorebiPaperV.PruningSums.objective_minimum`

Source line 134; theorem.

`KomorebiPaperV.PruningSums.objective_positive_gap`

Source line 138; theorem.

`KomorebiPaperV.PruningSums.objective_stability`

Source line 148; theorem.

`KomorebiPaperV.PruningSums.singleton_selector`

Source line 158; theorem.

`KomorebiPaperV.PruningSums.objective_gap_attained`

Source line 165; theorem.

11.20 SmithAction: 10 targets

Source: `formalization/lean/KomorebiPaperV/SmithAction.lean`.

`KomorebiPaperV.SmithAction.action_mul`

Source line 16; theorem.

`KomorebiPaperV.SmithAction.action_one`

Source line 20; theorem.

`KomorebiPaperV.SmithAction.action_scale`

Source line 23; theorem.

`KomorebiPaperV.SmithAction.action_nonzero`

Source line 28; theorem.

`KomorebiPaperV.SmithAction.half_turn`

Source line 36; theorem.

`KomorebiPaperV.SmithAction.half_turn_zero`

Source line 39; theorem.

`KomorebiPaperV.SmithAction.half_turn_infinity`

Source line 42; theorem.

`KomorebiPaperV.SmithAction.homogeneous_chart`

Source line 49; theorem.

`KomorebiPaperV.SmithAction.chart_composition`

Source line 55; theorem.

`KomorebiPaperV.SmithAction.finite_half_turn`

Source line 66; theorem.

11.21 SmithCyclic: 12 targets

Source: `formalization/lean/KomorebiPaperV/SmithCyclic.lean`.

`KomorebiPaperV.SmithCyclic.omega_ne_zero`

Source line 10; theorem.

`KomorebiPaperV.SmithCyclic.omega_pow`

Source line 12; theorem.

`KomorebiPaperV.SmithCyclic.omega_two`

Source line 19; theorem.

`KomorebiPaperV.SmithCyclic.action_respects_scaling`

Source line 45; theorem.

`KomorebiPaperV.SmithCyclic.projective_mul`

Source line 56; theorem.

`KomorebiPaperV.SmithCyclic.projective_one`

Source line 65; theorem.

`KomorebiPaperV.SmithCyclic.cyclic_zero`

Source line 75; theorem.

`KomorebiPaperV.SmithCyclic.cyclic_composition`

Source line 78; theorem.

`KomorebiPaperV.SmithCyclic.cayley_conjugacy`

Source line 87; theorem.

`KomorebiPaperV.SmithCyclic.cyclic_chart`

Source line 92; theorem.

`KomorebiPaperV.SmithCyclic.two_phase_swap`

Source line 97; theorem.

`KomorebiPaperV.SmithCyclic.two_phase_impedance_admittance`

Source line 100; theorem.

11.22 WaveletPackets: 7 targets

Source: `formalization/lean/KomorebiPaperV/WaveletPackets.lean`.

`KomorebiPaperV.WaveletPackets.low_norm_bound`

Source line 35; theorem.

`KomorebiPaperV.WaveletPackets.high_norm_bound`

Source line 39; theorem.

`KomorebiPaperV.WaveletPackets.child_energy`

Source line 53; theorem.

`KomorebiPaperV.WaveletPackets.packet_energy`

Source line 68; theorem.

`KomorebiPaperV.WaveletPackets.energy_implies_adjoint_identity`

Source line 80; theorem.

`KomorebiPaperV.WaveletPackets.packet_operator_reconstruction`

Source line 94; theorem.

`KomorebiPaperV.WaveletPackets.packet_reconstruction`

Source line 98; theorem.

11.23 ZTransform: 5 targets

Source: `formalization/lean/KomorebiPaperV/ZTransform.lean`.

`KomorebiPaperV.ZTransform.transform_unique`

Source line 18; theorem.

`KomorebiPaperV.ZTransform.delay_transform`

Source line 21; theorem.

`KomorebiPaperV.ZTransform.finite_transfer_transform`

Source line 34; theorem.

`KomorebiPaperV.ZTransform.fir_channel_transform`

Source line 49; theorem.

`KomorebiPaperV.ZTransform.fir_matrix_transform`

Source line 64; theorem.

12 Source architecture and exact dependency pins

The import list below is extracted from the 25 Lean files: the 23 proof modules, the aggregate import file, and the audit driver. Imports beginning with `Mathlib` are external mathematical dependencies. Imports beginning with `KomorebiPaperV` are local. The import graph describes compilation dependencies, which can be broader than the logical dependency of an individual theorem.

12.1 `KernelAudit.lean`

`KomorebiPaperV`

12.2 `KomorebiPaperV.lean`

`KomorebiPaperV.Packets`, `KomorebiPaperV.KnownEdge`, `KomorebiPaperV.Lifting`, `KomorebiPaperV.FiniteSelection`, `KomorebiPaperV.BinaryAdaptive`, `KomorebiPaperV.IntegerPolyphase`, `KomorebiPaperV.FIRTransfer`, `KomorebiPaperV.FiniteAdaptive`, `KomorebiPaperV.FiniteTransforms`, `KomorebiPaperV.HilbertPackets`, `KomorebiPaperV.CompletePrunings`, `KomorebiPaperV.CRT`, `KomorebiPaperV.SmithAction`, `KomorebiPaperV.HaarHilbert`, `KomorebiPaperV.SmithCyclic`, `KomorebiPaperV.CyclicSeries`, `KomorebiPaperV.HaarChannels`, `KomorebiPaperV.PruningSums`, `KomorebiPaperV.MaskPerturbation`, `KomorebiPaperV.WaveletPackets`, `KomorebiPaperV.PruningAlgorithm`, `KomorebiPaperV.ZTransform`, `KomorebiPaperV.FilterBank`

12.3 `KomorebiPaperV/BinaryAdaptive.lean`

`Mathlib.Basic.Real.Basic`, `Mathlib.Tactic`

12.4 `KomorebiPaperV/CRT.lean`

`Mathlib.Data.Nat.Prime.Basic`, `KomorebiPaperV.Packets`

12.5 `KomorebiPaperV/CompletePrunings.lean`

`KomorebiPaperV.FiniteAdaptive`

12.6 `KomorebiPaperV/CyclicSeries.lean`

`Mathlib.RingTheory.PowerSeries.Basic`, `Mathlib.Algebra.BigOperators.Ring.Finset`, `Mathlib.Algebra.BigOperators.Fin`, `Mathlib.Tactic`

12.7 `KomorebiPaperV/FIRTransfer.lean`

`KomorebiPaperV.IntegerPolyphase`

12.8 `KomorebiPaperV/FilterBank.lean`

`KomorebiPaperV.ZTransform`, `Mathlib.Algebra.MonoidAlgebra.Basic`, `Mathlib.LinearAlgebra.Matrix.ToLin`

12.9 KomorebiPaperV/FiniteAdaptive.lean

`Mathlib.MeasureTheory.Measure.Basic`, `KomorebiPaperV.FiniteSelection`

12.10 KomorebiPaperV/FiniteSelection.lean

`Mathlib.Basic.Real.Basic`, `Mathlib.Tactic`

12.11 KomorebiPaperV/FiniteTransforms.lean

`Mathlib.Analysis.Real.Sqrt`, `KomorebiPaperV.Lifting`, `KomorebiPaperV.Packets`

12.12 KomorebiPaperV/HaarChannels.lean

`KomorebiPaperV.HaarHilbert`

12.13 KomorebiPaperV/HaarHilbert.lean

`KomorebiPaperV.HilbertPackets`, `Mathlib.Analysis.Real.Sqrt`

12.14 KomorebiPaperV/HilbertPackets.lean

`Mathlib.Analysis.InnerProductSpace.l2Space`, `Mathlib.Analysis.Complex.Basic`, `Mathlib.Analysis.Normed.Lp.lpSpace`, `KomorebiPaperV.IntegerPolyphase`

12.15 KomorebiPaperV/IntegerPolyphase.lean

`KomorebiPaperV.Packets`

12.16 KomorebiPaperV/KnownEdge.lean

`Mathlib.Analysis.Normed.Group.Basic`, `Mathlib.Tactic`

12.17 KomorebiPaperV/Lifting.lean

`Mathlib.LinearAlgebra.Matrix.Determinant.Basic`, `Mathlib.LinearAlgebra.Matrix.Notation`, `Mathlib.Tactic`

12.18 KomorebiPaperV/MaskPerturbation.lean

`Mathlib.MeasureTheory.Function.LpSeminorm.Indicator`, `Mathlib.MeasureTheory.Function.LpSeminorm.TriangleInequality`, `Mathlib.Basic.Complex.Basic`, `Mathlib.Tactic`

12.19 KomorebiPaperV/Packets.lean

`Mathlib.Data.ZMod.Basic`, `Mathlib.Tactic`

12.20 `KomorebiPaperV/PruningAlgorithm.lean`

`KomorebiPaperV.PruningSums`

12.21 `KomorebiPaperV/PruningSums.lean`

`KomorebiPaperV.CompletePrunings`, `KomorebiPaperV.HilbertPackets`

12.22 `KomorebiPaperV/SmithAction.lean`

`Mathlib.Basic.Complex.Basic`, `Mathlib.Tactic`

12.23 `KomorebiPaperV/SmithCyclic.lean`

`KomorebiPaperV.SmithAction`, `Mathlib.Analysis.SpecialFunctions.Trigonometric.Basic`

12.24 `KomorebiPaperV/WaveletPackets.lean`

`KomorebiPaperV.HaarChannels`, `Mathlib.Analysis.InnerProductSpace.Adjoint`

12.25 `KomorebiPaperV/ZTransform.lean`

`KomorebiPaperV.FIRTransfer`, `Mathlib.Analysis.SpecialFunctions.Pow.Complex`, `Mathlib.LinearAlgebra.Matrix.ToLin`

12.26 Pinned external revisions

The package uses Lean `leanprover/lean4:v4.34.0-rc2`. The following revisions are copied from `lake-manifest.json`. They specify the historical dependency state. This guide preserves those revisions.

Package	Exact revision
mathlib	`aeb72f1e9981b6d35b426b16dca7759be3e7ba06`
plausible	`d9598f07b1bc701f1e3aae163d2681c1fd978793`
LeanSearchClient	`ba67e212be1197b84c1f1f6299488a10a3002713`
importGraph	`d8823026ac7ef130c253089d95685f9877b95323`
proofwidgets	`a8acbfd87375ff4abe14ce09db5b7664d383bc7f`
aesop	`18889deb9e83ea7420ef51c160d6f88552e744e3`
Qq	`507746ab8f4b643ccdacb2ec4cdb5853fa9f8ab3`
batteries	`4cac2177c37f5530c4da76aa8e4307f3fc9e4dcb`
Cli	`ab3a82db9fea14cf0fd7f5a2de650f4b534640af`