PAPER VII

Quantum Seed and Packet Calculus

Symmetry, circuits, geometric transport, channels, and finite effective dynamics

Follow the actual declarations from algebra orbits and packet circuits to geometric transport, quantum recovery, and finite resolvent bounds. Each interface keeps its own hypotheses and evidence.

Komorebi P.

Lean and evidence guide • 18 October 2026

How to use this volume

A guide to the actual formal development. The formal development covers 19 finite items through 323 named theorem declarations in 25 proof modules. The teaching chapters explain the mathematical interfaces and their hypotheses. The complete registers then give exact declarations, source locations, imports, and nine dependency pins.

This edition preserves 67 exact Lean excerpts. Six constructed objects have a separate inherited audit of their proof fields. Those six objects are recorded separately from the catalog of 323 theorems. Five continuum physics conjectures remain open.

Kernel execution evidence is inherited from the v0.6.0 release dated 6 September 2026. This edition checks the correspondence between the source files, declaration locations, saved axiom records, and printed excerpts. It also describes the finite computations recorded on 9 October 2026. The present proofreading pass did not rerun those computations or compile Lean.

Contents

1 Reading a Paper VII Lean theorem 1

2 Navigation through the traditional paper 4

3 Seeds, local orbits, and the packet resolution 10

4 Circuit corners, conserved packets, and leakage 18

5 Moving frames and an actual projected connection 23

6 Finite channels and the full Knill–Laflamme criterion 32

7 Effective resolvents, Euclidean transfer, and finite bounds 42

8 Evidence and reproducible replay 47

9 Complete map of finite results 52

10 All named theorem declarations 57

11 Module dependencies, source identities, and pins 69

12 Exact excerpt register 72

1 Reading a Paper VII Lean theorem

1.1 Three readings of one construction

The traditional reading begins with a finite state space, its operator algebra and a packet resolution. It derives matrix identities and explains what they mean for dynamics and recovery. The visual reading displays the same objects as corners, address sets, syndrome subspaces and local coordinates. The Lean reading asks a further question: what precise data must be supplied before each conclusion can be constructed?

An endpoint is a theorem whose conclusion expresses the desired result. A supporting lemma may isolate one algebraic or analytic step. A definition constructs an object but is not itself a proof that the object has every advertised property. The formalization deliberately uses several layers: general ring identities, complex matrix specializations, norm estimates, real calculus, spectral constructions and manifold calculus. Reading only a final theorem name loses the distinction between these layers.

The source base is the unchanged v0.6.0 snapshot. Older supplements explain how coverage grew, but their partial descriptions are not the last word on the current source. For example, the fourth block constructed recovery for already orthogonal syndrome data. The fifth block added the spectral reduction and necessity proof for general positive Gram data. The sixth block constructed the range atlas and global manifold curvature. This guide follows the current declarations and retains the older history only where it explains an assumption or correction.

1.2 Finite types, matrix types and type classes

Lean's `Matrix d m ℂ` means a function with a row index in `d`, a column index in `m`, and values in the complex numbers. The indices are types, not necessarily integers. A `Fintype d` instance gives the finite enumeration needed for a sum or dimension. A `DecidableEq d` instance lets a definition decide whether two indices agree. `Fin n` is the type of natural numbers smaller than `n`. It supplies a concrete index type without replacing an arbitrary finite type throughout the library.

The notation `Type*` leaves a universe level implicit. Braces in a binder, such as `{n : Type*}`, denote an implicit argument inferred from later data. Round parentheses denote an explicit argument. Square brackets denote an argument supplied through a type class. This is a structure or property that instance search should obtain. These brackets do not make an assumption disappear. A theorem with `[Nonempty n]` is only available when the state index has an inhabitant. A theorem with `[Invertible D]` has an actual invertibility requirement even though no ordinary parameter named `hD` is printed.

The declarations set `autoImplicit false`. Consequently an undeclared symbol is not silently promoted to a fresh theorem variable. A local `variable` command can still supply variables and type classes to subsequent declarations. An excerpt must therefore be read together with its surrounding section, not as an independent file. The chapter contracts explain those surrounding assumptions. The source file and line register allow the reader to check them directly.

Source notation	Mathematical reading
`Matrix d m ℂ`	This is a rectangular complex matrix. It can also represent the coefficient array of a bipartite state.
`n → ℂ`	This is a coordinate vector on a finite index type.
`→ₗ[ℂ]`	This map is linear over the complex numbers, with linearity stored in its structure.

Source notation	Mathematical reading
`→L[ℝ]`	This continuous map is linear over the real numbers. It is used for derivatives and operators that vary with a parameter.
`≃ₗ[ℂ]`	This is a linear equivalence with an inverse and linearity laws.
`≃ₜ`	This is a homeomorphism, including continuity in both directions.
`≃L[ℝ]`	This is a continuous equivalence that is linear over the real numbers.
`star`, `conjTranspose`	These denote the involution in an abstract star ring and the conjugate transpose for complex matrices.
`PosSemidef`, `PosDef`	These predicates describe positive semidefinite and positive definite matrices, respectively.
`IsStarProjection P`	This predicate requires both idempotence and self-adjointness of the projector.
`HasDerivAt f D t`	The actual real derivative of `f` at `t` is `D`.
`fderiv ℝ`, `mvfderiv I`	These are the Fréchet derivative and the derivative of a map from a manifold to a vector space.
`IsLeast S r`	This asserts that `r` belongs to `S` and is a lower bound for it.

1.3 Quantifiers and the information a structure stores

A family `P : I → R` is one element of the ring for each packet index. A dependent family `P : (i : F) → J i → Matrix (D i) (D i) ℂ` allows both the local dimension and the type of local packet addresses to vary with the tensor factor. This is more general than a tensor power of one fixed matrix.

The packet structure stores pairwise multiplication and completeness. Its multiplication law includes the diagonal case, so idempotence is a consequence. Self-adjointness is supplied separately when the construction is specialized to a star ring. This distinction lets circuit reconstruction work over any ring while the unitary identities require an adjoint.

Source: `formalization/QuantumPacketCore.lean`.

Exact lines 22 to 25. Excerpt `packet_structure`. Hash in Section 12.

```
structure PacketFamily (R I : Type*) [Ring R] [Fintype I] [DecidableEq I] where
  proj : I → R
  mul_proj : ∀ i j, proj i * proj j = if i = j then proj i else 0
  sum_proj : ∑ i, proj i = 1
```

Read `proj : I → R` as the object, `mul_proj` as its orthogonality law and `sum_proj` as its complete resolution of the identity. In a matrix realization the first law is $P_i P_j = \delta_{ij} P_i$. The second is $\sum_i P_i = 1$. No commutative multiplication assumption is present: matrix multiplication remains operator composition.

Structures used later have a different role. `OrthogonalSyndromes` stores an already normalized orthogonal error family, and `KnillLaflamme.Data` stores a general positive Gram matrix. `Data.diagonal` constructs diagonal error data. Its active positive directions are then normalized to construct `OrthogonalSyndromes`. Sufficiency assumes the existence of the general Gram data. Necessity constructs such

data from a recovery. The final equivalence joins both directions. The correct reading keeps these roles separate rather than interpreting every structure field as a fact proved from scratch.

1.4 How to read a proof script

The scripts repeatedly use a small set of proof moves. `rw` replaces an expression using a stated equality. `simp only` performs a controlled reduction with an explicit list of lemmas. The broader `simp` tactic also uses the available simplification database. `ext i j` reduces a matrix equality to equality of each entry. `funext` proves equality of functions by comparing values. `constructor` starts the two obligations of a conjunction or equivalence. `rintro` decomposes quantified and existential data. `obtain` names an existential witness already proved to exist. `refine` fixes the overall shape of a term and leaves specified holes to prove.

The tactic `noncomm_ring` normalizes ring expressions without commuting factors. It is appropriate for identities involving $P_b U P_a$. Ordinary `ring` is used for scalar arithmetic or commutative coefficient expressions. A normalization step must therefore be read in the ring where it is used. It does not permit arbitrary matrix factors to be reordered.

A line beginning `have` proves or names a local intermediate assertion. The goal remains the same until that assertion is used. A `calc` block presents a chain of equalities or inequalities. Each link is a separate obligation. Induction on a list proves all finite circuit depths, including the empty list. A finite example verified by `norm_num` or `fin_cases` proves that concrete example, not a theorem in arbitrary dimension.

For a derivative identity the important move is often `HasDerivAt.unique`: two proven derivatives of the same function at a point must coincide. The script first differentiates a product, then rewrites the product as a constant or as a known function, then invokes uniqueness. Thus the differentiated projector and unitary identities are derived from actual functions and their derivatives, rather than postulated as unrelated data of first order.

1.5 Scope is a part of the theorem

The finite results have broad mathematical scope, but their hypotheses also place clear limits on their use. A statement about the span of an algebra orbit permits every matrix in that algebra. Reachability under a physically specified set of unitary gates requires a separate argument. A theorem establishing complete positivity for finite channels quantifies over every finite reference index. Arbitrary infinite reference Hilbert spaces lie outside those declarations. A derivative computation under supplied smoothness does not prove the existence of a Schrödinger solution for every initial condition.

The modules for the projected connection do use arbitrary smooth real parameter manifolds and complete normed model spaces, which is broader than a single finite-dimensional chart. This generality of the mathematical parameter space is distinct from the paper's unproved continuum physics conjectures. Neither the word "manifold" nor an estimate in an abstract C-star algebra supplies a continuum tower, a physical constraint algebra, a renormalized limit or a Lorentzian emergence theorem.

Read each conclusion under its actual hypotheses and use its source locator to recover the surrounding assumptions. Source identity and excerpt fidelity are current documentation checks. Saved compiler and axiom transcripts are inherited proof evidence. This distinction continues through the replay chapter and full result map.

2 Navigation through the traditional paper

The final traditional volume has 159 physical PDF pages. Printed pages are the labels in that volume. PDF pages count the cover and Roman opening pages as well. Every location below was resolved from the actual body page label and checked against the source map. A shared subject does not imply identical formal coverage.

2.1 All 19 finite items in the traditional core

Item	Number	Core statement	Printed	PDF
2.1	I.2.1	Orbit dimension of one quantum seed	5	13
2.2	I.2.2	Quantum seed-rank formula	5	13
2.3	I.2.3	Local orbit and Schmidt rank	6	14
2.4	I.2.4	One seed for the computational basis	6	14
3.1	I.3.1	Cyclic quantum projectors	7	15
3.2	I.3.2	Tensor/polyphase refinement	8	16
4.1	I.4.1	Exact circuit path expansion	9	17
4.2	I.4.2	Packetwise unitarity conservation	9	17
5.1	I.5.1	Exact symmetry packets	10	18
5.2	I.5.2	Commutator leakage bound	11	19
6.1	I.6.1	Moving-frame Hamiltonian	12	20
6.3	I.6.3	Canonical packet connection	13	21
6.4	I.6.4	Projected connection	14	22
7.1	I.7.1	Block-diagonal measurement invariance	17	25
7.2	I.7.2	Finite Kraus channel	17	25
8.1	I.8.1	Finite Knill–Laflamme criterion	19	27
9.1	I.9.1	Exact Feshbach resolvent	21	29
10.1	I.10.1	Finite Euclidean transfer	24	32
11.1	I.11.1	Low-mode second-order bound	26	34

2.2 Strengthened statements and their formal counterparts

This index includes all 79 named statements with proofs in strengthened Part II. A relation identifies shared ingredients or a shared finite result. It does not claim that the whole stronger statement has a dedicated Lean theorem. Entries without a corresponding named endpoint remain written mathematical statements. The formal ledger still contains 19 items, and the declaration inventory still contains 323 entries.

Number (page)	Strengthened statement	Relation to the current formal source	PDF
II.2.2 (36)	Orbit dimension of one seed	Item 2.1 gives the same orbit dimension result.	44
II.2.3 (36)	Quantum seed rank	Item 2.2 gives the same sharp minimum number of seeds.	44
II.2.4 (36)	Local orbit and Schmidt rank	Item 2.3 relates the local orbit to reduced support.	44

Number (page)	Strengthened statement	Relation to the current formal source	PDF
II.2.5 (37)	Computational address orbit	Item 2.4 constructs the basis by addressed local X gates.	45
II.3.1 (38)	Finite group isotypic projectors	The traditional statement has no corresponding named endpoint among the current 25 modules.	46
II.3.2 (38)	Representation matrix units	The traditional statement has no corresponding named endpoint among the current 25 modules.	46
II.3.3 (39)	Cyclic charge projectors	Item 3.1 formalizes the cyclic specialization.	47
II.3.5 (39)	Charge selection rule	The traditional statement has no corresponding named endpoint among the current 25 modules.	47
II.3.6 (39)	Charge addition	The traditional statement has no corresponding named endpoint among the current 25 modules.	47
II.3.7 (40)	Tensor charge refinement	Item 3.2 formalizes finite grouping of tensor packets.	48
II.3.8 (40)	Tensor selection rule	The traditional statement has no corresponding named endpoint among the current 25 modules.	48
II.4.1 (41)	Exact circuit path expansion	Item 4.1 covers every finite circuit depth.	49
II.4.2 (41)	Packet unitarity identities	Item 4.2 gives both operator unitarity identities.	49
II.4.3 (41)	Probability conservation from one packet	Item 4.2 gives the operator identity. The probability corollary is proved in the traditional text.	49
II.4.4 (42)	Path interference formula	The traditional statement has no corresponding named endpoint among the current 25 modules.	50
II.4.5 (42)	Charge pruning of circuit paths	The traditional statement has no corresponding named endpoint among the current 25 modules.	50
II.5.1 (43)	Exact invariant packets	Item 5.1 proves packet invariance for the exponential of a fixed generator.	51
II.5.2 (43)	Off diagonal norm	The traditional statement has no corresponding named endpoint among the current 25 modules.	51
II.5.3 (44)	Fixed packet leakage	Item 5.2 gives the commutator leakage bound. The norm equality for the off-diagonal corner is proved in the traditional text.	52
II.5.4 (44)	Leakage probability	The traditional statement has no corresponding named endpoint among the current 25 modules.	52
II.6.1 (45)	Canonical packet connection	Item 6.3 gives the canonical connection identities along a curve.	53
II.6.2 (45)	Exact Kato transport	The traditional statement has no corresponding named endpoint among the current 25 modules.	53
II.6.3 (46)	Moving frame Hamiltonian	Item 6.1 proves the differentiated equation in moving coordinates.	54
II.6.4 (46)	Moving packet leakage	The traditional statement has no corresponding named endpoint among the current 25 modules.	54

Number (page)	Strengthened statement	Relation to the current formal source	PDF
II.6.5 (47)	Direct packet rotation	The traditional statement has no corresponding named endpoint among the current 25 modules.	55
II.6.6 (47)	Convergence of short packet rotations	The traditional statement has no corresponding named endpoint among the current 25 modules.	55
II.6.7 (49)	Nonabelian packet connection	Item 6.4 gives projected manifold curvature and gauge gluing. The local formulas in orthonormal frames are proved in the traditional text.	57
II.6.8 (49)	Bloch sphere curvature	The traditional statement has no corresponding named endpoint among the current 25 modules.	57
II.7.2 (50)	Complete operator packet system	The traditional statement has no corresponding named endpoint among the current 25 modules.	58
II.7.4 (50)	Dephasing as conditional expectation	The traditional statement has no corresponding named endpoint among the current 25 modules.	58
II.7.5 (51)	Block diagonal measurement invariance	Item 7.1 proves trace invariance for block diagonal observables.	59
II.7.6 (52)	Exact channel block calculus	The traditional statement has no corresponding named endpoint among the current 25 modules.	60
II.7.8 (53)	Coherence order projectors	The traditional statement has no corresponding named endpoint among the current 25 modules.	61
II.7.9 (53)	Covariant channels preserve coherence order	The traditional statement has no corresponding named endpoint among the current 25 modules.	61
II.7.10 (53)	Choi symmetry criterion	The traditional statement has no corresponding named endpoint among the current 25 modules.	61
II.7.11 (54)	Choi packet decomposition	The traditional statement has no corresponding named endpoint among the current 25 modules.	62
II.7.14 (55)	Charge preservation by a Lindblad generator	The traditional statement has no corresponding named endpoint among the current 25 modules.	63
II.7.15 (55)	Independent coherence evolution	The traditional statement has no corresponding named endpoint among the current 25 modules.	63
II.8.2 (56)	Knill Laflamme as obstruction vanishing	Item 8.1 gives the criterion for unnormalized error families and recovery on their span. The traditional text also uses a quotient formulation.	64
II.8.3 (57)	Orthogonal syndrome construction	Item 8.1 reduces general positive Gram data by spectral diagonalization.	65
II.8.4 (57)	Exact recovery on the error span	Item 8.1 uses one total recovery for normalized channels in the error span.	65
II.8.5 (58)	Syndrome measurement preserves logical information	The identities for syndrome cross errors support item 8.1. The traditional text supplies the measurement interpretation.	66

Number (page)	Strengthened statement	Relation to the current formal source	PDF
II.8.7 (58)	Meaning of the approximate defect	The traditional statement has no corresponding named endpoint among the current 25 modules.	66
II.9.2 (60)	Exact projected resolvent	Item 9.1 gives the projected resolvent with both required invertibility hypotheses.	68
II.9.3 (60)	Spectral equivalence	Item 9.1 relates invertibility of the full matrix and its Schur complement. The traditional text adds spectral and eigenvector formulations.	68
II.9.4 (61)	Upper half plane sign	The traditional statement has no corresponding named endpoint among the current 25 modules.	69
II.9.5 (61)	Uniform self energy bounds	The traditional statement has no corresponding named endpoint among the current 25 modules.	69
II.9.6 (62)	Low energy expansion	The traditional statement has no corresponding named endpoint among the current 25 modules.	70
II.9.7 (62)	Bare truncation error identity	The traditional statement has no corresponding named endpoint among the current 25 modules.	70
II.9.8 (63)	Exact retained memory equation	The traditional statement has no corresponding named endpoint among the current 25 modules.	71
II.9.9 (63)	Positive definite quantum memory	The traditional statement has no corresponding named endpoint among the current 25 modules.	71
II.9.10 (63)	Finite oscillatory memory expansion	The traditional statement has no corresponding named endpoint among the current 25 modules.	71
II.9.11 (64)	Exact local closure criterion	The traditional statement has no corresponding named endpoint among the current 25 modules.	72
II.9.12 (65)	Stability of the self energy	The traditional statement has no corresponding named endpoint among the current 25 modules.	73
II.9.13 (65)	Uniform Feshbach control	The traditional statement has no corresponding named endpoint among the current 25 modules.	73
II.10.1 (66)	Finite Euclidean transfer	Item 10.1 constructs the shifted exponential and proves its positivity and spectral bounds under the stated hypotheses.	74
II.10.2 (66)	Hamiltonian reconstruction from transfer	The traditional statement has no corresponding named endpoint among the current 25 modules.	74
II.10.3 (66)	Finite reflection positivity	The traditional statement has no corresponding named endpoint among the current 25 modules.	74
II.10.4 (67)	Reflection positive correlators	The traditional statement has no corresponding named endpoint among the current 25 modules.	75

Number (page)	Strengthened statement	Relation to the current formal source	PDF
II.10.5 (67)	Ground packet convergence	The traditional statement has no corresponding named endpoint among the current 25 modules.	75
II.10.6 (67)	Symmetry resolved transfer	The traditional statement has no corresponding named endpoint among the current 25 modules.	75
II.11.2 (68)	Finite scale limits	The traditional statement has no corresponding named endpoint among the current 25 modules.	76
II.11.3 (68)	Heat packet decomposition	The traditional statement has no corresponding named endpoint among the current 25 modules.	76
II.11.4 (69)	Low mode dispersion in dimension d	Item 11.1 gives the scalar bound in one dimension. The traditional text proves the summed result in d dimensions.	77
II.11.5 (69)	Fixed time heat trace convergence	The traditional statement has no corresponding named endpoint among the current 25 modules.	77
II.11.7 (70)	Continuum torus small time dimension	The traditional statement has no corresponding named endpoint among the current 25 modules.	78
II.11.8 (70)	Rigorous intermediate dimension window	The traditional statement has no corresponding named endpoint among the current 25 modules.	78
II.12.2 (71)	Finite constraint closure criterion	The traditional statement has no corresponding named endpoint among the current 25 modules.	79
II.12.3 (72)	Physical compression is not a closure test	ConstraintAudit formalizes finite compression to the common kernel.	80
II.12.4 (72)	Invariance of the physical space	The traditional statement has no corresponding named endpoint among the current 25 modules.	80
II.12.5 (73)	Exact constraint transport	The traditional statement has no corresponding named endpoint among the current 25 modules.	81
II.12.6 (73)	Approximate constraint transport	The traditional statement has no corresponding named endpoint among the current 25 modules.	81
II.13.2 (75)	Bounded Hamiltonian direct limit	The traditional statement has no corresponding named endpoint among the current 25 modules.	83
II.13.3 (75)	Exact packet direct limit	The traditional statement has no corresponding named endpoint among the current 25 modules.	83
II.13.4 (76)	Norm summable Hamiltonian tower	The traditional statement has no corresponding named endpoint among the current 25 modules.	84
II.13.5 (76)	Limit of packet leakage	The traditional statement has no corresponding named endpoint among the current 25 modules.	84

Number (page)	Strengthened statement	Relation to the current formal source	PDF
II.13.6 (77)	Reflection positivity is closed under limits	The traditional statement has no corresponding named endpoint among the current 25 modules.	85
II.13.7 (77)	Constraint closure under norm convergence	The traditional statement has no corresponding named endpoint among the current 25 modules.	85
II.13.8 (77)	Heat trace convergence from trace norm	The traditional statement has no corresponding named endpoint among the current 25 modules.	85

2.3 Retained supplements explain the constructed proofs

Part III retains the mathematical portions of all six formal supplements. Read their historical descriptions in order: later constructions strengthen earlier restricted coverage. The endpoint map ties each current result to its source.

Sections	Retained proof material	Printed	PDF
III.1–III.5	This supplement develops packet algebra, symmetry and measurements. It states the corrected resolvent and spacing hypotheses and audits finite constraints.	81–85	89–93
III.6–III.10	This supplement constructs an addressed computational seed, tensor refinements, moving coordinates, the canonical connection and Kraus amplification.	86–90	94–98
III.11–III.14	This supplement derives orbit spaces and sharp seed counts. It defines Schmidt rank independently and develops cyclic characters and the Euclidean exponential.	91–94	99–102
III.15–III.17	This supplement proves leakage for the exponential, projected curvature in a chart and total recovery from orthogonal syndromes.	95–97	103–105
III.18–III.23	This supplement proves spectral reduction for general Gram data, necessity from recovery, the bridge from states to operators, and correction with finite references and error spans.	98–104	106–112
III.24–III.27	This supplement constructs local products for the range, smooth transitions and the global connection. It proves the gauge law and manifold curvature formula.	104–106	112–114

3 Seeds, local orbits, and the packet resolution

3.1 One seed: identify the actual image before counting it

The starting coefficient array is $C \in \mathrm{Mat}_{d\times m}(\mathbb{C})$. Left multiplication by an arbitrary $d\times d$ matrix changes each row independently within the row space of C. This observation gives the orbit dimension formula. Lean first defines the row space as the range of the linear map given by right multiplication. It then defines the orbit as the range of that map acting on all d rows. The dimension is derived from this construction.

Contract. The row and multiplicity index types `d` and `m` have finite enumerations. The orbit under the algebra action is $\{AC : A \in \mathrm{Mat}_{d\times d}(\mathbb{C})\}$, which is already a linear subspace. The dimension formula permits an empty row type. The equivalence characterizing a full orbit adds `[Nonempty d]` because it cancels the positive row cardinality.

The lemma `mem_orbit_iff_rows` supplies both directions. If $X = AC$, each row of X is a linear combination of rows of C. Conversely, choose one coefficient row for each row of X, collect them into a matrix A, and obtain $AC = X$. The latter choice is an existence argument using classical choice. It is not a numerical algorithm for choosing coefficients.

The explicit linear equivalence below expresses this correspondence for each row. Its domain is the subspace of reachable coefficient arrays. Its codomain is the function space whose value at each row is a member of the row space. Both inverse laws reduce to the underlying functions.

Source: `formalization/SeedOrbit.lean`.

Exact lines 45 to 52. Excerpt `row_equiv`. Hash in Section 12.

```
noncomputable def orbitRowEquiv (C : Matrix d m ℂ) :
    orbitSpace C ≃ₗ[ℂ] (d → rowSpace C) where
  toFun X i := ⟨X.val i, (mem_orbit_iff_rows C X.val).mp X.property i⟩
  invFun f := ⟨fun i => (f i).val, (mem_orbit_iff_rows C _).mpr (fun i => (f i).property)⟩
  left_inv _ := rfl
  right_inv _ := rfl
  map_add' _ _ := rfl
  map_smul' _ _ := rfl
```

Proof reading. The `toFun` field extracts the rows and attaches the proof that each row belongs to the row space. The `invFun` field assembles those rows, using the reverse membership implication. The equalities of functions are definitional, so the inverse and linearity proofs are `rfl`. Taking dimensions of this equivalence gives $|d|$ rank C. The extra `orbit_span` lemma identifies the subspace with the span of the literal action set.

Source: `formalization/SeedOrbit.lean`.

Exact lines 68 to 71. Excerpt `orbit_dimension`. Hash in Section 12.

```
theorem span_orbit_dimension (C : Matrix d m ℂ) :
    finrank ℂ (Submodule.span ℂ (Set.range fun A : Matrix d d ℂ => A * C)) =
      Fintype.card d * C.rank := by
  rw [orbit_span, orbit_dimension]
```

Source: `formalization/SeedOrbit.lean`.

Exact lines 73 to 80. Excerpt `cyclic_rank`. Hash in Section 12.

```
theorem cyclic_iff_full_rank [Nonempty d] (C : Matrix d m ℂ) :
    orbitSpace C = ⊤ ↔ C.rank = Fintype.card m := by
  rw [Submodule.eq_top_iff_finrank_eq, orbit_dimension]
  have hdim : finrank ℂ (Matrix d m ℂ) = Fintype.card d * Fintype.card m := by
    change finrank ℂ (d → m → ℂ) = _
    simp [Module.finrank_pi_fintype]
  rw [hdim]
  exact Nat.mul_left_cancel_iff (Fintype.card_pos)
```

The nonempty condition deserves attention. If there are no row indices, the ambient coefficient space is zero and every orbit is already full, regardless of the multiplicity index. Cancelling $|d|$ would be invalid. Lean's final cancellation explicitly uses `Fintype.card_pos`. This records why the test for full rank needs an inhabitant. Multiplicity zero is permitted and gives a target of dimension zero.

3.2 Several seeds: stack them and prove a minimum

Continue with a nonempty row index type. For k seeds C_a, the allowed generation is

$$X = \sum_{a \in k} A_a C_a,$$

where the algebra element A_a may differ for each seed. The question is whether this family generates the whole space linearly. The implementation records its universal quantifier over X and its existential family of coefficient matrices.

Source: `formalization/SeedOrbit.lean`.

Exact lines 127 to 128. Excerpt `seed_generate`. Hash in Section 12.

```
def Generates (C : k → Matrix d m ℂ) : Prop :=
  ∀ X : Matrix d m ℂ, ∃ A : k → Matrix d d ℂ, ∑ a, A a * C a = X
```

Source: `formalization/SeedOrbit.lean`.

Exact lines 148 to 150. Excerpt `stack_rank`. Hash in Section 12.

```
theorem generates_iff_rank [Nonempty d] (C : k → Matrix d m ℂ) :
    Generates C ↔ (stackSeeds C).rank = Fintype.card m := by
  rw [generates_iff_rowOrbit, rowOrbit_full]
```

Stacking the seeds gives one matrix with row index $k \times d$. Its row space is the span of all the seed rows. The lemma `stacked_action` proves entrywise that multiplication by the large coefficient matrix is exactly the displayed sum. The rank test therefore says that the stacked seed matrix has full column rank. A rank upper bound gives the necessary inequality $|m| \leq |k||d|$.

The reverse direction is constructive at the matrix level. When the number of columns does not exceed the available row slots, choose an injection of column indices into row indices and place a 1 at each corresponding coordinate, with zeros elsewhere. The associated multiplication map is injective because its selected output coordinates reproduce the original input coordinates. The rank theorem follows from that injectivity.

Source: `formalization/SeedOrbit.lean`.

Exact lines 158 to 171. Excerpt `full_column`. Hash in Section 12.

```
theorem exists_full_column_rank (h : Fintype.card m ≤ Fintype.card d) :
    ∃ C : Matrix d m ℂ, C.rank = Fintype.card m := by
  classical
  obtain ⟨f⟩ := Function.Embedding.nonempty_of_card_le h
  let C : Matrix d m ℂ := fun i j => if i=f j then 1 else 0
  have hC (x : m → ℂ) (j : m) : (C *ᵥ x) (f j) = x j := by
    simp [C, Matrix.mulVec, dotProduct, f.injective.eq_iff]
  have hi : Function.Injective C.mulVecLin := by
    intro x y hxy
    funext j
    have h := congrFun hxy (f j)
    simpa only [Matrix.mulVecLin_apply, hC] using h
  refine ⟨C, ?_⟩
  rw [Matrix.rank, LinearMap.finrank_range_of_inj hi, Module.finrank_pi]
```

For a finite family of direct summands, `dd mm : I → ℕ` gives the matrix row dimension and multiplicity of each block. A seed now contains one coefficient array in every block. `BlockGenerates` quantifies over simultaneous block targets and permits simultaneous block algebra elements. The equivalence `block_generates_iff` proves that this is exactly the collection of individual generation tests. It does not replace the simultaneous problem by an unproved heuristic.

Contract. Every row dimension `dd i` is positive. Multiplicities `mm i` may vanish. The block index `I` is any finite type, including the empty type. The formula is the finite supremum of ceiling divisions in the natural numbers, so an empty block family gives zero.

Define

$$r = \max_{i \in I} \left\lceil \frac{m_i}{d_i} \right\rceil .$$

The source writes this as `Finset.univ.sup` with natural ceiling division. The lemma `seed_formula_le_iff` translates $r \leq s$ into all inequalities $m_i \leq s d_i$. The existence equivalence constructs seeds exactly when those inequalities hold. The final result is `IsLeast`: it proves both that r seeds exist and that every successful seed count is at least r.

Source: `formalization/SeedOrbit.lean`.

Exact lines 239 to 246. Excerpt `seed_minimum`. Hash in Section 12.

```
theorem quantum_seed_rank (dd mm : I → ℕ) (hd : ∀ i, 0 < dd i) :
    IsLeast {s : ℕ | ∃ C : Fin s → (i : I) → Matrix (Fin (dd i)) (Fin (mm i)) ℂ,
      BlockGenerates dd mm C} (seedRankFormula dd mm) := by
  constructor
  · exact (exists_block_seeds_iff dd mm hd _).mpr
      ((seed_formula_le_iff dd mm hd _).mp le_rfl)
  · intro s hs
    exact (seed_formula_le_iff dd mm hd s).mpr ((exists_block_seeds_iff dd mm hd s).mp hs)
```

Proof reading. The first half of the pair uses `le_rfl` to show the formula satisfies its own inequalities, then applies the proved existence criterion. The second half takes any witness at count s, converts its existence to the inequalities, then converts those inequalities back to $r \leq s$. Thus minimality and attainability are both present. The conclusion is stronger than merely displaying a candidate formula.

3.3 Schmidt rank is independently defined

For a bipartite vector $\psi : d \times m \to \mathbb{C}$, `coefficientEquiv` identifies $\psi(i, j)$ with the coefficient matrix C_{ij}. The local action $A \otimes 1$ becomes AC in those standard product coordinates. This identity is proved by expanding the Kronecker product and summing over the product index.

Source: `formalization/SeedOrbit.lean`.

Exact lines 261 to 265. Excerpt `local_action`. Hash in Section 12.

```
theorem local_tensor_action [DecidableEq m] (A : Matrix d d ℂ) (psi : d × m → ℂ) :
    coefficientEquiv ((A ⊗ₖ (1 : Matrix m m ℂ)) *ᵥ psi) = A * coefficientEquiv psi := by
  ext i j
  simp [coefficientEquiv, Matrix.mulVec, dotProduct,
    Fintype.sum_prod_type, Matrix.one_apply, Matrix.mul_apply]
```

The source defines Schmidt rank through product decompositions before proving that it agrees with matrix rank. `HasProductDecomposition C s` means an actual representation

$$C_{ij} = \sum_{a<s} u_a(i) v_a(j).$$

Such a representation factors C through an s-dimensional coordinate space, so rank $C \leq s$. Conversely, choose a basis of the row space and express each row in that basis. The basis vectors of the row space and their coordinate functions give a product decomposition of length exactly rank C. These two directions prove that matrix rank is the least possible product length.

Source: `formalization/SeedOrbit.lean`.

Exact lines 298 to 300. Excerpt `schmidt_minimum`. Hash in Section 12.

```
theorem product_length_minimum (C : Matrix d m ℂ) :
    IsLeast {s : ℕ | HasProductDecomposition C s} C.rank := by
  exact ⟨product_decomposition_at_rank C, fun _ h => product_decomposition_rank_bound C h⟩
```

Source: `formalization/SeedOrbit.lean`.

Exact lines 303 to 306. Excerpt `schmidt_definition`. Hash in Section 12.

```
noncomputable def schmidtRank (psi : d × m → ℂ) : ℕ := by
  classical
  exact Nat.find (show ∃ s, HasProductDecomposition (coefficientEquiv psi) s from
    ⟨_, product_decomposition_at_rank _⟩)
```

`Nat.find` takes the least natural number for which a product decomposition exists. The existence needed by `Nat.find` has already been proved at the rank. The equality `schmidt_rank_eq_matrix_rank` then uses the upper witness and the rank lower bound in opposite directions. It is an equality between independently introduced objects.

The reduced B matrix is also constructed explicitly:

$$(\rho_B)_{j\ell} = \sum_i \psi(i, j)\overline{\psi(i, \ell)}.$$

In the chosen convention it is $C^T(C^T)^*$. Its positivity follows from this Gram representation. Its rank agrees with the matrix rank of C, hence with Schmidt rank. Full support is expressed as surjectivity of its multiplication linear map. Combining this with the local orbit dimension gives the endpoint below.

Source: `formalization/SeedOrbit.lean`.

Exact lines 364 to 366. Excerpt `reduced_support`. Hash in Section 12.

```
theorem local_orbit_reduced_support [Nonempty d] (psi : d × m → ℂ) :
    localOrbit psi = ⊤ ↔ LinearMap.range (reducedB psi).mulVecLin = ⊤ := by
  rw [local_orbit_full_iff, reduced_full_support_iff]
```

Contract. The equivalence between a full local orbit and full reduced support assumes `[Nonempty d]`. The dimension equality itself includes zero vectors and empty index types. No normalization of ψ is required for these algebraic rank statements. The action uses every matrix A, so the result concerns the linear span under the local operator algebra. It does not assert reachability under an arbitrary restricted family of unitary controls.

3.4 Addressed bit flips give a computational basis

`Address n` is defined as `Fin n → ZMod 2`, the finite set of bit strings. `State n` is the complex vector space of functions on that address set. The seed is the coordinate vector at the zero address. The linear operator `translation a` evaluates a vector at $x - a$. On a ket it gives $|b\rangle \mapsto |a + b\rangle$.

Each `localX j` translates by the bit vector supported at coordinate j. The operator named `controlled X a j` in this module is selected by the classical address bit a_j: it is identity when that bit is zero, and the local X otherwise. This definition selects a gate according to the address. Despite its name, it does not separately formalize the controlled-NOT gate on two qubits.

The address circuit is the product of this finite gate list. Translation products add their addresses, and the sum of the local address contributions equals a. Therefore the circuit translates by a and sends the zero seed to the coordinate basis vector at a.

Source: `formalization/ComputationalSeed.lean`.

Exact lines 72 to 77. Excerpt `seed_translation`. Hash in Section 12.

```
theorem circuit_eq_translation {n : ℕ} (a : Address n) :
    addressCircuit a = translation a := by
  unfold addressCircuit
  simp_rw [controlledX_eq_translation]
  simpa only [List.map_ofFn, Function.comp_def, List.sum_ofFn, sum_local_addresses] using
    translation_list_product (List.ofFn (fun j : Fin n => Pi.single j (a j)))
```

Source: `formalization/ComputationalSeed.lean`.

Exact lines 80 to 82. Excerpt `seed_basis`. Hash in Section 12.

```
theorem circuit_seed_basis {n : ℕ} (a : Address n) :
    addressCircuit a (seed n) = Pi.basisFun ℂ (Address n) a := by
  rw [circuit_seed, Pi.basisFun_apply, ket]
```

The basis theorem then supplies linear independence and full span. The cardinality theorem gives 2^n. When $n = 0$, the type of bit strings has exactly one empty address, the gate list is empty, and its product is identity. The source covers this case without assuming that the number of qubits is positive.

3.5 Cyclic projectors from powers of an element of finite order

Let $N > 0$ and $S^N = 1$. The source uses `ZMod N` as the residue type and the standard primitive additive character χ. It defines

$$P_j = \frac{1}{N} \sum_{r \in \mathbb{Z}/N} \chi(-rj) S^{r.\mathrm{val}}.$$

The representation `cyclicPowers S` is constructed from the actual powers of S. The proof of its addition law uses $S^N = 1$ to reduce an exponent modulo N. The root of unity is identified with $\exp(2\pi i/N)$, and the coefficient lemma recovers the familiar inverse power ω^{-rj}.

Source: `formalization/CyclicProjectors.lean`.

Exact lines 22 to 26. Excerpt `cyclic_character`. Hash in Section 12.

```
theorem character_sum (t : ZMod N) :
    ∑ r : ZMod N, stdAddChar (t*r) = if t=0 then (N : ℂ) else 0 := by
  split_ifs with h
  · simp [h]
  · exact sum_eq_zero_of_ne_one (isPrimitive_stdAddChar N h)
```

The character sum provides the cancellation needed in the proof. When its argument is zero, the sum is N. When the argument is nonzero, primitivity makes the sum vanish. To prove multiplication of packets, first show that left multiplication by the representation acts on a packet by its character. Substitute that eigenaction into the packet product and collapse the remaining character sum.

Source: `formalization/CyclicProjectors.lean`.

Exact lines 44 to 58. Excerpt `cyclic_mul`. Hash in Section 12.

```
theorem packet_mul (rho : ZMod N → A)
    (hadd : ∀ a b, rho (a+b) = rho a * rho b) (j k : ZMod N) :
    packet rho j * packet rho k = if j=k then packet rho j else 0 := by
  conv_lhs => lhs; unfold packet
  rw [Algebra.smul_mul_assoc, Finset.sum_mul]
  simp only [Algebra.smul_mul_assoc, packet_eigen rho hadd, smul_smul,
    ← map_add_eq_mul]
  have hf (r : ZMod N) : -(r*j) + r*k = (k-j)*r := by ring
  simp only [hf]
  rw [← Finset.sum_smul, character_sum]
  by_cases h : j=k
  · subst k
    simp [NeZero.ne N]
  · have hk : k-j ≠ 0 := sub_ne_zero.mpr (Ne.symm h)
    simp [hk, h]
```

Completeness follows from exchanging the two finite sums and applying the same orthogonality in the other index. Adjointness requires a separate argument in a star ring: conjugate the character, replace r by $-r$, and use the adjoint law for unitary powers. An element of finite order gives the idempotent decomposition. Unitarity supplies the Hermitian condition in the theorem for complex matrices.

Source: formalization/CyclicProjectors.lean.

Exact lines 157 to 170. Excerpt cyclic_matrix. Hash in Section 12.

```
theorem finite_cyclic_projectors {n : Type*} [Fintype n] [DecidableEq n]
    (S : Matrix n n ℂ) (hS : S^N=1) (hU : S ∈ unitary (Matrix n n ℂ)) :
    (∀ j k : ZMod N, packet (cyclicPowers S) j * packet (cyclicPowers S) k =
      if j=k then packet (cyclicPowers S) j else 0) ∧
    (∑ j : ZMod N, packet (cyclicPowers S) j) = 1 ∧
    (∀ j : ZMod N, S * packet (cyclicPowers S) j =
      omega (N := N)^j.val • packet (cyclicPowers S) j) ∧
    ∀ j : ZMod N, (packet (cyclicPowers S) j).IsHermitian := by
  obtain ⟨hp, hs, he⟩ := cyclic_quantum_projectors S hS
  refine ⟨hp, hs, ?_, ?_⟩
  · intro j
    simpa only [character_as_power] using he j
  · intro j
    exact cyclic_packet_selfAdjoint S hS hU j
```

Contract. The ambient general construction is a complex algebra over a ring. The concrete endpoint has a finite matrix state index and decidable equality. `[NeZero N]` excludes zero modulus, includes $N = 1$, and justifies division by N. The theorem does not require every packet to have nonzero rank.

3.6 Tensor factors and regrouping by residue

The tensor implementation uses a finite factor index `F`, with a potentially different state index `D i` in every factor. A tensor matrix is defined by the product of its local entries. Multiplication expands into a sum over dependent tuples of intermediate indices. The finite identity expressing a product of sums converts that sum back into the product of local matrix multiplications.

Source: formalization/TensorRefinement.lean.

Exact lines 20 to 27. Excerpt tensor_mul. Hash in Section 12.

```
theorem tensor_mul (A B : (i : F) → Matrix (D i) (D i) ℂ) :
    tensor (fun i => A i * B i) = tensor A * tensor B := by
  ext x y
  simp only [tensor, Matrix.mul_apply]
  rw [Fintype.prod_sum]
  apply Finset.sum_congr rfl
  intro z _
  exact Finset.prod_mul_distrib
```

The product formula, identity formula, adjoint formula and sum formula then prove that the fine packets indexed by tuples are mutually orthogonal, complete and Hermitian. The coarse construction sums fine packets with the same address. The general proof over a ring shows that grouping an orthogonal resolution by any finite address map remains an orthogonal resolution. For polyphase grouping, the address is the sum of the factor residues in `ZMod N`.

Source: `formalization/TensorRefinement.lean`.

Exact lines 139 to 154. Excerpt `tensor_refinement`. Hash in Section 12.

```
theorem tensor_polyphase_refinement {F : Type*} [Fintype F] [DecidableEq F] {D : F → Type*}
    [∀ i, Fintype (D i)] [∀ i, DecidableEq (D i)]
    (N : ℕ) [NeZero N] (P : (i : F) → ZMod N → Matrix (D i) (D i) ℂ)
    (hmul : ∀ i j k, P i j * P i k = if j = k then P i j else 0)
    (hsum : ∀ i, ∑ j, P i j = 1)
    (hstar : ∀ i j, (P i j).conjTranspose = P i j) :
    let Q := fun a : F → ZMod N => tensor (fun i => P i (a i))
    let C := coarse Q (fun a => ∑ i, a i)
    (∀ r s, C r * C s = if r = s then C r else 0) ∧
      (∑ r, C r) = 1 ∧ (∀ r, (C r).conjTranspose = C r) := by
  classical
  dsimp only
  refine ⟨?_, ?_, ?_⟩
  · exact coarse_mul _ _ (fine_mul P hmul)
  · exact coarse_complete _ _ (fine_complete P hsum)
  · exact coarse_selfAdjoint _ _ (fine_selfAdjoint P hstar)
```

Proof reading. The two `let` bindings expose the fine family Q_a and the coarse family C_r inside the statement. After `dsimp only`, each conclusion is a direct application of the generic coarse theorem to a theorem already proved for the fine family. The code therefore separates tensor construction from finite grouping, which makes the proof reusable.

The input here is a packet resolution in each tensor factor. The preceding theorem can construct these local packets from each factor's cyclic symmetry. The tensor theorem takes the resulting local resolutions as input. Together the two constructions implement the traditional passage from cyclic local packets to a tensor refinement and packets grouped by total residue.

4 Circuit corners, conserved packets, and leakage

4.1 The order of the indices is part of the definition

For a packet resolution P, the corner of an operator is

$$U_{ba} = P_b U P_a.$$

The target index b comes first, and the source index a comes second. Matrix factors act from the right on a vector. The gate list in `QuantumPacketCore` is written latest to earliest, $[U_m, \dots, U_1]$, so its product is $U_m \cdots U_1$. The source comments and definitions make both conventions explicit. They determine the orientation of every path.

Contract. The basic corner identities require an arbitrary ring `R` and a finite packet index with decidable equality. They require the packet multiplication and completeness laws, but neither an adjoint nor a norm. The coefficients are generally noncommutative.

Inserting $\sum_c P_c = 1$ between two operators gives

$$(UV)_{ba} = \sum_c U_{bc} V_{ca}.$$

The code first rewrites an individual product of corners. The adjacent $P_c P_c$ collapses by idempotence, while all other factors keep their order. Only then does it combine the finite sum and use completeness.

Source: `formalization/QuantumPacketCore.lean`.

Exact lines 41 to 50. Excerpt `compose_blocks`. Hash in Section 12.

```
theorem compose_blocks (U V : R) (b a : I) :
    P.block (U * V) b a = ∑ c, P.block U b c * P.block V c a := by
  have term (c : I) : P.block U b c * P.block V c a =
      P.proj b * U * P.proj c * V * P.proj a := by
    unfold block
    calc
      _ = P.proj b * U * (P.proj c * P.proj c) * V * P.proj a := by noncomm_ring
      _ = _ := by rw [P.idempotent]
  simp_rw [term]
  simp only [← Finset.sum_mul, ← Finset.mul_sum, P.sum_proj, block, mul_assoc, one_mul]
```

This is the local mechanism for the whole path expansion. `pathSum` is an explicit recursively nested finite sum. For no gates, its value is $P_b P_a$, preserving the boundary packet. For a gate U followed by a shorter circuit, it sums U_{bc} times the shorter path from a to c.

Source: `formalization/QuantumPacketCore.lean`.

Exact lines 63 to 68. Excerpt `circuit_path`. Hash in Section 12.

```
theorem circuit_path_expansion (Us : List R) (b a : I) :
    P.block (circuit Us) b a = P.pathSum Us b a := by
  induction Us generalizing b a with
  | nil => simp [circuit, pathSum, block]
  | cons U Us ih =>
      simp only [circuit, pathSum, P.compose_blocks, ih]
```

Proof reading. The induction generalizes both boundary indices, which makes the induction hypothesis available at every intermediate c. The empty case reduces the identity corner. In the nonempty case, the lemma for composing corners creates the finite sum required by the recursive definition. The induction hypothesis then replaces each shorter corner. The resulting proof covers every finite circuit depth.

Summing the resulting path values over source and target reconstructs the complete circuit operator. It retains coherent complex amplitudes and operator products. No step converts the path sum into a stochastic transition law or assumes that different paths are mutually incoherent.

4.2 The two unitarity identities use different hypotheses

To take adjoints, supply the structure of a star ring and self-adjoint packet elements. The adjoint swaps source and target:

$$U^*_{ba} = (U^*)_{ab}.$$

For the column identity, the source needs only $U^*U = 1$:

$$\sum_b U^*_{ba} U_{bc} = \delta_{ac} P_a.$$

For the row identity, it needs $UU^* = 1$:

$$\sum_a U_{ba} U^*_{da} = \delta_{bd} P_b.$$

The source states these separately rather than hiding both inside a single unitary predicate. The distinction matters in abstract star rings where a one-sided inverse need not be two-sided.

Source: `formalization/QuantumPacketCore.lean`.

Exact lines 119 to 131. Excerpt `unitarity_columns`. Hash in Section 12.

```
theorem unitarity_columns (U : R) (hU : star U * U = 1) (a c : I) :
    (∑ b, star (P.block U b a) * P.block U b c) =
      if a = c then P.proj a else 0 := by
  have term (b : I) : star (P.block U b a) * P.block U b c =
      P.proj a * star U * P.proj b * U * P.proj c := by
    rw [P.adjoint_block hstar]
    unfold block
    calc
      _ = P.proj a * star U * (P.proj b * P.proj b) * U * P.proj c := by noncomm_ring
      _ = _ := by rw [P.idempotent]
  simp_rw [term]
  rw [← Finset.sum_mul, ← Finset.sum_mul, ← Finset.mul_sum, P.sum_proj, mul_one]
  rw [mul_assoc (P.proj a), hU, mul_one, P.mul_proj]
```

Proof reading. The local term is reorganized to $P_a U^* P_b U P_c$, using self-adjointness and $P_b^2 = P_b$. Summing over b replaces the middle resolution by the identity. The given equation $U^*U = 1$ removes the operator factors. The final packet multiplication gives the conditional expression in the conclusion. The concrete matrix wrappers merely specialize this already proved identity.

For an actual state, this identity can be combined with the inner product on the Hilbert space to interpret conservation of total packet probability. The formal statement displayed here is the operator

identity. A chosen vector, state normalization and the corresponding interpretation through the inner product are additional data in such an application.

4.3 Exact commutation propagates to the actual exponential

If H commutes with each P_a, its corners between distinct packets vanish and the diagonal corners reconstruct H. The dynamics layer uses a normed complex algebra and the exponential defined by its power series. A `Commute` proof for H and P_a propagates through complex scalar multiplication and the exponential. Thus e^{zH} is diagonal in the same packet resolution for every complex scalar z.

Source: `formalization/QuantumPacketCore.lean`.

Exact lines 168 to 171. Excerpt `symmetry_exp`. Hash in Section 12.

```
theorem symmetry_exponential_no_transfer (H : R)
    (hH : ∀ i, H * P.proj i = P.proj i * H) (z : ℂ)
    (b a : I) (hba : b ≠ a) : P.block (NormedSpace.exp (z • H)) b a = 0 :=
  P.symmetry_no_transfer _ (P.symmetry_exponential_commutes H hH z) b a hba
```

For evolution in real time, set $z = -it$. The conclusion is exact zero transfer between distinct packets. The commutation hypothesis is still required. It can be proved from a chosen symmetry or supplied as data. A packet resolution alone does not imply it.

4.4 Leakage begins with a differentiated conjugation

The quantitative result works first in an abstract complex C-star algebra. Its matrix specialization activates the operator norm on the Hilbert space through `Matrix.Norms.L2Operator`. This scope is essential: the displayed bound is not an arbitrary entrywise matrix norm estimate.

Let $G^* = -G$ and set $V(t) = e^{tG}Pe^{-tG}$. The source constructs this function and differentiates the actual exponential product:

$$V'(t) = e^{tG}(GP - PG)e^{-tG}.$$

It proves that the exponentials are unitary. Left and right multiplication by unitary elements preserve the C-star norm, so $\|V'(t)\| = \|GP - PG\|$.

Source: `formalization/LeakageBound.lean`.

Exact lines 31 to 38. Excerpt `transport_derivative`. Hash in Section 12.

```
theorem transported_derivative (G P : A) (t : ℝ) :
    HasDerivAt (transported G P)
      (exp (t • G) * (G * P - P * G) * exp (t • (-G))) t := by
  have hd := ((hasDerivAt_exp_smul_const G t).mul_const P).mul
    (hasDerivAt_exp_smul_const' (-G) t)
  convert hd using 1
  · rfl
  · noncomm_ring
```

Source: `formalization/LeakageBound.lean`.

Exact lines 46 to 54. Excerpt `transport_lipschitz`. Hash in Section 12.

```
theorem transported_lipschitz (G P : A) (hG : star G = -G) (t s : ℝ) :
    ‖transported G P t - transported G P s‖ ≤
      |t-s| * ‖G*P-P*G‖ := by
  have hh := Convex.norm_image_sub_le_of_norm_deriv_le
    (f := transported G P) (s := Set.univ)
    (fun x _ => (transported_derivative G P x).differentiableAt)
    (fun x _ => (transported_derivative_norm G P hG x).le)
    convex_univ (Set.mem_univ s) (Set.mem_univ t)
  simpa [Real.norm_eq_abs, mul_comm] using hh
```

Proof reading. The Lipschitz proof applies a mean-value norm inequality on the convex set of all real times. The existence of the derivative follows from `transported_derivative`, and its uniform derivative bound comes from `transported_derivative_norm`. Both are proved for the actual function. Taking one time to be zero gives a bound for $V(t)-P$. Multiplying by e^{tG} converts it into a bound for $e^{tG}P-Pe^{tG}$.

For an orthogonal projection,

$$(1-P)UP = (1-P)(UP-PU)P.$$

The two projection norms are at most one. Hence the leakage corner is bounded by the exponential commutator. Substitute $G=-iH$ for self-adjoint H. Its commutator norm equals the norm of $HP-PH$. This gives the concrete endpoint.

Source: `formalization/LeakageBound.lean`.

Exact lines 167 to 171. Excerpt `leakage_matrix`. Hash in Section 12.

```
theorem matrix_commutator_leakage_bound {n : Type*} [Fintype n] [DecidableEq n]
    (H P : Matrix n n ℂ) (hH : H.IsHermitian) (hP : P.IsHermitian)
    (hPP : P*P=P) (t : ℝ) :
    ‖(1-P) * exp ((-(t : ℂ)*Complex.I) • H) * P‖ ≤ |t| * ‖H*P-P*H‖ := by
  exact commutator_leakage_bound H P hH.isSelfAdjoint ⟨hPP, hP.isSelfAdjoint⟩ t
```

Contract. The matrix index may be empty. H must be Hermitian and P must be both Hermitian and idempotent. Time is every real t, including zero and negative times. The proof does not divide by projector rank, dimension or time, so it includes $P=0$ and $P=1$. The estimate concerns a fixed generator H and its exponential, rather than a general time-dependent propagator.

The same file proves a cap at one, giving $\min(1, |t|\|[H,P]\|)$, and exact zero leakage when the commutator is zero. If $H=H_0+V$ with $[H_0,P]=0$ and both terms self-adjoint, only V contributes to the commutator. The elementary estimate $\|[V,P]\| \le 2\|V\|$ gives the perturbation corollary.

Source: `formalization/LeakageBound.lean`.

Exact lines 151 to 164. Excerpt `leakage_perturbation`. Hash in Section 12.

```
theorem perturbation_leakage_bound (H₀ V P : A)
    (hH₀ : IsSelfAdjoint H₀) (hV : IsSelfAdjoint V)
```

```
    (hP : IsStarProjection P) (hcomm : H₀*P=P*H₀) (t : ℝ) :
    ‖(1-P)*exp ((-(t : ℂ)*Complex.I) • (H₀+V))*P‖ ≤ 2*|t| *‖V‖ := by
  have hc : (H₀+V)*P-P*(H₀+V) = V*P-P*V := by
    rw [add_mul, mul_add, hcomm]
    noncomm_ring
  have hh := commutator_leakage_bound (H₀+V) P (hH₀.add hV) hP t
  rw [hc] at hh
  calc
    _ ≤ |t| *‖V*P-P*V‖ := hh
    _ ≤ |t| *(2*‖V‖) := mul_le_mul_of_nonneg_left
      (projection_commutator_norm_le V P hP) (abs_nonneg t)
    _ = _ := by ring
```

4.5 Dephasing and a precisely restricted measurement invariance

The dephasing map is $\Delta(\rho) = \sum_a P_a \rho P_a$. It removes off-diagonal corners and is idempotent. For finite complex matrices, trace duality gives

$$\mathrm{tr}(\Delta(E)\rho) = \mathrm{tr}(E\Delta(\rho)).$$

It follows by expanding the finite sum and cyclically rotating matrix factors inside each trace. The subsequent measurement theorem supplies the condition $\Delta(E) = E$, which says precisely that the observable is block diagonal.

Source: `formalization/QuantumPacketCore.lean`.

Exact lines 208 to 211. Excerpt `measurement`. Hash in Section 12.

```
theorem measurement_invariance (E rho : Matrix n n ℂ) (hE : P.dephase E = E) :
    Matrix.trace (E * rho) = Matrix.trace (E * P.dephase rho) := by
  rw [← hE] at ⊢
  simpa only [P.dephase_idempotent] using dephase_trace_duality P (P.dephase E) rho
```

Neither positivity nor a trace of one for ρ is needed for this algebraic identity. Calling it a probability equality requires an appropriate state and effect in the application. Taking $E = 1$ proves trace preservation, since completeness and idempotence give $\Delta(1) = 1$. General observables with off-diagonal packet components need not satisfy the invariance premise. Dephasing may change their expectation values.

This chapter explains why the finite core begins with a resolution in a ring and only adds hypotheses when a later operation needs them. The exact circuit formula, conservation identities, exponential dynamics and leakage bound are related constructions. Each theorem states the hypotheses needed for its own conclusion.

5 Moving frames and an actual projected connection

5.1 Differentiate the coordinates before rewriting the equation

Let $U(t)$ be a unitary frame of finite complex matrices and $\psi(t)$ an actual differentiable vector. With $y(t) = U(t)^*\psi(t)$, the effective Hamiltonian is

$$H_{\text{eff}} = U^*HU - iU^*\dot{U}.$$

The source uses real time derivatives of complex-valued entries. The variables `dU` and `dpsi` are candidate derivative values, and the hypotheses `HasDerivAt` assert that they are the actual derivatives at the specified time.

Source: `formalization/MovingFrames.lean`.

Exact lines 23 to 29. Excerpt `frame_product`. Hash in Section 12.

```
theorem matrix_vector_derivative (A : ℝ → Matrix n n ℂ) (y : ℝ → n → ℂ)
    (dA : Matrix n n ℂ) (dy : n → ℂ) (t : ℝ)
    (ha : ∀ i j, HasDerivAt (fun s => A s i j) (dA i j) t)
    (hy : ∀ i, HasDerivAt (fun s => y s i) (dy i) t) (i : n) :
    HasDerivAt (fun s => (A s *ᵥ y s) i) ((dA *ᵥ y t + A t *ᵥ dy) i) t := by
  simpa only [Matrix.mulVec, dotProduct, Pi.add_apply, Pi.mul_apply, Finset.sum_add_distrib] using
    HasDerivAt.fun_sum (u := Finset.univ) (fun j _ => (ha i j).mul (hy j))
```

Proof reading. The proof applies the scalar product rule to each summand of the product of a matrix and a vector, then differentiates the finite sum. It produces the two expected terms without assuming differentiability of the transformed coordinates in advance. The lemma for differentiating an adjoint conjugates the derivative of the transposed entry. It uses the fact that conjugation is linear over the real numbers.

Differentiating $U^*U = 1$ gives $\dot{U}^*U + U^*\dot{U} = 0$, so $U^*\dot{U}$ is skew-adjoint and $-iU^*\dot{U}$ is Hermitian. The conjugated term U^*HU is Hermitian when H is Hermitian. That last premise is explicitly present in the final source. Unitarity of U by itself does not establish Hermiticity of an arbitrary H.

Source: `formalization/MovingFrames.lean`.

Exact lines 110 to 132. Excerpt `moving_coordinates`. Hash in Section 12.

```
theorem moving_frame_coordinates [DecidableEq n]
    (U : ℝ → Matrix n n ℂ) (psi : ℝ → n → ℂ)
    (H dU : Matrix n n ℂ) (dpsi : n → ℂ) (t : ℝ)
    (hu : ∀ i j, HasDerivAt (fun s => U s i j) (dU i j) t)
    (hp : ∀ i, HasDerivAt (fun s => psi s i) (dpsi i) t)
    (hleft : ∀ s, (U s).conjTranspose * U s = 1)
    (hright : ∀ s, U s * (U s).conjTranspose = 1)
    (hsch : Complex.I • dpsi = H *ᵥ psi t)
    (hH : H.conjTranspose = H) :
    let y := fun s => (U s).conjTranspose *ᵥ psi s
    let dy := dU.conjTranspose *ᵥ psi t + (U t).conjTranspose *ᵥ dpsi
    (∀ i, HasDerivAt (fun s => y s i) (dy i) t) ∧
      Complex.I • dy = effectiveHamiltonian (U t) H dU *ᵥ y t ∧
      (effectiveHamiltonian (U t) H dU).conjTranspose =
```

```
      effectiveHamiltonian (U t) H dU := by
  dsimp only
  have hy := matrix_vector_derivative (fun s => (U s).conjTranspose) psi
    dU.conjTranspose dpsi t (adjoint_derivative U dU t hu) hp
  refine ⟨hy, ?_, effective_hamiltonian_hermitian U H dU t hu hleft hH⟩
  apply moving_frame_equation U psi (fun s => (U s).conjTranspose *ᵥ psi s)
    H dU dpsi _ t hu hp hy (hleft t) _ hsch
  intro s
  rw [Matrix.mulVec_mulVec, hright, Matrix.one_mulVec]
```

Contract. The matrix index is finite with decidable equality. Entrywise derivative hypotheses are at one real time t. Both unitary identities hold for every time, permitting construction of the inverse coordinates. The original Schrödinger equation $i\dot{\psi} = H\psi(t)$ is a premise, and $H^* = H$ is the premise for the Hermiticity conclusion. The theorem constructs y, constructs its derivative and proves its equation. Existence of a solution for arbitrary initial data is a separate question.

Proof reading. After unfolding the two `let` expressions, the proof obtains the actual derivative of $U^*\psi$ from the product rule. It gives that derivative proof to `moving_frame_equation`. Unitarity on the right reconstructs $\psi = Uy$, while unitarity on the left removes U^*U when the differentiated equation is multiplied by U^*. The separate Hermiticity lemma uses the differentiated unitary identity and the supplied Hermiticity of H.

5.2 The canonical connection derives its identities for first derivatives

For a differentiable complete orthogonal family $P_a(t)$, write $D_a = \dot{P}_a(t)$. Differentiating multiplication gives

$$D_a P_b + P_a D_b = \delta_{ab} D_a.$$

Differentiating completeness gives $\sum_a D_a = 0$. Differentiating self-adjointness gives $D_a^* = D_a$. These are consequences of actual derivatives of the original functions.

Source: `formalization/CanonicalConnection.lean`.

Exact lines 78 to 95. Excerpt `connection_jet`. Hash in Section 12.

```
theorem differentiated_orthogonality
    (hd : ∀ a, HasDerivAt (P a) (D a) t)
    (hmul : ∀ s a b, P a s * P b s = if a = b then P a s else 0)
    (a b : I) : D a * P b t + P a t * D b = if a = b then D a else 0 := by
  have h := (hd a).mul (hd b)
  change HasDerivAt (fun s => P a s * P b s) (D a * P b t + P a t * D b) t at h
  by_cases hab : a = b
  · subst b
    have heq : (fun s => P a s * P a s) = P a := by
      funext s
      simpa only [ite_true] using hmul s a a
    rw [heq] at h
    simpa only [ite_true] using h.unique (hd a)
  · have heq : (fun s => P a s * P b s) = fun _ => (0 : R) := by
      funext s
      simpa only [hab, ite_false] using hmul s a b
    rw [heq] at h
    simpa only [hab, ite_false] using h.unique (hasDerivAt_const t (0 : R))
```

The general algebraic part of the file can take these jet identities as hypotheses. The endpoint below proves those hypotheses from the derivative data before applying that algebraic part. It constructs

$$C = \sum_a D_a P_a = \frac{1}{2} \sum_a [D_a, P_a]$$

and establishes $C^* = -C$ and $[C, P_b] = D_b$.

Source: `formalization/CanonicalConnection.lean`.

Exact lines 113 to 131. Excerpt `canonical_connection`. Hash in Section 12.

```
theorem canonical_connection_of_hasDerivAt
    (hd : ∀ a, HasDerivAt (P a) (D a) t)
    (hmul : ∀ s a b, P a s * P b s = if a = b then P a s else 0)
    (hsum : ∀ s, ∑ a, P a s = 1)
    (hstar : ∀ s a, star (P a s) = P a s) :
    let C := canonical (fun a => P a t) D
    star C = -C ∧
      (∀ b, C * P b t - P b t * C = D b) ∧
      (1 / 2 : ℝ) • (∑ a, (D a * P a t - P a t * D a)) = C := by
  dsimp only
  have hj := differentiated_orthogonality P D t hd hmul
  have hdiag (a : I) : D a * P a t + P a t * D a = D a := by
    simpa only [ite_true] using hj a a
  have hz := differentiated_completeness P D t hd hsum
  refine ⟨?_, ?_, ?_⟩
  · exact canonical_skewAdjoint _ _ (hstar t)
      (differentiated_selfAdjoint P D t hd hstar) hz hdiag
  · exact canonical_commutator _ _ (hmul t) (hsum t) hj
  · exact canonical_half_commutator _ _ hz hdiag
```

Contract. The theorem for actual derivatives works in a normed real algebra with a star ring structure, compatible real scalar action and continuous involution. The finite packet index has decidable equality. The projector multiplication, completeness and adjoint identities hold for all real times. This theorem treats a curve of projectors. Manifold curvature is developed in the later modules.

The skew-adjointness calculation sums $D_a P_a + P_a D_a = D_a$. Their sum vanishes, giving $C + C^* = 0$. The commutator calculation first isolates $CP_b = D_b P_b$, then obtains $D_b + P_b C = D_b P_b$ from the differentiated multiplication identities and completeness. Subtracting gives the desired commutator. The factor of one half uses the real scalar structure and the same zero sum.

5.3 The crucial off-diagonal derivative block is proved

For an idempotent P that varies with a parameter, differentiating $P^2 = P$ in direction v gives

$$D_v P \, P + P \, D_v P = D_v P.$$

Multiplying this equation by P and cancelling an identical summand gives $P(D_v P)P = 0$. The curvature calculation therefore uses a consequence of idempotence and differentiation.

Source: `formalization/ProjectedCurvature.lean`.

Exact lines 22 to 26. Excerpt `sandwich_zero`. Hash in Section 12.

```
theorem jet_sandwich_zero (P D : R) (hP : P*P=P)
    (hD : D*P+P*D=D) : P*D*P=0 := by
  have hh := congrArg (fun z => P*z) hD
  simp only [mul_add, ← mul_assoc, hP] at hh
  exact add_eq_right.mp hh
```

For a range section s with $Ps = s$, differentiating the range condition gives

$$D_v s = (D_v P)s + P(D_v s).$$

The chart connection is $\nabla_X s = P(D_X s)$. Its curvature is the actual commutator

$$R(X,Y)s = \nabla_X \nabla_Y s - \nabla_Y \nabla_X s - \nabla_{[X,Y]} s.$$

The bracket term is essential: it cancels the change in the vector fields in the second directional derivatives. The proof uses Mathlib's actual Fréchet derivative and actual Lie bracket, rather than independently supplied coefficients of second order.

Source: `formalization/LocalProjectedCurvature.lean`.

Exact lines 79 to 94. Excerpt `local_curvature`. Hash in Section 12.

```
theorem projected_connection_curvature (P : E → F →L[ℝ] F)
    (X Y : E → E) (s : E → F) (x : E)
    (hP : DifferentiableAt ℝ P x) (hPP : ∀ x, P x * P x = P x)
    (hX : DifferentiableAt ℝ X x) (hY : DifferentiableAt ℝ Y x)
    (hs : ContDiffAt ℝ 2 s x) (hPs : ∀ x, P x (s x)=s x) :
    curvature P X Y s x = curvatureForm P x (X x) (Y x) (s x) := by
  have hs1 : DifferentiableAt ℝ s x := hs.differentiableAt (by norm_num)
  have hz (v : E) (z : F) : P x (fderiv ℝ P x v (P x z)) = 0 := by
    have hh := congrArg (fun L : F →L[ℝ] F => L z)
      (projector_differential_sandwich P x hP hPP v)
    simpa [mul_apply_eq_comp] using hh
  rw [curvature_expansion P X Y s x hP hPP hX hY hs]
  conv_lhs => rw [section_jet P s x hP hs1 hPs (Y x),
    section_jet P s x hP hs1 hPs (X x)]
  simp only [map_add, hz, add_zero]
  simp [curvatureForm, hPs, map_sub]
```

Contract. The local theorem has real normed model and fiber spaces. P, X and Y are differentiable at the specified point. The section s is twice continuously differentiable there. The idempotent equation and the equation expressing that the section lies in the range hold as functions. An earlier theorem in global chart coordinates assumes global differentiability. The local variant supports use inside arbitrary chart neighborhoods.

Proof reading. Expand the derivative of the projected directional derivative by the product rule. The derivative identity for the Lie bracket cancels the second derivatives of s and the derivatives of X, Y. Substitute the differentiated identity for a section in the range into the two remaining terms involving first derivatives. The components $P(D_v P)P(D_w s)$ vanish by the sandwich lemma. The surviving term is $P[(D_X P), (D_Y P)]s$, and $Ps = s$ puts the final right-hand P into the curvature form.

5.4 Build the range atlas from the projector

The global development begins with an operator family $P : M \to \mathcal{L}_{\mathbb{R}}(F, F)$. Its fiber is the fixed subspace $\{v : P(x)v = v\}$ with the subspace topology. The total space is the subtype of $M \times F$ consisting of such fixed pairs. The atlas is constructed from P rather than supplied as a hypothesis.

For idempotents p, q, define

$$T(p, q) = pq + (1 - p)(1 - q).$$

This is identity at $p = q$, and it intertwines the two projectors. Near a base point a, the map $T(P(x), P(a))$ is invertible because invertibility is open in the complete operator algebra and it is identity at $x = a$.

Source: `formalization/GlobalRangeBundle.lean`.

Exact lines 43 to 43. Excerpt `range_transport`. Hash in Section 12.

```
def transport (p q : F →L[ℝ] F) : F →L[ℝ] F := p*q+(1-p)*(1-q)
```

Source: `formalization/GlobalRangeBundle.lean`.

Exact lines 49 to 54. Excerpt `range_intertwines`. Hash in Section 12.

```
theorem transport_intertwines (p q : F →L[ℝ] F) (hp : p*p=p) (hq : q*q=q) :
    p*transport p q=transport p q*q := by
  unfold transport
  have ha : p*(p*q)=p*q := by rw [← mul_assoc,hp]
  have hb : p*(q*q)=p*q := by rw [hq]
  noncomm_ring [hp,hq,ha,hb]
```

Let $U_a = \{x : T(P(x), P(a)) \text{ is invertible}\}$. Reconstruction transports the fixed fiber at a into the ambient space at x. Coordinates apply the inverse transport and project to the fixed fiber at a. The intertwiners show that both maps land in the correct fibers. On U_a, the two inverse identities follow from genuine inverse cancellation.

Source: `formalization/GlobalRangeBundle.lean`.

Exact lines 180 to 209. Excerpt `range_product`. Hash in Section 12.

```
def localProduct (P : M → F →L[ℝ] F) (hPP : ∀ x, P x*P x=P x)
    (hP : ContMDiff I 𝓘(ℝ, F →L[ℝ] F) ∞ P) (a : M) :
    LocalTotal P a ≃ₜ (domain P a × fiber (P a)) where
  toFun z := ((z.val.val.1,z.property),toCoord P hPP a z.val.val.1 z.val.val.2)
  invFun y := ⟨⟨(y.1.val,fromCoord P a y.1.val y.2),
    fromCoord_range P hPP a y.1.val y.2⟩,y.1.property⟩
  left_inv := by
    intro z
    apply Subtype.ext
    apply Subtype.ext
    apply Prod.ext
    · rfl
    · exact fromCoord_toCoord P hPP a _ z.property _ z.val.property
  right_inv := by
```

```
    intro y
    apply Prod.ext
    · rfl
    · exact toCoord_fromCoord P hPP a _ y.1.property y.2
  continuous_toFun := by
    have hx : Continuous (fun z : LocalTotal P a => z.val.val.1) := by fun_prop
    have hv : Continuous (fun z : LocalTotal P a => z.val.val.2) := by fun_prop
    have hc : Continuous (fun z : LocalTotal P a => toCoord P hPP a z.val.val.1) :=
      (smooth_toCoord P hPP hP a).continuousOn.comp_continuous hx (fun z => z.property)
    exact (hx.subtype_mk _).prodMk (hc.clm_apply hv)
  continuous_invFun := by
    have hx : Continuous (fun y : domain P a × fiber (P a) => y.1.val) := by fun_prop
    have hv : Continuous (fun y : domain P a × fiber (P a) => y.2) := by fun_prop
    have hc : Continuous (fun y : domain P a × fiber (P a) => fromCoord P a y.1.val) :=
      (smooth_fromCoord P hP a).continuous.comp hx
    exact ((hx.prodMk (hc.clm_apply hv)).subtype_mk _).subtype_mk _
```

This declaration is a homeomorphism from the restricted actual total space to $U_a \times \mathrm{fiber}(P(a))$. In addition to inverse laws, it proves continuity in both directions using the smooth coordinate and reconstruction maps. Transitions are composed from these actual maps. They are continuous linear maps in each fiber, smooth in the base, and satisfy the cocycle law by cancelling the reconstruction and coordinate maps in the middle.

Source: `formalization/GlobalRangeBundle.lean`.

Exact lines 241 to 252. Excerpt `range_atlas`. Hash in Section 12.

```
theorem smooth_range_bundle_atlas (P : M → F →L[ℝ] F) (hPP : ∀ x, P x*P x=P x)
    (hP : ContMDiff I 𝓘(ℝ, F →L[ℝ] F) ∞ P) :
    (∀ a, IsOpen (domain P a) ∧ a ∈ domain P a) ∧
    (∀ a, Nonempty (LocalTotal P a ≃ₜ (domain P a × fiber (P a)))) ∧
    (∀ a b, ContMDiffOn I 𝓘(ℝ, fiber (P a) →L[ℝ] fiber (P b)) ∞
      (transition P hPP a b) (domain P a ∩ domain P b)) ∧
    (∀ a x, x ∈ domain P a → transition P hPP a a x=1) ∧
    (∀ a b c x, x ∈ domain P a ∩ domain P b ∩ domain P c →
      (transition P hPP b c x).comp (transition P hPP a b x)=transition P hPP a c x) := by
  exact ⟨fun a => ⟨isOpen_domain P hP.continuous a,mem_domain P hPP a⟩,
    fun a => ⟨localProduct P hPP hP a),smooth_transition P hPP hP,
    transition_self P hPP,fun a b c x hx => transition_cocycle P hPP a b c x hx.1.2⟩
```

Contract. The construction of the range atlas uses a smooth real projector map on a charted parameter space and a complete real normed fiber space. This bundle construction requires idempotence and does not require self-adjointness. The explicit atlas theorem does not demand a global chart. Local linear fiber equivalences prove local constancy of finite rank whenever finite rank is being interpreted. The underlying fiber equivalence is the stronger structural statement.

The implementation uses this explicit smooth atlas directly. It does not package it as a Mathlib `ContMDiffVectorBundle` instance. The total space, open domains, local product homeomorphisms, transitions and their identities are all constructed and proved. The distinction concerns how these results are packaged in Lean.

5.5 The global derivative and gauge gluing

On the manifold, the global connection is defined using `mvfderiv`, the derivative of a map from a manifold to a vector space. A tangent vector at x has type `TangentSpace I x`, so a vector field is a dependent function giving one such vector at every point.

Source: `formalization/GlobalProjectedConnection.lean`.

Exact lines 29 to 30. Excerpt `global_nabla`. Hash in Section 12.

```
def nabla (P : M → F →L[ℝ] F) (X : ∀ x : M, TangentSpace I x) (s : M → F) (x : M) : F :=
  P x (mvfderiv I s x (X x))
```

Source: `formalization/GlobalProjectedConnection.lean`.

Exact lines 59 to 63. Excerpt `global_leibniz`. Hash in Section 12.

```
theorem nabla_leibniz (P : M → F →L[ℝ] F) (X : ∀ x : M, TangentSpace I x)
    (s : M → F) (c : M → ℝ) (x : M) (hs : MDifferentiableAt I 𝓘(ℝ,F) s x)
    (hc : MDifferentiableAt I 𝓘(ℝ,ℝ) c x) (hPs : P x (s x)=s x) :
    nabla I P X (c • s) x = (mvfderiv I c x (X x)) • s x + c x • nabla I P X s x := by
  simp [nabla,mvfderiv_smul hc hs,hPs,add_comm]
```

The global range property follows immediately from idempotence. Additivity in sections uses actual differentiability premises. The Leibniz rule needs the range condition for $s(x)$ because the scalar derivative term initially contains $P(x)s(x)$. Linearity in the vector field uses linearity of the derivative. A germ lemma says that locally equal functions have the same covariant derivative at the point.

Local coordinate connections reconstruct a range section, differentiate it globally and then take coordinates. On an open domain, local reconstruction equals the original section in a neighborhood. The germ lemma therefore identifies the local and global connection. On an overlap, coordinate transitions carry one representation into the other.

Source: `formalization/GlobalProjectedConnection.lean`.

Exact lines 115 to 124. Excerpt `global_glue`. Hash in Section 12.

```
theorem local_connections_glue (P : M → F →L[ℝ] F) (hPP : ∀ x, P x*P x=P x)
    (hP : ContMDiff I 𝓘(ℝ,F →L[ℝ] F) ∞ P) (a b : M)
    (X : ∀ x : M, TangentSpace I x) (s : M → F) (hPs : ∀ x, P x (s x)=s x)
    (x : M) (hx : x ∈ domain P a ∩ domain P b) :
    transition P hPP a b x (localNabla I P hPP a X (fun y => toCoord P hPP a y (s y)) x) =
      localNabla I P hPP b X (fun y => toCoord P hPP b y (s y)) x := by
  rw [localNabla_represents_global I P hPP hP a X s hPs x hx.1,
    localNabla_represents_global I P hPP hP b X s hPs x hx.2]
  change toCoord P hPP b x (fromCoord P a x (toCoord P hPP a x (nabla I P X s x))) = _
  rw [fromCoord_toCoord P hPP a x hx.1 _ (nabla_in_range I P hPP X s x)]
```

Source: `formalization/ProjectedBundleCompletion.lean`.

Exact lines 40 to 53. Excerpt `smooth_section`. Hash in Section 12.

```
theorem smooth_section_iff (P : M → F →L[ℝ] F) (hPP : ∀ x, P x*P x=P x)
    (hP : ContMDiff I 𝓘(ℝ,F →L[ℝ] F) ∞ P) (a x : M) (hx : x ∈ domain P a)
    (s : M → F) (hPs : ∀ᶠ y in 𝓝 x, P y (s y)=s y) :
    ContMDiffAt I 𝓘(ℝ,F) ∞ s x ↔
      ContMDiffAt I 𝓘(ℝ,fiber (P a)) ∞ (fun y => toCoord P hPP a y (s y)) x := by
  constructor
  · intro hs
    exact ((smooth_toCoord P hPP hP a x hx).contMDiffAt
      ((isOpen_domain P hP.continuous a).mem_nhds hx)).clm_apply hs
  · intro hs
    have hr := (smooth_fromCoord P hP a x).clm_apply hs
    apply hr.congr_of_eventuallyEq
    filter_upwards [hPs,(isOpen_domain P hP.continuous a).mem_nhds hx] with y hy hya
    exact (fromCoord_toCoord P hPP a y hya (s y) hy).symm
```

Source: `formalization/ProjectedBundleCompletion.lean`.

Exact lines 57 to 70. Excerpt `gauge_law`. Hash in Section 12.

```
theorem local_connections_gauge (P : M → F →L[ℝ] F) (hPP : ∀ x, P x*P x=P x)
    (hP : ContMDiff I 𝓘(ℝ,F →L[ℝ] F) ∞ P) (a b x : M)
    (hx : x ∈ domain P a ∩ domain P b) (X : ∀ y : M, TangentSpace I y)
    (u : M → fiber (P a)) :
    localNabla I P hPP b X (fun y => transition P hPP a b y (u y)) x =
      transition P hPP a b x (localNabla I P hPP a X u x) := by
  have hs : ∀ y, P y (fromCoord P a y (u y))=fromCoord P a y (u y) :=
    fun y => fromCoord_range P hPP a y (u y)
  change localNabla I P hPP b X (fun y => toCoord P hPP b y (fromCoord P a y (u y))) x = _
  rw [localNabla_represents_global I P hPP hP b X (fun y => fromCoord P a y (u y)) hs x hx.2]
  change toCoord P hPP b x (nabla I P X (fun y => fromCoord P a y (u y)) x) =
    toCoord P hPP b x (fromCoord P a x
      (toCoord P hPP a x (nabla I P X (fun y => fromCoord P a y (u y)) x)))
  rw [fromCoord_toCoord P hPP a x hx.1 _ (nabla_in_range I P hPP X _ x)]
```

The equivalence for smooth sections needs only an eventual range condition near the point, not an unproved globally extended section. The gauge law in the completion file applies to arbitrary local coordinate functions and compares their reconstructed connection values on an overlap. It is not restricted to a prechosen global frame.

5.6 Manifold curvature and its endpoint

The final curvature theorem uses a smooth real boundaryless manifold modeled on a complete real normed space E. The model is $\mathcal{J}(\mathbb{R}, E)$. The manifold has `ChartedSpace E M` and a smooth `IsManifold` instance for that model. The source pulls vector fields into a chart centered at the point, identifies the manifold derivative and bracket with their chart versions, and then proves the cancellation using actual manifold operations.

Source: `formalization/ManifoldProjectedCurvature.lean`.

Exact lines 214 to 232. Excerpt `manifold_curvature`. Hash in Section 12.

```
theorem projected_connection_curvature (P : M → F →L[ℝ] F)
    (X Y : ∀ x : M, TangentSpace 𝓘(ℝ,E) x) (s : M → F) (x : M)
    (hP : ContMDiff 𝓘(ℝ,E) 𝓘(ℝ,F →L[ℝ] F) ∞ P) (hPP : ∀ x, P x*P x=P x)
    (hX : ContMDiff 𝓘(ℝ,E) 𝓘(ℝ,E).tangent ∞
      (fun x => (X x : TangentBundle 𝓘(ℝ,E) M)))
    (hY : ContMDiff 𝓘(ℝ,E) 𝓘(ℝ,E).tangent ∞
      (fun x => (Y x : TangentBundle 𝓘(ℝ,E) M)))
    (hs : ContMDiff 𝓘(ℝ,E) 𝓘(ℝ,F) ∞ s) (hPs : ∀ x, P x (s x)=s x) :
    curvature 𝓘(ℝ,E) P X Y s x = curvatureForm 𝓘(ℝ,E) P x (X x) (Y x) (s x) := by
  have hP1 := hP.mdifferentiableAt (x := x) (by simp)
  have hs1 := hs.mdifferentiableAt (x := x) (by simp)
  have hz (v : TangentSpace 𝓘(ℝ,E) x) (z : F) : P x (mvfderiv 𝓘(ℝ,E) P x v (P x z))=0 := by
    have hh := congrArg (fun L : F →L[ℝ] F => L z)
      (projector_differential_sandwich P x hP1 hPP v)
    simpa [mul_apply_eq_comp] using hh
  rw [curvature_expansion P X Y s x hP hPP hX hY hs]
  conv_lhs => rw [section_jet P s x hP1 hs1 hPs (Y x),section_jet P s x hP1 hs1 hPs (X x)]
  simp only [map_add,hz,add_zero]
  simp [curvatureForm,hPs,map_sub]
```

Contract. Here P, the two vector fields and the ambient range section are smooth. The equations $P(x)^2 = P(x)$ and $P(x)s(x) = s(x)$ are given globally. The conclusion is $R(X,Y)s = P(dP \wedge dP)P\,s$ at an arbitrary point. The final manifold curvature endpoint assumes no global chart, no supplied curvature identity and no supplied range atlas. The atlas portion separately requires completeness of the fiber space. Finite complex quantum spaces fit by restriction to their underlying real spaces.

The next theorem proves tensoriality: replacing the two fields and the section by smooth extensions with the same values at the point leaves curvature unchanged. The completion file makes the curvature form alternating and linear over the real numbers, proves it maps into the actual fiber, and defines the resulting continuous linear fiber endomorphism. Constant projector maps, including zero and identity, have zero curvature form.

The six files on projected geometry follow this mathematical development. `ProjectedCurvature` establishes the algebra and calculus on the parameter space. `LocalProjectedCurvature` makes the hypotheses local, and `GlobalRangeBundle` constructs the atlas. `GlobalProjectedConnection` defines the connection and proves that its local descriptions agree. `ManifoldProjectedCurvature` proves the global curvature formula. Finally, `ProjectedBundleCompletion` proves compatibility with intrinsic sections, gauge transformations and the curvature form on each fiber. The later global constructions extend the results of the earliest chart file.

6 Finite channels and the full Knill–Laflamme criterion

6.1 Complete positivity has an actual amplification map

For finite complex matrices, the Kraus channel is the map

$$\Phi(\rho) = \sum_a K_a \rho K_a^*.$$

Its additivity and complex homogeneity construct an actual linear map. Each summand preserves positive semidefiniteness by a matrix congruence, so the finite sum is positive. Trace preservation is a separate theorem under $\sum_a K_a^* K_a = 1$, proved by cyclic trace rotation and completeness of that Kraus sum.

To prove complete positivity the source does not merely restate positivity. It defines blockwise amplification on matrices indexed by $J \times n$ and identifies it entrywise with the actual Kraus family $1_J \otimes K_a$.

Source: `formalization/KrausChannels.lean`.

Exact lines 48 to 56. Excerpt `kraus_amplify`. Hash in Section 12.

```
theorem amplification_eq_kraus {J : Type*} [Fintype J] [DecidableEq J]
    (K : A → Matrix n n ℂ) (rho : Matrix (J × n) (J × n) ℂ) :
    amplify (channel K) rho =
      channel (fun a => (1 : Matrix J J ℂ) ⊗ₖ K a) rho := by
  ext ⟨i, u⟩ ⟨j, v⟩
  simp_rw [amplify, channel, Matrix.sum_apply, Matrix.mul_apply,
    Matrix.conjTranspose_apply, Matrix.kroneckerMap_apply]
  simp [Fintype.sum_prod_type, Matrix.one_apply, apply_ite]
  rfl
```

Source: `formalization/KrausChannels.lean`.

Exact lines 58 to 62. Excerpt `kraus_cp`. Hash in Section 12.

```
theorem channel_completely_positive {J : Type*} [Fintype J] [DecidableEq J]
    (K : A → Matrix n n ℂ) {rho : Matrix (J × n) (J × n) ℂ}
    (hrho : rho.PosSemidef) : (amplify (channel K) rho).PosSemidef := by
  rw [amplification_eq_kraus]
  exact channel_positive _ hrho
```

Contract. The system and Kraus index types are finite. The reference type `J` is arbitrary finite with decidable equality. Complete positivity itself does not require Kraus normalization. Normalization is needed for trace preservation and preservation of density matrices. Every finite reference dimension is quantified in the theorem, rather than sampled in an example. Infinite reference systems are not encoded by these finite matrix declarations.

Proof reading. The amplification identity expands entries of a Kronecker product, uses the identity matrix to force equality of the reference indices, and collapses the corresponding sums. Once amplification is literally a Kraus map, the earlier positivity theorem applies without a new positivity hypothesis about the amplified channel.

6.2 Orthogonal syndromes are the recovery stage

The recovery construction begins with a code projection P and an orthogonal normalized family E_a satisfying

$$E_a P = E_a, \qquad E_a^* E_b = \delta_{ab} P.$$

Their output packets $Q_a = E_a E_a^*$ are orthogonal star projections. Their sum Q is a star projection and $Q^\perp = 1 - Q$ is its complement.

Source: `formalization/SyndromeRecovery.lean`.

Exact lines 21 to 27. Excerpt `syndrome_structure`. Hash in Section 12.

```
structure OrthogonalSyndromes (R I : Type*) [Ring R] [StarRing R]
    [Fintype I] [DecidableEq I] where
  code : R
  code_projection : IsStarProjection code
  error : I → R
  restricted : ∀ a, error a * code = error a
  gram : ∀ a b, star (error a) * error b = if a=b then code else 0
```

The decoder is

$$\mathcal{R}(\sigma) = \sum_a E_a^* \sigma E_a + Q^\perp \sigma Q^\perp.$$

The complement term is important. Without it the decoder would generally lose trace on input outside the sum of syndrome output spaces. The implementation uses `Option I` to index the complete Kraus family for recovery: `some a` denotes E_a^*, and `none` denotes $Q^\perp$. Thus the extra branch is encoded in the actual finite family.

Source: `formalization/SyndromeRecovery.lean`.

Exact lines 84 to 90. Excerpt `syndrome_complete`. Hash in Section 12.

```
theorem recoveryKraus_complete :
    (∑ a, star (S.recoveryKraus a)*S.recoveryKraus a) = 1 := by
  rw [Fintype.sum_option]
  simp only [recoveryKraus, star_star, S.complement_projection.isSelfAdjoint.star_eq,
    S.complement_projection.isIdempotentElem.eq]
  change S.complement + S.total = 1
  simp [complement]
```

Proof reading. The sum over `Option I` splits into the complement branch and the syndrome branches. Self-adjointness and idempotence turn the complement contribution into $Q^\perp$. The syndrome contribution is Q. Their sum is the identity. This proves trace preservation on the entire ambient matrix space, not merely on noisy code states.

For an arbitrary operator supported on the code, $P\rho P = \rho$, the decoder of a cross error is

$$\mathcal{R}(E_a \rho E_b^*) = \delta_{ab} \rho.$$

The orthogonal Gram law kills every nonmatching syndrome contribution, and the complement annihilates each error output.

Source: `formalization/SyndromeRecovery.lean`.

Exact lines 103 to 124. Excerpt `syndrome_cross`. Hash in Section 12.

```
theorem decoder_cross_error (rho : R) (hrho : S.code*rho*S.code=rho) (a b : I) :
    S.decoder (S.error a*rho*star (S.error b)) = if a=b then rho else 0 := by
  have hc (c : I) :
      star (S.error c)*(S.error a*rho*star (S.error b))*S.error c =
        if c=a then (if a=b then rho else 0) else 0 := by
    have heq : star (S.error c)*(S.error a*rho*star (S.error b))*S.error c =
        (star (S.error c)*S.error a)*rho*(star (S.error b)*S.error c) := by
      noncomm_ring
    rw [heq, S.gram, S.gram]
    by_cases hca : c=a
    · subst c
      by_cases hab : a=b
      · subst b; simp [hrho]
      · simp [hab, Ne.symm hab]
    · simp [hca]
  have hq : S.complement*(S.error a*rho*star (S.error b))*S.complement=0 := by
    calc
      _ = (S.complement*S.error a)*rho*(star (S.error b)*S.complement) := by
        noncomm_ring
      _ = 0 := by rw [S.complement_error, zero_mul, zero_mul]
  simp only [decoder, hc, hq, add_zero]
  simp
```

Over an abstract star ring, this gives an algebraic correction identity. The construction for complex matrices additionally proves positivity, trace preservation, complete positivity at every finite reference dimension and preservation of density matrices. A weighted normalized mixture and a coherent error superposition are then handled by linearity and the formula for cross errors.

This file alone would prove recovery for supplied orthogonal syndrome data. It would not prove that arbitrary Knill–Laflamme Gram data can be put into that form, and it would not prove necessity of the Gram condition. The current source has separate modules for those obligations.

6.3 General Gram data are not assumed diagonal

The finite Knill–Laflamme data consist of a star projection P, errors E_a, and a positive semidefinite coefficient matrix C with

$$PE_a^* E_b P = C_{ab} P.$$

No invertibility of C is required. The matrix can be singular and can have off-diagonal coefficients.

Source: `formalization/KnillLaflammeSufficiency.lean`.

Exact lines 187 to 193. Excerpt `kl_data`. Hash in Section 12.

```
structure Data (n I : Type*) [Fintype n] [DecidableEq n] [Fintype I] [DecidableEq I] where
  code : Matrix n n ℂ
  code_projection : IsStarProjection code
  error : I → Matrix n n ℂ
  gramMatrix : Matrix I I ℂ
```

```
  gram_positive : gramMatrix.PosSemidef
  gram : ∀ a b, code * star (error a) * error b * code = gramMatrix a b • code
```

First restrict each error to the code by replacing E_a with $E_a P$. This changes neither the noisy output of a supported operator nor its compressed Gram data. The source's `mix E U r` uses the columns of U as the new error coordinates. It proves that mixing transforms the Gram matrix to $U^* C U$ and that unitary mixing leaves the Kraus noise map unchanged.

The actual Hermitian spectral theorem supplies `eigenvectorUnitary` and real eigenvalues. Positive semidefiniteness proves those eigenvalues are nonnegative. The declaration below constructs the diagonal error data directly from that spectral information.

Source: `formalization/KnillLaflammeSufficiency.lean`.

Exact lines 204 to 214. Excerpt `kl_diagonal`. Hash in Section 12.

```
noncomputable def diagonal : DiagonalErrors n I where
  code := K.code
  code_projection := K.code_projection
  error := mix (fun a => K.error a*K.code)
    (K.gram_positive.isHermitian.eigenvectorUnitary : Matrix I I ℂ)
  restricted := mix_restricted K.code (fun a => K.error a*K.code)
    (fun a => by rw [mul_assoc, K.code_projection.isIdempotentElem.eq]) _
  weight := K.gram_positive.isHermitian.eigenvalues
  nonnegative := K.gram_positive.eigenvalues_nonneg
  gram := diagonalized_gram K.code (fun a => K.error a*K.code) K.gramMatrix
    K.gram_positive K.restricted_error_gram
```

Proof reading. The `error` field mixes the errors restricted to the code by the spectral unitary. The `weight` field uses the actual eigenvalues. The `gram` field applies the proved diagonalization identity. The `restricted` field uses idempotence of the code projector, and the `nonnegative` field comes from positivity of the input Gram matrix. These fields construct the required diagonal data.

6.4 Singular directions are eliminated without division by zero

If a diagonal weight is zero, the corresponding restricted error satisfies $E^* E = 0$. For finite complex matrices this forces $E = 0$. Therefore the construction does not normalize a zero spectral direction.

Source: `formalization/KnillLaflammeSufficiency.lean`.

Exact lines 107 to 110. Excerpt `kl_null`. Hash in Section 12.

```
theorem zero_weight_error (a : I) (ha : D.weight a = 0) : D.error a = 0 := by
  have hh : star (D.error a) * D.error a = 0 := by
    simpa only [ite_true, ha, Complex.ofReal_zero, zero_smul] using D.gram a a
  exact Matrix.conjTranspose_mul_self_eq_zero.mp hh
```

The active index is the subtype `{a : I // 0 < D.weight a}`. For an active error define $F_a = \lambda_a^{-1/2} E_a$. The source proves the square root is nonzero and its square equals the weight before using its inverse. The normalized errors satisfy $F_a^* F_b = \delta_{ab} P$, so the preceding syndrome construction applies.

For an unnormalized original error family, the decoder corrects the noise to the exact scale

$$\mathcal{R}\left(\sum_a E_a \rho E_a^*\right) = \mathrm{tr}(C)\rho.$$

Unitary mixing preserves the noise, the restricted noise agrees on the code, and the spectral eigenvalue sum is the trace of C. If the original errors form a normalized channel and $P \neq 0$, compressing the Kraus completeness relation gives $\mathrm{tr}(C)P = P$, so the scale is one. If $P = 0$, every supported operator is zero and the correction assertion follows directly.

Source: `formalization/KnillLaflammeSufficiency.lean`.

Exact lines 252 to 259. Excerpt `kl_corrects`. Hash in Section 12.

```
theorem recovery_corrects (hcomplete : (∑ a, star (K.error a)*K.error a) = 1)
    (rho : Matrix n n ℂ) (hrho : K.code*rho*K.code=rho) :
    K.diagonal.syndromes.decoder (noise K.error rho) = rho := by
  rw [K.recovery_noise_scaled rho hrho]
  by_cases hp : K.code = 0
  · have hr : rho=0 := by simpa [hp] using hrho.symm
    simp [hr]
  · rw [K.gram_trace_one hp hcomplete, one_smul]
```

Contract. This sufficiency statement assumes finite complex matrices, a star code projection, a positive Gram matrix and its compressed Gram relation, together with global Kraus normalization for exact channel correction with scale one. The correction conclusion covers all operators supported on the code, including those that are not positive. The decoder is an actual finite CPTP extension on the whole space. Its spectral construction is mathematical and noncomputable. The theorem supplies neither a numerical eigenvalue routine nor a performance guarantee for an executable decoder.

6.5 Necessity starts from actual recovery

The reverse implication takes a finite recovery family R_r with $\sum_r R_r^* R_r = 1$, and assumes its actual composite Kraus map fixes every operator supported on the code. It must derive the Gram condition rather than take it as a premise.

The key rigidity lemma says that if A^*A is the rank-one Gram matrix of a vector p, then every row of A is a scalar multiple of p. If $p = 0$, positivity gives $A = 0$. Otherwise choose a nonzero coordinate p_t. A vector supported in two coordinates and orthogonal to p is in the kernel of A^*A, hence of A. The resulting equation in two coordinates yields the row proportionality.

Source: `formalization/KnillLaflammeNecessity.lean`.

Exact lines 19 to 45. Excerpt `kl_rigidity`. Hash in Section 12.

```
theorem rank_one_gram_rigidity {q J : Type*} [Fintype q] [DecidableEq q]
    [Fintype J] [DecidableEq J] (A : Matrix J q ℂ) (p : q → ℂ)
    (hgram : A.conjTranspose*A = Matrix.vecMulVec (star p) p) :
    ∃ z : J → ℂ, ∀ j t, A j t = z j*p t := by
  by_cases hp : p=0
  · have hA : A=0 := Matrix.conjTranspose_mul_self_eq_zero.mp (by simpa [hp] using hgram)
```

```
    exact ⟨fun _ => 0, by simp [hA]⟩
  · obtain ⟨t,ht⟩ : ∃ t, p t ≠ 0 := Function.ne_iff.mp hp
    refine ⟨fun j => A j t / p t, ?_⟩
    intro j s
    let v : q → ℂ := Pi.single s (p t) - Pi.single t (p s)
    have hpv : p ▪ᵥ v = 0 := by
      simp [v, dotProduct_sub, dotProduct_single, mul_comm]
    have hav : A *ᵥ v=0 := by
      apply (Matrix.conjTranspose_mul_self_mulVec_eq_zero A v).mp
      rw [hgram]
      ext u
      change (∑ x, star (p u) * p x * v x) = 0
      simp_rw [mul_assoc]
      rw [← Finset.mul_sum]
      change star (p u) * (p ▪ᵥ v) = 0
      rw [hpv, mul_zero]
    have heq : A j s*p t - A j t*p s=0 := by
      simpa [v, Matrix.mulVec_sub, Matrix.mulVec_single, mul_comm] using congrFun hav j
    change A j s = (A j t / p t) * p s
    field_simp [ht]
    linear_combination heq
```

To apply this to Kraus operators, flatten each operator's entries into a row indexed by a pair of matrix coordinates. Evaluate the asserted map equality on matrix units. This yields the rank-one Gram equality for the flattened matrix. Each composite recovery operator $R_r E_a P$ must therefore be $z_{ra} P$.

Now insert the recovery completeness relation into $P E_a^* E_b P$. It becomes the sum of adjoint products of those composite operators. Their scalar form gives a positive Gram matrix $C = Z^* Z$ with the required coefficients.

Source: `formalization/KnillLaflammeNecessity.lean`.

Exact lines 86 to 105. Excerpt `kl_necessity`. Hash in Section 12.

```
theorem gram_of_recovery (P : Matrix n n ℂ) (hP : IsStarProjection P)
    (E : I → Matrix n n ℂ) (R : J → Matrix n n ℂ)
    (hR : (∑ r, star (R r)*R r)=1)
    (hcorrect : ∀ rho, P*rho*P=rho → noise R (noise E rho)=rho) :
    ∃ C : Matrix I I ℂ, C.PosSemidef ∧
      ∀ a b, P*star (E a)*E b*P=C a b • P := by
  obtain ⟨z,hz⟩ := corrected_kraus_family P hP E R hcorrect
  let Z : Matrix J I ℂ := fun r a => z (r,a)
  refine ⟨Z.conjTranspose*Z, Matrix.posSemidef_conjTranspose_mul_self Z, ?_⟩
  intro a b
  have heq : P*star (E a)*E b*P = ∑ r, star (R r*E a*P)*(R r*E b*P) := by
    calc
      _ = P*star (E a)*(∑ r, star (R r)*R r)*E b*P := by rw [hR, mul_one]
      _ = _ := by
        simp only [Finset.mul_sum, Finset.sum_mul, star_mul,
          hP.isSelfAdjoint.star_eq, mul_assoc]
  rw [heq]
  simp_rw [hz, star_smul, hP.isSelfAdjoint.star_eq, smul_mul_assoc,
    mul_smul_comm, smul_smul, hP.isIdempotentElem.eq]
  simp only [Matrix.mul_apply, Matrix.conjTranspose_apply, Finset.sum_smul, Z]
```

Proof reading. The witness Z^*Z is positive by an existing matrix theorem. The intermediate `heq` is obtained by inserting $\sum_r R_r^* R_r = 1$ without reordering the error factors. After rewriting every $R_r E_a P$ as a scalar times P, self-adjointness and idempotence reduce the sum to the entries of Z^*Z times P. Thus positivity is proved from the recovery, not imposed on an unknown matrix.

6.6 The state formulation is bridged to all code operators

A physical criterion is usually stated for density matrices: positive semidefinite, trace one and supported on the code. The necessity argument above uses all operators supported on the code. The source supplies the missing bridge.

First, a nonzero positive matrix has positive nonzero trace. Scaling by the inverse real trace produces a density matrix. Linearity then removes that normalization. Zero positive matrices are handled separately. Next, the complex polarization identity expresses an arbitrary outer product xy^* as a linear combination of pure positive matrices. Finally, matrix units span all matrices. Two maps that are linear over the complex numbers therefore agree on every operator if they agree on all pure states.

Source: `formalization/QuantumStateBridge.lean`.

Exact lines 119 to 129. Excerpt `density_bridge`. Hash in Section 12.

```
theorem density_corrects_iff_operators {I J : Type*} [Fintype I] [Fintype J]
    (P : Matrix n n ℂ) (hP : IsStarProjection P)
    (E : I → Matrix n n ℂ) (R : J → Matrix n n ℂ) :
    (∀ rho : Matrix n n ℂ, rho.PosSemidef → Matrix.trace rho=1 → P*rho*P=rho →
      noise R (noise E rho)=rho) ↔
    (∀ rho : Matrix n n ℂ, P*rho*P=rho → noise R (noise E rho)=rho) := by
  constructor
  · intro hh rho hs
    exact fixes_operators_of_density P hP ((noiseLinear R).comp (noiseLinear E)) hh rho hs
  · intro hh rho _ _ hs
    exact hh rho hs
```

The predicate `StateCorrectable P E` existentially quantifies a natural number and a recovery family on `Fin m`. Its recovery is normalized and fixes all normalized code density matrices. The final endpoint combines the density bridge with the equivalence already proved on the operator space.

Source: `formalization/QuantumStateBridge.lean`.

Exact lines 139 to 151. Excerpt `kl_criterion`. Hash in Section 12.

```
theorem state_correctable_iff_gram {I : Type*} [Fintype I] [DecidableEq I]
    (P : Matrix n n ℂ) (hP : IsStarProjection P) (E : I → Matrix n n ℂ)
    (hE : (∑ a, star (E a)*E a)=1) :
    StateCorrectable P E ↔ ∃ C : Matrix I I ℂ, C.PosSemidef ∧
      ∀ a b, P*star (E a)*E b*P=C a b • P := by
  rw [← correctable_iff_gram P hP E hE]
  constructor
  · rintro ⟨m,R,hR,hcorrect⟩
    exact ⟨m,R,hR,(density_corrects_iff_operators P hP E R).mp hcorrect⟩
  · rintro ⟨m,R,hR,hcorrect⟩
    exact ⟨m,R,hR,(density_corrects_iff_operators P hP E R).mpr hcorrect⟩

open scoped Kronecker
```

Contract. The code is a star projection. The error family is finite and normalized by $\sum_a E_a^* E_a = 1$. The existential Gram matrix must be positive semidefinite and satisfy every compressed cross relation. The theorem permits the zero code. In that case, no nonzero normalized code density matrix exists, and the operator bridge remains valid because the supported operator space is zero.

6.7 Reference systems and error spans

The theorem for finite reference systems identifies each reference block of $1 \otimes \mathcal{R}$ and $1 \otimes \mathcal{E}$ with the corresponding system noise map. The supported reference matrix condition implies each system block is supported on the code, so the operator correction theorem applies block by block.

Source: `formalization/QuantumStateBridge.lean`.

Exact lines 164 to 182. Excerpt `kl_reference`. Hash in Section 12.

```
theorem recovery_with_reference {I J A : Type*} [Fintype I] [Fintype J] [Fintype A] [DecidableEq
↪  A]
    (P : Matrix n n ℂ) (E : I → Matrix n n ℂ) (R : J → Matrix n n ℂ)
    (hcorrect : ∀ rho, P*rho*P=rho → noise R (noise E rho)=rho)
    (rho : Matrix (A × n) (A × n) ℂ)
    (hs : ((1 : Matrix A A ℂ) ⊗ₖ P)*rho*((1 : Matrix A A ℂ) ⊗ₖ P)=rho) :
    amplifiedNoise R (amplifiedNoise E rho)=rho := by
  have hblock (i j : A) :
      P*(Matrix.of fun a b => rho (i,a) (j,b))*P=(Matrix.of fun a b => rho (i,a) (j,b)) := by
    ext u v
    have hh := congrArg (fun M : Matrix (A × n) (A × n) ℂ => M (i,u) (j,v)) hs
    simpa [Matrix.mul_apply, Matrix.kroneckerMap_apply, Fintype.sum_prod_type, Matrix.one_apply]
    ↪  using hh
  ext ⟨i,u⟩ ⟨j,v⟩
  rw [amplifiedNoise_block]
  have hin : (fun a b => amplifiedNoise E rho (i,a) (j,b)) =
      noise E (fun a b => rho (i,a) (j,b)) := by
    ext a b
    exact amplifiedNoise_block E rho i j a b
  rw [hin]
  exact congrArg (fun M : Matrix n n ℂ => M u v) (hcorrect _ (hblock i j))
```

It follows that the same correction identity holds for entangled states with any finite reference system. The displayed algebraic identity requires neither positivity nor a trace of one. These conditions can be supplied to interpret the matrix as a density state. This is stronger than checking only product states and does not rely on assuming separability.

The module for recovery on the error span keeps track of cross coefficients. With the source's Gram orientation,

$$\mathcal{R}(E_a \rho E_b^*) = C_{ba} \rho.$$

The reversed indices are deliberate. Expanding a coherent combination produces the scalar $\sum_{a,b} c_a \overline{c_b} C_{ba}$. Summing such expressions over a new Kraus family F_j in the old error span gives a common scalar multiplier. If the new family is normalized, trace preservation forces that multiplier to one on a nonzero code. For the zero code, the only supported operator is again zero.

Source: `formalization/ErrorSpanRecovery.lean`.

Exact lines 138 to 165. Excerpt `kl_span`. Hash in Section 12.

```
theorem recovery_span_channel (F : J → Matrix n n ℂ) (c : J → I → ℂ)
    (hc : ∀ j, F j=∑ a, c j a • K.error a)
    (hF : (∑ j, star (F j)*F j)=1)
    (rho : Matrix n n ℂ) (hrho : K.code*rho*K.code=rho) :
    K.diagonal.syndromes.decoder (noise F rho)=rho := by
  let q : ℂ := ∑ j, ∑ a, ∑ b, c j a*star (c j b)*K.gramMatrix b a
  have hscaled (rho : Matrix n n ℂ) (hs : K.code*rho*K.code=rho) :
      K.diagonal.syndromes.decoder (noise F rho)=q • rho := by
    change K.diagonal.syndromes.decoderLinear (∑ j, F j*rho*star (F j)) = _
    rw [map_sum]
    change (∑ j, K.diagonal.syndromes.decoder (F j*rho*star (F j))) = _
    simp_rw [hc, K.recovery_error_superposition rho hs]
    rw [← Finset.sum_smul]
  rw [hscaled rho hrho]
  by_cases hp : K.code=0
  · have hz : rho=0 := by simpa [hp] using hrho.symm
    simp [hz]
  · have hpos : K.code.PosSemidef := by
      have hx := Matrix.posSemidef_conjTranspose_mul_self K.code
      change (star K.code*K.code).PosSemidef at hx
      rw [K.code_projection.isSelfAdjoint.star_eq, K.code_projection.isIdempotentElem.eq] at hx
      exact hx
    have ht : Matrix.trace K.code ≠ 0 := (hpos.trace_eq_zero_iff).not.mpr hp
    have hs : K.code*K.code*K.code=K.code := by simp only [K.code_projection.isIdempotentElem.eq]
    have hh := congrArg Matrix.trace (hscaled K.code hs)
    rw [K.recovery_preserves_trace, noise_preserves_trace F hF, Matrix.trace_smul] at hh
    have hq : q=1 := mul_right_cancel₀ ht (by simpa using hh.symm)
    rw [hq, one_smul]
```

This is one fixed recovery correcting every normalized channel in the error span, with the coefficients explicitly witnessing span membership. It does not claim correction with scale one for an arbitrary unnormalized linear combination. For unnormalized error families, the separate criterion requires a nonnegative scale t independent of the supported input operator and proves equivalence with the Gram relation.

Source: `formalization/ErrorSpanRecovery.lean`.

Exact lines 188 to 209. Excerpt `kl_family`. Hash in Section 12.

```
theorem error_family_correctable_iff_gram (P : Matrix n n ℂ) (hP : IsStarProjection P)
    (E : I → Matrix n n ℂ) :
    ErrorFamilyCorrectable P E ↔ ∃ C : Matrix I I ℂ, C.PosSemidef ∧
      ∀ a b, P*star (E a)*E b*P=C a b • P := by
  constructor
  · rintro ⟨m,R,hR,t,ht,hcorrect⟩
    exact gram_of_scaled_recovery P hP E R hR t ht hcorrect
  · rintro ⟨C,hC,hgram⟩
    let K : Data n I := ⟨P,hP,E,C,hC,hgram⟩
    let e := Fintype.equivFin (Option K.diagonal.Active)
```

```
    let R := K.diagonal.syndromes.recoveryKraus
    refine (Fintype.card (Option K.diagonal.Active), fun r => R (e.symm r), ?_,
      (Matrix.trace C).re, (RCLike.nonneg_iff.mp hC.trace_nonneg).1, ?_)
    · exact (e.symm.sum_comp (fun a => star (R a)*R a)).trans K.recovery_complete
    · intro rho hrho
      change (∑ r, R (e.symm r)*noise E rho*star (R (e.symm r))) = _
      rw [e.symm.sum_comp (fun a => R a*noise E rho*star (R a))]
      have hreal : ((Matrix.trace C).re : ℂ)=Matrix.trace C := by
        exact Complex.ext rfl (by simpa using (RCLike.nonneg_iff.mp hC.trace_nonneg).2.symm)
      rw [hreal]
      exact (K.diagonal.syndromes.decoder_eq_kraus (noise E rho)).symm.trans
        (K.recovery_noise_scaled rho hrho)
```

The example module checks illustrative finite cases, including singular Gram matrices and Gram matrices that are not diagonal. Those examples help expose conventions, but the general theorem rests on spectral reduction, normalized syndrome recovery, necessity through the rank-one argument, and the bridge from density matrices to operators. The resulting theorem is the full criterion for finite complex matrices under its stated hypotheses. The earlier construction for orthogonal syndromes was one stage of this proof.

7 Effective resolvents, Euclidean transfer, and finite bounds

7.1 A resolvent identity needs two invertibility hypotheses

Split the state index into a retained type P and an eliminated type Q. Let the full operator have blocks $A_{pp}, A_{pq}, A_{qp}, A_{qq}$. The eliminated resolvent block is $D = z1_Q - A_{qq}$, and the Schur complement is

$$S(z) = z1_P - A_{pp} - A_{pq}D^{-1}A_{qp}.$$

The theorem identifies the retained corner of the full inverse with $S(z)^{-1}$. The two assumptions supplied through bracketed type classes assert invertibility of D and invertibility of the full resolvent matrix.

Source: `formalization/SchurFeshbach.lean`.

Exact lines 27 to 44. Excerpt `feshbach`. Hash in Section 12.

```
theorem projected_resolvent (App : Matrix P P ℂ) (Apq : Matrix P Q ℂ)
    (Aqp : Matrix Q P ℂ) (Aqq : Matrix Q Q ℂ) (z : ℂ)
    [Invertible (Matrix.scalar Q z-Aqq)]
    [Invertible (Matrix.scalar (P ⊕ Q) z-Matrix.fromBlocks App Apq Aqp Aqq)] :
    ((Matrix.scalar (P ⊕ Q) z-Matrix.fromBlocks App Apq Aqp Aqq)⁻¹).submatrix
      Sum.inl Sum.inl = (schur App Apq Aqp Aqq z)⁻¹ := by
  let D := Matrix.scalar Q z-Aqq
  let A := Matrix.scalar P z-App
  have he := resolvent_blocks App Apq Aqp Aqq z
  let : Invertible (Matrix.fromBlocks A (-Apq) (-Aqp) D) := Invertible.copy ‹_› _ he.symm
  let : Invertible (A-(-Apq)*⅟D*(-Aqp)) := Matrix.invertibleOfFromBlocks₂₂Invertible A (-Apq)
   ↪  (-Aqp) D
  have hinv := Matrix.invOf_fromBlocks₂₂_eq A (-Apq) (-Aqp) D
  have hs : A-(-Apq)*⅟D*(-Aqp)=schur App Apq Aqp Aqq z := by
    simp only [Matrix.neg_mul,Matrix.mul_neg,neg_neg,Matrix.invOf_eq_nonsing_inv,schur,D,A]
  rw [he,← Matrix.invOf_eq_nonsing_inv,hinv]
  change ⅟(A-(-Apq)*⅟D*(-Aqp))=(schur App Apq Aqp Aqq z)⁻¹
  rw [Matrix.invOf_eq_nonsing_inv]
  congr 1
```

Contract. Both finite types of state indices have decidable equality. The eliminated block and the full resolvent matrix are invertible. Lean's matrix inverse notation is a total operation, but that notation alone does not supply genuine invertibility at a singular matrix. The theorem uses the invertibility assumptions to justify the actual inverse identities.

Proof reading. First rewrite $z1 - A$ as the displayed block matrix, retaining the negative signs on the two off-diagonal blocks. Copy the given full invertibility instance to that rewritten matrix. A Mathlib theorem for block inverses supplies invertibility of its Schur complement and the full inverse formula. Rewrite the inverse supplied by the invertibility instance as the nonsingular matrix inverse, then select the retained submatrix. The two negative off-diagonal factors multiply to the sign already encoded in the definition of `schur`.

The source retains a concrete counterexample to dropping full invertibility: at $z = 0$ with $H = \mathrm{diag}(0, -1)$, the eliminated resolvent block is 1, while the full matrix $z1 - H = \mathrm{diag}(0, 1)$ is singular.

Source: `formalization/SchurFeshbach.lean`.

Exact lines 59 to 64. Excerpt `feshbach_counterexample`. Hash in Section 12.

```
theorem eliminated_invertible_full_singular :
    IsUnit (1 : ℂ) ∧ ¬ IsUnit (!![0, 0; 0, 1] : Matrix (Fin 2) (Fin 2) ℂ) := by
  constructor
  · exact isUnit_one
  · rw [Matrix.isUnit_iff_isUnit_det]
    norm_num [Matrix.det_fin_two]
```

The companion `schur_invertible` theorem says that, when the eliminated block is invertible, full invertibility is equivalent to invertibility of the Schur complement. This locates the remaining pole rather than assuming it absent. The corrected assumption is part of the checked interpretation of the traditional result. This source reuses the retained Paper VI proof. The present edition does not rerun the earlier paper.

7.2 Euclidean transfer is defined by the exponential series

The transfer is

$$T_e(t) = \exp(-t(H - e1)).$$

The definition uses the actual matrix exponential. For Hermitian H, a theorem from continuous functional calculus identifies it with the scalar function $x \mapsto e^{-t(x-e)}$ applied to H.

Source: `formalization/EuclideanTransfer.lean`.

Exact lines 39 to 47. Excerpt `euclidean_cfc`. Hash in Section 12.

```
theorem transfer_cfc (H : Matrix n n ℂ) (hH : H.IsHermitian) (e t : ℝ) :
    transfer H e t = cfc (fun x : ℝ => Real.exp ((-t)*(x-e))) H := by
  rw [cfc_comp' Real.exp (fun x : ℝ => (-t)*(x-e)) H]
  have hlin : cfc (fun x : ℝ => (-t)*(x-e)) H = (-t) • (H - e • (1 : Matrix n n ℂ)) := by
    rw [cfc_const_mul (-t) (fun x : ℝ => x-e) H,
      cfc_sub (fun x : ℝ => x) (fun _ : ℝ => e) H, cfc_id' ℝ H hH.isSelfAdjoint, cfc_const e H
      ↪ hH.isSelfAdjoint]
    simp only [Algebra.algebraMap_eq_smul_one]
  rw [hlin, CFC.real_exp_eq_normedSpace_exp (shifted_hermitian H hH e t).isSelfAdjoint]
  rfl
```

Proof reading. The proof composes the functional calculus for the exponential with the affine scalar function $-t(x - e)$. It evaluates that affine function through the calculus as $-t(H - e1)$. A theorem relating real continuous functional calculus to the exponential in a normed space identifies the final object with the original power series. The spectral conclusions therefore apply to the actual transfer, not an independently postulated spectral surrogate.

The shift e_0 is the minimum of the actual Hermitian eigenvalues. It is defined only when the finite state index is nonempty, because an empty spectrum has no minimum element. The source proves both membership of e_0 in the spectrum and the assertion that it is a lower bound. Positive definiteness of $T_e(t)$ follows from strict positivity of the scalar exponential for every real t. Semigroup composition follows because the two scaled generators commute.

Source: `formalization/EuclideanTransfer.lean`.

Exact lines 89 to 96. Excerpt `euclidean_endpoint`. Hash in Section 12.

```
theorem finite_euclidean_transfer [Nonempty n] (H : Matrix n n ℂ)
    (hH : H.IsHermitian) (t s : ℝ) (ht : 0 ≤ t) :
    (transfer H (groundEnergy hH) t).PosDef ∧
    transfer H (groundEnergy hH) (t+s) =
      transfer H (groundEnergy hH) t * transfer H (groundEnergy hH) s ∧
    spectrum ℝ (transfer H (groundEnergy hH) t) ⊆ Set.Ioc 0 1 := by
  exact ⟨transfer_posDef H hH _ _, transfer_add H _ _ _,
    transfer_spectrum_interval H hH _ _ ht (ground_energy_le hH)⟩
```

Contract. The endpoint assumes a nonempty finite state index, Hermitian H and nonnegative t. It includes $t = 0$. The other time s is arbitrary real in the semigroup identity. Spectral containment in $(0, 1]$ uses $t \geq 0$ and the shift by the ground energy. A separate endpoint states containment for the complex spectrum as the real interval's complex image. No infinite-dimensional transfer or continuum limit is concluded.

7.3 The low-mode bound keeps its exact constant

The dispersion expression for the finite circle is $\lambda_h(w) = 4h^{-2}\sin^2(wh/2)$. The source first proves

$$0 \leq x^2 - \sin^2 x \leq \frac{x^4}{3}$$

for every real x, using the quadratic upper bound for sine squared, the cubic error bound for $x - \sin x$ and the estimate $|x + \sin x| \leq 2|x|$. Substituting $x = wh/2$ yields

$$0 \leq w^2 - \lambda_h(w) \leq \frac{w^4}{12}h^2.$$

Source: `formalization/LowModeConvergence.lean`.

Exact lines 32 to 45. Excerpt `low_mode`. Hash in Section 12.

```
theorem low_mode_error (w h : ℝ) (hh : h≠0) :
    0 ≤ w^2-eigenvalue w h ∧ w^2-eigenvalue w h ≤ w^4/12*h^2 := by
  have hs := sine_square_error (w*h/2)
  have hpos : 0 ≤ 4/h^2 := by positivity
  have he : w^2-eigenvalue w h=4/h^2*((w*h/2)^2-Real.sin (w*h/2)^2) := by
    unfold eigenvalue
    field_simp
    ring
  rw [he]
  constructor
  · exact mul_nonneg hpos hs.1
  · calc
      _ ≤ 4/h^2*((w*h/2)^4/3) := mul_le_mul_of_nonneg_left hs.2 hpos
      _ = w^4/12*h^2 := by field_simp; ring
```

Proof reading. The proof scales the sine inequality by the nonnegative quantity $4/h^2$. A field calculation identifies the scaled error with $w^2 - \lambda_h(w)$. The final algebra simplifies its upper bound to the coefficient $1/12$. The hypothesis that the spacing is nonzero justifies division. The conclusion holds for every real w, without a restriction to small frequencies.

For a fixed frequency, the right-hand side tends to zero quadratically as h tends to zero through nonzero values. A growing frequency range needs control of w^4h^2. The theorem by itself does not prove convergence of an entire spectrum, a plateau in spectral dimension, a dynamics limit or a spacetime reconstruction. Those applications require their own quantified range and analytic arguments.

7.4 A compressed anomaly can vanish for purely algebraic reasons

The constraint audit sits beside the 19 finite ledger items. If every constraint annihilates P on the right, then every product term in the proposed anomaly also annihilates P on the right, so $P\mathcal{A}P = 0$ for arbitrary structure coefficients. The source needs no projector axioms for this observation.

Source: `formalization/ConstraintAudit.lean`.

Exact lines 31 to 34. Excerpt `constraint_kernel`. Hash in Section 12.

```
theorem compressed_anomaly_zero (C : I → R) (f : I → R)
    (P : R) (hP : ∀ c, C c * P = 0) (a b : I) :
    P * anomaly C f a b * P = 0 := by
  rw [mul_assoc, anomaly_annihilates_common_kernel C f P hP, mul_zero]
```

It also constructs a nonzero physical projector and two rational 3×3 constraints that act on its complement. Their commutator is nonzero, yet its physical compression is zero.

Source: `formalization/ConstraintAudit.lean`.

Exact lines 56 to 60. Excerpt `constraint_nonclosure`. Hash in Section 12.

```
theorem commutator_nonzero : constraintX * constraintZ - constraintZ * constraintX ≠ 0 := by
  intro h
  have h12 := congrArg (fun A : Matrix (Fin 3) (Fin 3) ℚ => A 1 2) h
  norm_num [constraintX, constraintZ, Matrix.mul_apply, Fin.sum_univ_succ] at h12
  norm_num [Matrix.cons_val_two] at h12
```

This supplies a concrete scope warning: exact compression to a common kernel cannot by itself certify operator closure on the whole space. It does not prove that an intended continuum constraint algebra exists or that every refinement has the same behavior. The hypotheses of the finite audit must be checked before applying it to a proposed physical construction.

7.5 The broader manuscript and the five open targets

The preserved original paper and the strengthened traditional edition discuss memory, geometry, diagnostics and physical interpretation beyond the named finite endpoints. A traditional proof can be more explanatory than a formal theorem statement, and a strengthened mathematical statement can go beyond the selected formal endpoint. Coverage is assigned by the full result map, not by similarity of wording or proximity on a page.

The original manuscript names five continuum conjectures: refinement without anomalies, uniform Feshbach control, Lorentzian reconstruction, an Einstein infrared limit and a massless sector with spin two. None is a theorem target among the 19 ledger items. Finite path reconstruction, finite recovery, a projected connection, a resolvent identity and a dispersion error bound do not jointly prove those existence and limiting claims. The formal source provides rigorous finite ingredients and, in the geometric case, a broad calculus on parameter manifolds under its stated assumptions.

The complete theorem catalog includes seven declarations from the constraint audit, four Schur–Feshbach declarations and the other supporting lemmas. The count 323 therefore means named declarations in the current finite development, not 323 independently stated claims in the traditional paper, and not 323 continuum results.

8 Evidence and reproducible replay

8.1 What the saved compiler record establishes

The inherited run is `verification/final_finite_release_run/SUMMARY.json` in the unchanged v0.6.0 snapshot. It began on 6 September 2026 at 15:30:24 UTC and was updated at 15:32:49.641452 UTC. It records successful compilation of all 25 current modules, all 323 named theorem axiom audits, and two deliberate rejected controls. Current static reconciliation checks the source hashes and saved log hashes against those records.

The allowed axiom set is `propext`, `Classical.choice`, and `Quot.sound`. Every reported theorem dependency set is a subset of it. One reported theorem has no axiom dependencies. These are foundational dependencies accepted in the recorded Lean environment. They do not remove manuscript hypotheses such as invertibility of the full resolvent or Kraus normalization. Axiom auditing and hypothesis reading answer different questions.

The separate inherited `CONSTRUCTED_OBJECTS_AUDIT.json`, in the directory for the same release run, records six constructed objects: the fiber, projection to the fiber, local product homeomorphism, local continuous linear fiber equivalence, covariant derivative taking values in the fiber, and fiber curvature. Definitions can contain proofs of inverse, linearity or continuity fields, so an inventory of theorem names alone would miss part of this evidence. The audit of six objects remains separate from the count of 323 theorems. The computation report dated 9 October 2026 records 12 correspondence checks among the inherited audit, source, inventory, runner, transcript and cumulative summary. Those checks added no theorem declarations and did not execute the Lean compiler.

The source archive omits the compiled public and private interfaces, the portable compiler kit and the Mathlib cache. The saved success record is inherited compilation evidence. It is not a fresh compilation on this host or an independent recheck of all imported Mathlib proofs. The recorded release reused its pinned library environment and did not rebuild earlier papers.

8.2 Source and document checks in this edition

The guide resolves nested namespaces in active source and derives 323 distinct theorem declarations across 25 modules. That set matches the saved theorem inventory and every saved theorem axiom report. All 19 finite ledger entries resolve, including `GlobalConnection.covariant` among the geometric supporting endpoints. Where a ledger entry gives only a module, the guide adds explicitly identified source pointers. It preserves the original ledger.

All 67 selected excerpts are exact contiguous source blocks and retain exactly the bytes of the retrieved guide's raw UTF-8 selections. Their line ranges, original module hashes and excerpt hashes accompany this document. Rendered code uses explicit font fallback for the mathematical script letters and small square symbol used for dot products absent from the monospaced font. The raw source is unchanged. The PDF checks cover every theorem name, excerpt, reference, missing glyph and text bound.

The local `static_check.py` checks saved source and log identities, transitive local import hashes, declarations, dependency pins, inherited rejection records and the audit of six constructed objects. It also compares every delimited mathematical span in the six teaching chapters with the retrieved edition. The prose is checked against a preserved baseline and an explicit record of the approved changes. That record covers each complete chapter, including its hypotheses and explanations of proofs. These checks establish editorial preservation and consistency with the source. They do not compile Lean or validate a new proof.

8.3 The finite computations recorded on 9 October 2026

The report `verification/monograph_checks/COMPUTATIONAL_REVIEW.json` records the finite computations performed on 9 October 2026. Its six records retain the commands, environment versions, limits, inputs, outputs and hashes of those executions. The original and corrected numerical checks used Python 3.12.14, NumPy 2.3.5 and SciPy 1.17.0. The symbolic checks used SymPy 1.14.0. Statements about unavailable dependencies in the retained reports of 6 October describe the host used at that earlier date. The present proofreading pass preserves the October 9 evidence without repeating those numerical or symbolic executions.

In that recorded run, the original companion passed 22 pytest cases. A second entry point called the same 22 functions directly, so the report contains 22 distinct tests. The verifier software passed 16 regression methods, comprising nine positive methods and seven controls using malformed inputs or tampering. These Python controls differ from the two inherited deliberate Lean rejections. No fresh kernel run or compiler rejection test was possible during this proofreading pass because `lean`, `lake` and `elan` are absent.

The report records an exact match for all 1,048 stored result leaves, including 1,045 numerical fields, against the original experiment replay. The recorded comparison bounds are absolute 10^{-10} and relative 10^{-8}, although equality made those allowances unnecessary in the recorded run. The unchanged strengthened suite passed 18,049 checks in 15 groups. Its separately corrected copy passed 17,699 checks in the same 15 groups, with 87 exact mode-index assertions for $N = 4, \dots, 32$. The two lower endpoints were changed from `-N//2` to `-(N//2)`. For odd `N`, Python floor division otherwise adds a repeated residue representative. The difference of 350 checks reflects corrected bookkeeping. The raw source remains intact, and the two overlapping suite totals are not added.

The independent exact rational witnesses contain 269 assertions in total, including three explicit negative controls and a separate singular-inverse rejection benchmark. The original checks of visual examples executed 180 SymPy assertions, including nine rejected controls and 171 positive assertions. The extended checks executed 373 positive exact assertions and 12 rejected controls over finite examples using rational and Gaussian rational numbers. All 38 vector plate sources regenerated with identical bytes, comprising 26 original plates and 12 extended plates. Regenerating the plates checks consistency of the presentation and does not add mathematical proof coverage.

A separate audit, `verification/monograph_checks/TRADITIONAL_EXACT_MODELS_REVIEW.json`, records checks of the three traditional illustrations added in that edition. It passed 30 positive exact assertions and rejected six deliberately false assertions. The model JSON and three generated PDFs reproduced with identical bytes. These are fixed examples of seed support, block inversion and channel scaling. They add no formal targets or general recovery conclusion.

The manifest records retain the actual finite ranges, conventions and tolerances. Original numerical checks typically use bounds from 10^{-12} to 10^{-10}, with a 10^{-6} certificate for moving frames. The strengthened base tolerance is 2×10^{-9}, with individual bounds through 2×10^{-7}. Exact arithmetic gates have no numerical tolerance. These executions overlap and use different count units. Their totals are neither summed as independent evidence nor interpreted as universal theorems. They do not settle any of the five physical continuum targets.

Evidence	Time or source	Meaning
25 module compiler exits	Saved 6 September release	Inherited compilation under the locked toolchain.

Evidence	Time or source	Meaning
323 theorem axiom records	Saved release, currently reconciled	Current source declarations agree with the stored audited set.
Audits of six constructed objects	Separate saved release record	Proof fields in definitions are audited separately.
Source, excerpt and document checks	Proofreading on 18 October	Exact source correspondence, reviewed prose and readable navigation.
22 pytest cases, 16 verifier methods	Recorded 9 October execution	Software checks at the recorded inputs and conditions.
Numerical and exact witness checks	Recorded 9 October executions	Finite computations with explicit arithmetic and assumptions.
38 vector source regenerations	Recorded 9 October replay	Deterministic plate sources, with no added theorem coverage.

8.4 Rebuild only this guide

The guide directory contains the authored TeX, frozen navigation, rendered code, raw excerpts, styles and font licenses. Font binaries are not included in this delivery. The styles use the matching bundled fonts when those are supplied and otherwise use the named installed fonts. The script `build.py` copies the complete source tree to an isolated temporary directory. It performs four XeLaTeX passes to resolve the document structure, then runs `xdvipdfmx` once to produce the PDF. It compiles the authored document without regenerating prose, source data, figures or mathematical evidence, and it does not invoke Lean.

```
python editable/lean_guide/build.py
python editable/lean_guide/build.py --output-directory /path/to/pdf_rebuild
python editable/lean_guide/build.py --check-only
```

Run these commands from the package root, or supply an absolute path to the script. Python 3, XeLaTeX, `xdvipdfmx` and the TeX packages named in the preamble are required. Fonts resolve through the relative `fonts/` directory when the matching binaries are present. Otherwise the styles use installed P052, Nimbus Sans, Latin Modern and DejaVu Sans Mono fonts. The optional `--output-directory` keeps delivered products unchanged, and `--check-only` writes no delivered products.

Regenerating the source tables additionally requires FontTools and the preserved snapshot. The assembler uses the bundled DejaVu Sans Mono and Latin Modern Math files when present, or matching installed files otherwise. Run the following commands only when regenerating the tables as part of an edited release. Refresh the recorded checks and document hashes afterward.

```
python editable/lean_guide/assemble.py
python editable/lean_guide/static_check.py
python editable/lean_guide/build.py
python editable/lean_guide/verify_pdf.py
```

Both source scripts accept `--source PATH` for a relocated snapshot. Assembly stops if counts, inventory, saved axiom sets or source identities disagree. A new source edition requires a deliberate revision of the expected counts and endpoint selection together with its own evidence. These scripts reconcile documentation. They are not proof checkers.

8.5 Replay the formal development with the pinned kit

With the separately supplied pinned portable kit, set the working directory to `source/complete/Paper_VII_Complete_v0.6.0_With_Visual_Appendix/original_snapshot`, relative to the package root. Choose a new output directory to preserve the inherited evidence.

```
python scripts/verify.py --kit /path/to/kit \
  --out verification/my_run --timeout 240
python scripts/audit_constructions.py --kit /path/to/kit \
  --run verification/my_run
python scripts/test_verifier.py
```

The verifier checks runtime version and Mathlib revision, then compiles local modules in dependency order with one worker and strict warnings. It compares emitted axiom audits with the exact inventory and checks both deliberate invalid controls. The construction audit uses the fresh run's actual build interfaces. It rejects stale source or interface identities.

For short execution windows, the retained checkpoint mechanism can pause after a few checks:

```
python scripts/verify.py --kit /path/to/kit \
  --out verification/my_run --timeout 240 \
  --max-checks 3 --resume
```

Repeat until the verifier reports complete success. Exit code 2 means paused or incomplete. Resume checks the verifier, runner, inventory, source dependencies and both public and private interfaces before trusting a cached module. An interrupted build remains incomplete even when earlier modules passed.

The `check_low_memory.py` launcher uses ordinary Lean with one worker. Its name does not promise an enforced memory cap. The snapshot is not a delivered Lake project with a current `lakefile`. Its provenance manifest supplies the portable kit's locked dependencies. A bare `lake build` would not reproduce the retained procedure.

8.6 The wrapper in the retained complete bundle

Set the working directory to `source/complete/Paper_VII_Complete_v0.6.0_With_Visual_Appendix`. Its wrapper provides three actions:

```
python Run-PaperVII.py verify
python Run-PaperVII.py examples --output-dir /path/to/example_replay
python Run-PaperVII.py original --output-dir /path/to/original_replay
```

The integrity action verifies the manifest and checksums. The example action runs the original example suite for the visual appendix with its declared dependencies. The original action runs the original

workflow for numerical examples, figures, TeX and verifier software. Its own documentation identifies that action as a companion replay rather than Lean execution. The October 9 report records the individual checks that were actually executed. It does not claim that the original combined PowerShell procedure, the full figure generation procedure or the historical manuscript rebuild ran as a single workflow.

All nine dependency revisions and the compiler commit appear in the dependency appendix. Source and transcript hashes bind this reading to its inherited artifacts. Document construction, finite computation and kernel replay each have a distinct purpose. The release records their outcomes separately.

9 Complete map of finite results

The following map reproduces all 19 ledger scope statements verbatim and resolves their current formal endpoints. Their recorded completion is inherited from the source release under the stated hypotheses. It does not assert that every sentence in the traditional paper is separately formalized. Source pointers added where the ledger gives only a module are identified explicitly.

9.1 2.1 / Orbit dimension of one quantum seed

Preserved ledger scope. Actual matrix multiplication orbit, explicit equivalence with independently chosen rows, d times rank(C), and full-rank cyclicity when d is nonempty. No orbit-dimension hypothesis is assumed.

Formal endpoints.

- `KomorebiPaperVII.SeedOrbit.span_orbit_dimension`, `SeedOrbit.lean`, line 68.

9.2 2.2 / Quantum seed-rank formula

Preserved ledger scope. Actual sums of k matrix actions, full-column-rank seed construction, simultaneous finite direct-sum generation, and an IsLeast proof for max_i ceil(m_i/d_i). Every d_i is positive. Empty block families use supremum 0.

Formal endpoints.

- `KomorebiPaperVII.SeedOrbit.quantum_seed_rank`, `SeedOrbit.lean`, line 239.

9.3 2.3 / Local orbit and Schmidt rank

Preserved ledger scope. Actual Kronecker local action in standard product coordinates. Schmidt rank is independently defined as minimum product-vector decomposition length, with its equality to matrix rank proved. Derives local orbit dimension and equivalence with full support of the actual B-reduced matrix; the full-orbit equivalence assumes a nonempty A index.

Formal endpoints.

- `KomorebiPaperVII.SeedOrbit.local_orbit_reduced_support`, `SeedOrbit.lean`, line 364.

9.4 2.4 / One seed for the computational basis

Preserved ledger scope. Explicit local Pauli X addressing for every n, including n=0. The constructed gate word sends the zero seed to the standard coordinate basis, with linear independence, span and cardinality 2^n.

Formal endpoints.

- `KomorebiPaperVII.ComputationalSeed.circuit_seed_basis`, `ComputationalSeed.lean`, line 80.

9.5 3.1 / Cyclic quantum projectors

Preserved ledger scope. Constructs the primitive complex character and rho(r)=S^val(r) from S^N=1 for every N>0, including N=1. Proves exact exponential-root coefficient formula, character orthogonality, multiplication, completeness, eigenaction, and self-adjointness for unitary S. Includes a concrete complex-matrix theorem.

Formal endpoints.

- `KomorebiPaperVII.CyclicProjectors.finite_cyclic_projectors`, `CyclicProjectors.lean`, line 157.

9.6 3.2 / Tensor/polyphase refinement

Preserved ledger scope. Arbitrary finite tensor factors and dimensions. Proves tensor product multiplication, identity, adjoint and sum laws, then orthogonality, completeness and self-adjointness after grouping by total residue for N>0.

Formal endpoints.

- `KomorebiPaperVII.TensorRefinement.tensor_polyphase_refinement`, `TensorRefinement.lean`, line 139.

9.7 4.1 / Exact circuit path expansion

Preserved ledger scope. Arbitrary finite gate list, corner expansion and complete reconstruction.

Source pointers added by the guide.

- `KomorebiPaperVII.PacketFamily.circuit_path_expansion`, `QuantumPacketCore.lean`, line 63.
- `KomorebiPaperVII.PacketFamily.circuit_reconstruction`, `QuantumPacketCore.lean`, line 70.

9.8 4.2 / Packetwise unitarity conservation

Preserved ledger scope. Both identities in a star ring, plus concrete complex-matrix specializations.

Source pointers added by the guide.

- `KomorebiPaperVII.PacketFamily.unitarity_columns`, `QuantumPacketCore.lean`, line 119.
- `KomorebiPaperVII.PacketFamily.unitarity_rows`, `QuantumPacketCore.lean`, line 134.
- `KomorebiPaperVII.matrix_unitarity_columns`, `QuantumPacketCore.lean`, line 181.
- `KomorebiPaperVII.matrix_unitarity_rows`, `QuantumPacketCore.lean`, line 188.

9.9 5.1 / Exact symmetry packets

Preserved ledger scope. Generator blocks and actual exponential; zero cross-packet transfer.

Source pointers added by the guide.

- `KomorebiPaperVII.PacketFamily.symmetry_exponential_no_transfer`, `QuantumPacketCore.lean`, line 168.
- `KomorebiPaperVII.PacketFamily.symmetry_exponential_diagonal`, `QuantumPacketCore.lean`, line 163.

9.10 5.2 / Commutator leakage bound

Preserved ledger scope. Actual exponential with self-adjoint H and orthogonal P. Proves the Hilbert operator-norm bound for all real times, without nonzero-rank or nonempty-matrix assumptions. Also proves observable-conjugation Lipschitz continuity, the cap at one, and a symmetry-breaking perturbation estimate.

Formal endpoints.

- `KomorebiPaperVII.Leakage.matrix_commutator_leakage_bound`, `LeakageBound.lean`, line 167.

9.11 6.1 / Moving-frame Hamiltonian

Preserved ledger scope. Entrywise actual real derivatives of finite complex matrices and vectors. Constructs y=U*psi and its derivative, verifies the transformed Schrodinger equation and Hermiticity. The effective-Hamiltonian Hermiticity conclusion explicitly assumes H*=H.

Formal endpoints.

- `KomorebiPaperVII.MovingFrames.moving_frame_coordinates`, `MovingFrames.lean`, line 110.

9.12 6.3 / Canonical packet connection

Preserved ledger scope. Actual derivatives in a normed real star algebra with continuous star. Differentiates projector multiplication, completeness and self-adjointness to prove both formulas for C, skew-adjointness and [C,P_b]=dot(P_b). Does not include manifold curvature.

Formal endpoints.

- `KomorebiPaperVII.Connection.canonical_connection_of_hasDerivAt`, `CanonicalConnection.lean`, line 113.

9.13 6.4 / Projected connection

Preserved ledger scope. Smooth real boundaryless parameter manifolds modeled on a complete normed space; complete real normed fibers, including finite quantum spaces through underlying real calculus. Constructs the actual range total space, open local product homeomorphisms, smooth fiber-linear transitions, inverse and cocycle laws, and locally constant rank. Proves local smooth-section equivalence, the actual global covariant derivative, connection laws and gauge gluing. Derives curvature using actual manifold derivatives and the actual Lie bracket, and proves tensoriality. Uses an explicit smooth atlas rather than a packaged Mathlib ContMDiffVectorBundle instance. No global chart, pre-existing range-bundle atlas or curvature identity is assumed.

Formal endpoints.

- `KomorebiPaperVII.ManifoldCurvature.projected_connection_curvature`, `ManifoldProjectedCurvature.lean`, line 214.
- `KomorebiPaperVII.RangeBundle.smooth_range_bundle_atlas`, `GlobalRangeBundle.lean`, line 241.
- `KomorebiPaperVII.BundleCompletion.localProduct_overlap`, `ProjectedBundleCompletion.lean`, line 31.
- `KomorebiPaperVII.BundleCompletion.smooth_section_iff`, `ProjectedBundleCompletion.lean`, line 40.

- `KomorebiPaperVII.GlobalConnection.covariant`, `GlobalProjectedConnection.lean`, line 44.
- `KomorebiPaperVII.GlobalConnection.local_connections_glue`, `GlobalProjectedConnection.lean`, line 115.
- `KomorebiPaperVII.BundleCompletion.local_connections_gauge`, `ProjectedBundleCompletion.lean`, line 57.

9.14 7.1 / Block-diagonal measurement invariance

Preserved ledger scope. Trace duality, measurement invariance and trace preservation for finite complex matrices.

Source pointers added by the guide.

- `KomorebiPaperVII.measurement_invariance`, `QuantumPacketCore.lean`, line 208.
- `KomorebiPaperVII.dephase_trace_preserving`, `QuantumPacketCore.lean`, line 213.

9.15 7.2 / Finite Kraus channel

Preserved ledger scope. Finite complex matrix Kraus map: linearity, positivity, trace preservation under sum(K* K)=I, identity-tensor amplification at every finite auxiliary dimension, complete positivity and density-matrix preservation.

Formal endpoints.

- `KomorebiPaperVII.Kraus.channel_completely_positive`, `KrausChannels.lean`, line 58.

9.16 8.1 / Finite Knill-Laflamme criterion

Preserved ledger scope. Full finite complex-matrix Knill-Laflamme criterion. Constructs a total completely positive trace-preserving recovery from general positive semidefinite Gram data using actual spectral diagonalization, positive-eigenvalue support and proof that null errors vanish. Derives necessity from actual recovery via rank-one Kraus rigidity. Proves equivalence for normalized code density matrices, preservation with arbitrary finite references, the unnormalized error-family criterion with an explicit nonnegative scale, and correction of every normalized channel in the same error span by the same recovery. Zero code is handled separately. This is a mathematical, noncomputable spectral construction, not a numerical decoder implementation.

Formal endpoints.

- `KomorebiPaperVII.KnillLaflamme.state_correctable_iff_gram`, `QuantumStateBridge.lean`, line 139.
- `KomorebiPaperVII.KnillLaflamme.error_family_correctable_iff_gram`, `ErrorSpanRecovery.lean`, line 188.
- `KomorebiPaperVII.KnillLaflamme.Data.recovery_span_channel`, `ErrorSpanRecovery.lean`, line 138.
- `KomorebiPaperVII.KnillLaflamme.recovery_with_reference`, `QuantumStateBridge.lean`, line 164.

9.17 9.1 / Exact Feshbach resolvent

Preserved ledger scope. Corrected: eliminated block AND full resolvent matrix invertible. Paper VI proof reused.

Source pointers added by the guide.

- `KomorebiPaperVII.SchurFeshbach.projected_resolvent`, `SchurFeshbach.lean`, line 27.
- `KomorebiPaperVII.SchurFeshbach.schur_invertible`, `SchurFeshbach.lean`, line 46.

9.18 10.1 / Finite Euclidean transfer

Preserved ledger scope. Actual power-series matrix exponential, identified with continuous functional calculus. Constructs the minimum eigenvalue on a nonempty finite state space, proves positive definiteness, semigroup composition, and real and complex spectrum containment in (0,1] for nonnegative time.

Formal endpoints.

- `KomorebiPaperVII.EuclideanTransfer.finite_euclidean_transfer`, `EuclideanTransfer.lean`, line 89.

9.19 11.1 / Low-mode second-order bound

Preserved ledger scope. Explicit h != 0; stronger all-real-frequency bound with constant 1/12. Paper VI proof reused.

Source pointers added by the guide.

- `KomorebiPaperVII.LowModeConvergence.low_mode_error`, `LowModeConvergence.lean`, line 32.

10 All named theorem declarations

The following catalog is derived from active declarations in the 25 current source modules, with nested namespaces resolved. It is reconciled against the saved theorem inventory and all 323 saved axiom reports. It includes supporting lemmas, the finite constraint audit, and strengthened endpoints. The count therefore describes formal declarations rather than 323 separate manuscript theorems. The source line identifies the declaration. "Core" marks an endpoint in the map of finite results, while "support" marks a declaration used in the formal development or its scope audit. All saved axiom sets lie within `propext`, `Classical.choice`, and `Quot.sound`. The complete sets accompany the guide with their source identities.

10.1 CanonicalConnection: 9 theorem declarations

Source: `formalization/CanonicalConnection.lean`.

Named declaration	Source line
`KomorebiPaperVII.Connection.canonical_right`	18, support
`KomorebiPaperVII.Connection.canonical_left_relation`	23, support
`KomorebiPaperVII.Connection.canonical_commutator`	39, support
`KomorebiPaperVII.Connection.canonical_skewAdjoint`	48, support
`KomorebiPaperVII.Connection.canonical_half_commutator`	59, support
`KomorebiPaperVII.Connection.differentiated_orthogonality`	78, support
`KomorebiPaperVII.Connection.differentiated_completeness`	98, support
`KomorebiPaperVII.Connection.differentiated_selfAdjoint`	106, support
`KomorebiPaperVII.Connection.canonical_connection_of_hasDerivAt`	113, Core 6.3

10.2 ComputationalSeed: 14 theorem declarations

Source: `formalization/ComputationalSeed.lean`.

Named declaration	Source line
`KomorebiPaperVII.ComputationalSeed.translation_zero`	24, support
`KomorebiPaperVII.ComputationalSeed.translation_add`	27, support
`KomorebiPaperVII.ComputationalSeed.translation_ket`	31, support
`KomorebiPaperVII.ComputationalSeed.translation_involution`	35, support
`KomorebiPaperVII.ComputationalSeed.localX_on_ket`	40, support
`KomorebiPaperVII.ComputationalSeed.controlledX_eq_translation`	51, support
`KomorebiPaperVII.ComputationalSeed.translation_list_product`	63, support
`KomorebiPaperVII.ComputationalSeed.sum_local_addresses`	68, support
`KomorebiPaperVII.ComputationalSeed.circuit_eq_translation`	72, support
`KomorebiPaperVII.ComputationalSeed.circuit_seed`	78, support
`KomorebiPaperVII.ComputationalSeed.circuit_seed_basis`	80, Core 2.4
`KomorebiPaperVII.ComputationalSeed.addressed_seed_linearIndependent`	83, support
`KomorebiPaperVII.ComputationalSeed.addressed_seed_spans`	87, support
`KomorebiPaperVII.ComputationalSeed.number_of_addresses`	91, support

10.3 ConstraintAudit: 7 theorem declarations

Source: `formalization/ConstraintAudit.lean`.

Named declaration	Source line
`KomorebiPaperVII.ConstraintAudit.anomaly_annihilates_common_kernel`	26, support
`KomorebiPaperVII.ConstraintAudit.compressed_anomaly_zero`	31, support
`KomorebiPaperVII.ConstraintAudit.physical_idempotent`	43, support
`KomorebiPaperVII.ConstraintAudit.physical_nonzero`	46, support
`KomorebiPaperVII.ConstraintAudit.constraints_annihilate_physical`	51, support
`KomorebiPaperVII.ConstraintAudit.commutator_nonzero`	56, support
`KomorebiPaperVII.ConstraintAudit.physical_commutator_zero`	62, support

10.4 CyclicProjectors: 18 theorem declarations

Source: `formalization/CyclicProjectors.lean`.

Named declaration	Source line
`KomorebiPaperVII.CyclicProjectors.character_sum`	22, support
`KomorebiPaperVII.CyclicProjectors.packet_eigen`	28, support
`KomorebiPaperVII.CyclicProjectors.packet_mul`	44, support
`KomorebiPaperVII.CyclicProjectors.packet_idempotent`	60, support
`KomorebiPaperVII.CyclicProjectors.packet_complete`	65, support
`KomorebiPaperVII.CyclicProjectors.powers_zero`	76, support
`KomorebiPaperVII.CyclicProjectors.powers_add`	79, support
`KomorebiPaperVII.CyclicProjectors.powers_one`	84, support
`KomorebiPaperVII.CyclicProjectors.cyclic_quantum_projectors`	87, support
`KomorebiPaperVII.CyclicProjectors.star_character`	97, support
`KomorebiPaperVII.CyclicProjectors.packet_selfAdjoint`	104, support
`KomorebiPaperVII.CyclicProjectors.powers_star`	116, support
`KomorebiPaperVII.CyclicProjectors.cyclic_packet_selfAdjoint`	127, support
`KomorebiPaperVII.CyclicProjectors.omega_exponential`	136, support
`KomorebiPaperVII.CyclicProjectors.character_as_power`	140, support
`KomorebiPaperVII.CyclicProjectors.packet_coefficient`	143, support
`KomorebiPaperVII.CyclicProjectors.cyclic_projector_formula`	151, support
`KomorebiPaperVII.CyclicProjectors.finite_cyclic_projectors`	157, Core 3.1

10.5 ErrorSpanRecovery: 9 theorem declarations

Source: `formalization/ErrorSpanRecovery.lean`.

Named declaration	Source line
`KomorebiPaperVII.KnillLaflamme.scaled_corrected_kraus_family`	23, support

Named declaration	Source line
`KomorebiPaperVII.KnillLaflamme.compressed_gram_from_scalar`	48, support
`KomorebiPaperVII.KnillLaflamme.noise_preserves_trace`	65, support
`KomorebiPaperVII.KnillLaflamme.Data.recovery_scalar_composites`	79, support
`KomorebiPaperVII.KnillLaflamme.Data.recovery_cross_error`	94, support
`KomorebiPaperVII.KnillLaflamme.Data.recovery_error_superposition`	112, support
`KomorebiPaperVII.KnillLaflamme.Data.recovery_span_channel`	138, Core 8.1
`KomorebiPaperVII.KnillLaflamme.gram_of_scaled_recovery`	175, support
`KomorebiPaperVII.KnillLaflamme.error_family_correctable_iff_gram`	188, Core 8.1

10.6 EuclideanTransfer: 12 theorem declarations

Source: `formalization/EuclideanTransfer.lean`.

Named declaration	Source line
`KomorebiPaperVII.EuclideanTransfer.shifted_hermitian`	23, support
`KomorebiPaperVII.EuclideanTransfer.transfer_hermitian`	27, support
`KomorebiPaperVII.EuclideanTransfer.transfer_zero`	30, support
`KomorebiPaperVII.EuclideanTransfer.transfer_add`	33, support
`KomorebiPaperVII.EuclideanTransfer.transfer_cfc`	39, support
`KomorebiPaperVII.EuclideanTransfer.transfer_spectrum`	49, support
`KomorebiPaperVII.EuclideanTransfer.transfer_posDef`	54, support
`KomorebiPaperVII.EuclideanTransfer.transfer_spectrum_interval`	65, support
`KomorebiPaperVII.EuclideanTransfer.ground_energy_le`	77, support
`KomorebiPaperVII.EuclideanTransfer.ground_energy_mem`	83, support
`KomorebiPaperVII.EuclideanTransfer.finite_euclidean_transfer`	89, Core 10.1
`KomorebiPaperVII.EuclideanTransfer.transfer_complex_spectrum`	99, support

10.7 GlobalProjectedConnection: 10 theorem declarations

Source: `formalization/GlobalProjectedConnection.lean`.

Named declaration	Source line
`KomorebiPaperVII.GlobalConnection.nabla_in_range`	48, support
`KomorebiPaperVII.GlobalConnection.nabla_add_sections`	53, support
`KomorebiPaperVII.GlobalConnection.nabla_leibniz`	59, support
`KomorebiPaperVII.GlobalConnection.nabla_add_vectors`	65, support
`KomorebiPaperVII.GlobalConnection.nabla_smul_vector`	69, support
`KomorebiPaperVII.GlobalConnection.nabla_germ`	73, support
`KomorebiPaperVII.GlobalConnection.smooth_directional`	81, support
`KomorebiPaperVII.GlobalConnection.smooth_nabla`	88, support
`KomorebiPaperVII.GlobalConnection.localNabla_represents_global`	102, support
`KomorebiPaperVII.GlobalConnection.local_connections_glue`	115, Core 6.4

10.8 GlobalRangeBundle: 22 theorem declarations

Source: `formalization/GlobalRangeBundle.lean`.

Named declaration	Source line
`KomorebiPaperVII.RangeBundle.mem_fiber`	30, support
`KomorebiPaperVII.RangeBundle.project_coe`	36, support
`KomorebiPaperVII.RangeBundle.project_fiber`	39, support
`KomorebiPaperVII.RangeBundle.transport_self`	45, support
`KomorebiPaperVII.RangeBundle.transport_intertwines`	49, support
`KomorebiPaperVII.RangeBundle.transport_inverse_intertwines`	56, support
`KomorebiPaperVII.RangeBundle.mem_domain`	73, support
`KomorebiPaperVII.RangeBundle.isOpen_domain`	76, support
`KomorebiPaperVII.RangeBundle.fromCoord_range`	90, support
`KomorebiPaperVII.RangeBundle.toCoord_fromCoord`	97, support
`KomorebiPaperVII.RangeBundle.fromCoord_toCoord`	105, support
`KomorebiPaperVII.RangeBundle.transition_self`	121, support
`KomorebiPaperVII.RangeBundle.transition_cocycle`	127, support
`KomorebiPaperVII.RangeBundle.smooth_transport`	139, support
`KomorebiPaperVII.RangeBundle.smooth_inverse_transport`	144, support
`KomorebiPaperVII.RangeBundle.smooth_toCoord`	154, support
`KomorebiPaperVII.RangeBundle.smooth_fromCoord`	159, support
`KomorebiPaperVII.RangeBundle.smooth_transition`	164, support
`KomorebiPaperVII.RangeBundle.localProduct_base`	212, support
`KomorebiPaperVII.RangeBundle.localProduct_transition`	217, support
`KomorebiPaperVII.RangeBundle.fiber_rank_locally_constant`	234, support
`KomorebiPaperVII.RangeBundle.smooth_range_bundle_atlas`	241, Core 6.4

10.9 KnillLaflammeExamples: 8 theorem declarations

Source: `formalization/KnillLaflammeExamples.lean`.

Named declaration	Source line
`KomorebiPaperVII.KnillLaflamme.Examples.redundant_complete`	25, support
`KomorebiPaperVII.KnillLaflamme.Examples.redundant_gram_positive`	32, support
`KomorebiPaperVII.KnillLaflamme.Examples.redundant_gram_singular`	36, support
`KomorebiPaperVII.KnillLaflamme.Examples.redundant_gram_off_diagonal`	41, support
`KomorebiPaperVII.KnillLaflamme.Examples.redundant_correctable`	45, support
`KomorebiPaperVII.KnillLaflamme.Examples.dephasing_complete`	57, support
`KomorebiPaperVII.KnillLaflamme.Examples.dephasing_not_correctable`	64, support
`KomorebiPaperVII.KnillLaflamme.Examples.zero_code_correctable`	74, support

10.10 KnillLaflammeNecessity: 7 theorem declarations

Source: `formalization/KnillLaflammeNecessity.lean`.

Named declaration	Source line
`KomorebiPaperVII.KnillLaflamme.rank_one_gram_rigidity`	19, support
`KomorebiPaperVII.KnillLaflamme.kraus_rank_one_rigidity`	52, support
`KomorebiPaperVII.KnillLaflamme.corrected_kraus_family`	68, support
`KomorebiPaperVII.KnillLaflamme.gram_of_recovery`	86, support
`KomorebiPaperVII.KnillLaflamme.correctable_iff_gram`	115, support
`KomorebiPaperVII.KnillLaflamme.not_correctable_of_nonscalar`	133, support
`KomorebiPaperVII.KnillLaflamme.recovery_cross_from_scalar`	142, support

10.11 KnillLaflammeSufficiency: 22 theorem declarations

Source: `formalization/KnillLaflammeSufficiency.lean`.

Named declaration	Source line
`KomorebiPaperVII.KnillLaflamme.mix_restricted`	28, support
`KomorebiPaperVII.KnillLaflamme.mix_gram`	33, support
`KomorebiPaperVII.KnillLaflamme.mix_noise`	45, support
`KomorebiPaperVII.KnillLaflamme.diagonalized_gram`	74, support
`KomorebiPaperVII.KnillLaflamme.DiagonalErrors.zero_weight_error`	107, support
`KomorebiPaperVII.KnillLaflamme.DiagonalErrors.sqrt_weight_ne_zero`	118, support
`KomorebiPaperVII.KnillLaflamme.DiagonalErrors.sqrt_weight_sq`	121, support
`KomorebiPaperVII.KnillLaflamme.DiagonalErrors.normalized_restricted`	125, support
`KomorebiPaperVII.KnillLaflamme.DiagonalErrors.normalized_gram`	128, support
`KomorebiPaperVII.KnillLaflamme.DiagonalErrors.error_eq_sqrt_normalized`	152, support
`KomorebiPaperVII.KnillLaflamme.DiagonalErrors.decode_diagonal_error`	156, support
`KomorebiPaperVII.KnillLaflamme.DiagonalErrors.decode_noise`	175, support
`KomorebiPaperVII.KnillLaflamme.Data.restricted_error_gram`	198, support
`KomorebiPaperVII.KnillLaflamme.Data.restricted_noise`	216, support
`KomorebiPaperVII.KnillLaflamme.Data.diagonal_noise`	227, support
`KomorebiPaperVII.KnillLaflamme.Data.recovery_noise_scaled`	232, support
`KomorebiPaperVII.KnillLaflamme.Data.gram_trace_one`	238, support
`KomorebiPaperVII.KnillLaflamme.Data.recovery_corrects`	252, support
`KomorebiPaperVII.KnillLaflamme.Data.recovery_complete`	262, support
`KomorebiPaperVII.KnillLaflamme.Data.recovery_preserves_trace`	267, support
`KomorebiPaperVII.KnillLaflamme.Data.recovery_completely_positive`	271, support
`KomorebiPaperVII.KnillLaflamme.Data.exists_recovery`	278, support

10.12 KrausChannels: 7 theorem declarations

Source: `formalization/KrausChannels.lean`.

Named declaration	Source line
`KomorebiPaperVII.Kraus.channel_add`	16, support
`KomorebiPaperVII.Kraus.channel_smul`	20, support
`KomorebiPaperVII.Kraus.channel_positive`	29, support
`KomorebiPaperVII.Kraus.channel_trace_preserving`	33, support
`KomorebiPaperVII.Kraus.amplification_eq_kraus`	48, support
`KomorebiPaperVII.Kraus.channel_completely_positive`	58, Core 7.2
`KomorebiPaperVII.Kraus.channel_density_matrix`	64, support

10.13 LeakageBound: 17 theorem declarations

Source: `formalization/LeakageBound.lean`.

Named declaration	Source line
`KomorebiPaperVII.Leakage.exp_unitary`	25, support
`KomorebiPaperVII.Leakage.transported_derivative`	31, support
`KomorebiPaperVII.Leakage.transported_derivative_norm`	40, support
`KomorebiPaperVII.Leakage.transported_lipschitz`	46, support
`KomorebiPaperVII.Leakage.exponential_commutator_bound`	56, support
`KomorebiPaperVII.Leakage.leakage_corner`	67, support
`KomorebiPaperVII.Leakage.leakage_corner_norm`	75, support
`KomorebiPaperVII.Leakage.skew_generator_leakage_bound`	88, support
`KomorebiPaperVII.Leakage.generator_skew`	96, support
`KomorebiPaperVII.Leakage.generator_commutator_norm`	100, support
`KomorebiPaperVII.Leakage.commutator_leakage_bound`	106, support
`KomorebiPaperVII.Leakage.exact_symmetry_zero_leakage`	114, support
`KomorebiPaperVII.Leakage.unitary_corner_norm`	122, support
`KomorebiPaperVII.Leakage.capped_commutator_leakage_bound`	133, support
`KomorebiPaperVII.Leakage.projection_commutator_norm_le`	141, support
`KomorebiPaperVII.Leakage.perturbation_leakage_bound`	151, support
`KomorebiPaperVII.Leakage.matrix_commutator_leakage_bound`	167, Core 5.2

10.14 LocalProjectedCurvature: 6 theorem declarations

Source: `formalization/LocalProjectedCurvature.lean`.

Named declaration	Source line
`KomorebiPaperVII.LocalCurvature.projector_jet`	25, support
`KomorebiPaperVII.LocalCurvature.projector_differential_sandwich`	33, support

Named declaration	Source line
`KomorebiPaperVII.LocalCurvature.section_jet`	38, support
`KomorebiPaperVII.LocalCurvature.nabla_derivative`	47, support
`KomorebiPaperVII.LocalCurvature.curvature_expansion`	60, support
`KomorebiPaperVII.LocalCurvature.projected_connection_curvature`	79, support

10.15 LowModeConvergence: 2 theorem declarations

Source: `formalization/LowModeConvergence.lean`.

Named declaration	Source line
`KomorebiPaperVII.LowModeConvergence.sine_square_error`	15, support
`KomorebiPaperVII.LowModeConvergence.low_mode_error`	32, Core 11.1

10.16 ManifoldProjectedCurvature: 17 theorem declarations

Source: `formalization/ManifoldProjectedCurvature.lean`.

Named declaration	Source line
`KomorebiPaperVII.ManifoldCurvature.derivative_in_chart`	28, support
`KomorebiPaperVII.ManifoldCurvature.differentiable_in_chart`	34, support
`KomorebiPaperVII.ManifoldCurvature.smooth_in_chart`	39, support
`KomorebiPaperVII.ManifoldCurvature.derivative_clm_apply`	49, support
`KomorebiPaperVII.ManifoldCurvature.chartField_self`	65, support
`KomorebiPaperVII.ManifoldCurvature.chartField_differentiable`	74, support
`KomorebiPaperVII.ManifoldCurvature.bracket_in_chart`	80, support
`KomorebiPaperVII.ManifoldCurvature.directional_chart_pullback`	91, support
`KomorebiPaperVII.ManifoldCurvature.second_directional_in_chart`	115, support
`KomorebiPaperVII.ManifoldCurvature.derivative_lieBracket`	132, support
`KomorebiPaperVII.ManifoldCurvature.projector_jet`	150, support
`KomorebiPaperVII.ManifoldCurvature.projector_differential_sandwich`	161, support
`KomorebiPaperVII.ManifoldCurvature.section_jet`	167, support
`KomorebiPaperVII.ManifoldCurvature.nabla_derivative`	178, support
`KomorebiPaperVII.ManifoldCurvature.curvature_expansion`	191, support
`KomorebiPaperVII.ManifoldCurvature.projected_connection_curvature`	214, Core 6.4
`KomorebiPaperVII.ManifoldCurvature.curvature_is_tensorial`	236, support

10.17 MovingFrames: 9 theorem declarations

Source: `formalization/MovingFrames.lean`.

Named declaration	Source line
`KomorebiPaperVII.MovingFrames.matrix_product_derivative`	15, support
`KomorebiPaperVII.MovingFrames.matrix_vector_derivative`	23, support
`KomorebiPaperVII.MovingFrames.adjoint_derivative`	32, support
`KomorebiPaperVII.MovingFrames.differentiated_unitarity`	37, support
`KomorebiPaperVII.MovingFrames.frame_velocity_skew`	52, support
`KomorebiPaperVII.MovingFrames.connection_hermitian`	60, support
`KomorebiPaperVII.MovingFrames.effective_hamiltonian_hermitian`	72, support
`KomorebiPaperVII.MovingFrames.moving_frame_equation`	83, support
`KomorebiPaperVII.MovingFrames.moving_frame_coordinates`	110, Core 6.1

10.18 ProjectedBundleCompletion: 12 theorem declarations

Source: `formalization/ProjectedBundleCompletion.lean`.

Named declaration	Source line
`KomorebiPaperVII.BundleCompletion.total_domain_open`	25, support
`KomorebiPaperVII.BundleCompletion.localProduct_overlap`	31, Core 6.4
`KomorebiPaperVII.BundleCompletion.smooth_section_iff`	40, Core 6.4
`KomorebiPaperVII.BundleCompletion.local_connections_gauge`	57, Core 6.4
`KomorebiPaperVII.BundleCompletion.curvatureForm_self`	73, support
`KomorebiPaperVII.BundleCompletion.curvatureForm_swap`	76, support
`KomorebiPaperVII.BundleCompletion.curvatureForm_add_left`	81, support
`KomorebiPaperVII.BundleCompletion.curvatureForm_smul_left`	86, support
`KomorebiPaperVII.BundleCompletion.curvatureForm_in_range`	90, support
`KomorebiPaperVII.BundleCompletion.fiberCurvature_coe`	103, support
`KomorebiPaperVII.BundleCompletion.constant_projector_flat`	108, support
`KomorebiPaperVII.BundleCompletion.smooth_section_chart_independent`	112, support

10.19 ProjectedCurvature: 20 theorem declarations

Source: `formalization/ProjectedCurvature.lean`.

Named declaration	Source line
`KomorebiPaperVII.Curvature.jet_sandwich_zero`	22, support
`KomorebiPaperVII.Curvature.nabla_in_range`	47, support
`KomorebiPaperVII.Curvature.projector_jet`	52, support
`KomorebiPaperVII.Curvature.projector_differential_sandwich`	60, support
`KomorebiPaperVII.Curvature.section_jet`	65, support

Named declaration	Source line
KomorebiPaperVII.Curvature.nabla_derivative	74, support
KomorebiPaperVII.Curvature.curvature_expansion	87, support
KomorebiPaperVII.Curvature.projected_connection_curvature	106, support
KomorebiPaperVII.Curvature.curvatureForm_alternating	123, support
KomorebiPaperVII.Curvature.curvatureForm_antisymmetric	126, support
KomorebiPaperVII.Curvature.curvatureForm_left_range	131, support
KomorebiPaperVII.Curvature.curvatureForm_right_range	136, support
KomorebiPaperVII.Curvature.curvatureForm_pullback	142, support
KomorebiPaperVII.Curvature.nabla_scalar_leibniz	149, support
KomorebiPaperVII.Curvature.nabla_add_vectorfields	157, support
KomorebiPaperVII.Curvature.nabla_smul_vectorfield	162, support
KomorebiPaperVII.Curvature.curvatureForm_add_left	167, support
KomorebiPaperVII.Curvature.curvatureForm_add_right	172, support
KomorebiPaperVII.Curvature.curvatureForm_smul_left	177, support
KomorebiPaperVII.Curvature.curvatureForm_smul_right	181, support

10.20 QuantumPacketCore: 19 theorem declarations

Source: `formalization/QuantumPacketCore.lean`.

Named declaration	Source line
KomorebiPaperVII.PacketFamily.idempotent	32, support
KomorebiPaperVII.PacketFamily.reconstruct	38, support
KomorebiPaperVII.PacketFamily.compose_blocks	41, support
KomorebiPaperVII.PacketFamily.circuit_path_expansion	63, Core 4.1
KomorebiPaperVII.PacketFamily.circuit_reconstruction	70, Core 4.1
KomorebiPaperVII.PacketFamily.symmetry_no_transfer	76, support
KomorebiPaperVII.PacketFamily.symmetry_diagonal	80, support
KomorebiPaperVII.PacketFamily.dephase_idempotent	90, support
KomorebiPaperVII.PacketFamily.adjoint_block	114, support
KomorebiPaperVII.PacketFamily.unitarity_columns	119, Core 4.2
KomorebiPaperVII.PacketFamily.unitarity_rows	134, Core 4.2
KomorebiPaperVII.PacketFamily.symmetry_exponential_commutes	156, support
KomorebiPaperVII.PacketFamily.symmetry_exponential_diagonal	163, Core 5.1
KomorebiPaperVII.PacketFamily.symmetry_exponential_no_transfer	168, Core 5.1
KomorebiPaperVII.matrix_unitarity_columns	181, Core 4.2
KomorebiPaperVII.matrix_unitarity_rows	188, Core 4.2
KomorebiPaperVII.dephase_trace_duality	195, support
KomorebiPaperVII.measurement_invariance	208, Core 7.1
KomorebiPaperVII.dephase_trace_preserving	213, Core 7.1

10.21 QuantumStateBridge: 9 theorem declarations

Source: `formalization/QuantumStateBridge.lean`.

Named declaration	Source line
`KomorebiPaperVII.KnillLaflamme.outer_polarization`	21, support
`KomorebiPaperVII.KnillLaflamme.linearMap_eq_of_pure`	30, support
`KomorebiPaperVII.KnillLaflamme.sandwich_pure`	64, support
`KomorebiPaperVII.KnillLaflamme.fixes_positive_of_density`	70, support
`KomorebiPaperVII.KnillLaflamme.fixes_operators_of_density`	100, support
`KomorebiPaperVII.KnillLaflamme.density_corrects_iff_operators`	119, support
`KomorebiPaperVII.KnillLaflamme.state_correctable_iff_gram`	139, Core 8.1
`KomorebiPaperVII.KnillLaflamme.amplifiedNoise_block`	157, support
`KomorebiPaperVII.KnillLaflamme.recovery_with_reference`	164, Core 8.1

10.22 SchurFeshbach: 4 theorem declarations

Source: `formalization/SchurFeshbach.lean`.

Named declaration	Source line
`KomorebiPaperVII.SchurFeshbach.resolvent_blocks`	20, support
`KomorebiPaperVII.SchurFeshbach.projected_resolvent`	27, Core 9.1
`KomorebiPaperVII.SchurFeshbach.schur_invertible`	46, Core 9.1
`KomorebiPaperVII.SchurFeshbach.eliminated_invertible_full_singular`	59, support

10.23 SeedOrbit: 34 theorem declarations

Source: `formalization/SeedOrbit.lean`.

Named declaration	Source line
`KomorebiPaperVII.SeedOrbit.mem_orbit_iff`	31, support
`KomorebiPaperVII.SeedOrbit.mem_orbit_iff_rows`	35, support
`KomorebiPaperVII.SeedOrbit.row_dimension`	54, support
`KomorebiPaperVII.SeedOrbit.orbit_dimension`	58, support
`KomorebiPaperVII.SeedOrbit.orbit_span`	63, support
`KomorebiPaperVII.SeedOrbit.span_orbit_dimension`	68, Core 2.1
`KomorebiPaperVII.SeedOrbit.cyclic_iff_full_rank`	73, support
`KomorebiPaperVII.SeedOrbit.cyclic_multiplicity_bound`	82, support
`KomorebiPaperVII.SeedOrbit.rowOrbit_mem`	93, support
`KomorebiPaperVII.SeedOrbit.rowOrbit_dimension`	112, support
`KomorebiPaperVII.SeedOrbit.rowOrbit_full`	117, support
`KomorebiPaperVII.SeedOrbit.stacked_action`	130, support
`KomorebiPaperVII.SeedOrbit.generates_iff_rowOrbit`	135, support

Named declaration	Source line
KomorebiPaperVII.SeedOrbit.generates_iff_rank	148, support
KomorebiPaperVII.SeedOrbit.seed_count_lower	152, support
KomorebiPaperVII.SeedOrbit.exists_full_column_rank	158, support
KomorebiPaperVII.SeedOrbit.exists_generating_seeds_iff	173, support
KomorebiPaperVII.SeedOrbit.block_generates_iff	195, support
KomorebiPaperVII.SeedOrbit.exists_block_seeds_iff	210, support
KomorebiPaperVII.SeedOrbit.seed_formula_le_iff	229, support
KomorebiPaperVII.SeedOrbit.quantum_seed_rank	239, Core 2.2
KomorebiPaperVII.SeedOrbit.local_tensor_action	261, support
KomorebiPaperVII.SeedOrbit.product_decomposition_rank_bound	272, support
KomorebiPaperVII.SeedOrbit.product_decomposition_at_rank	289, support
KomorebiPaperVII.SeedOrbit.product_length_minimum	298, support
KomorebiPaperVII.SeedOrbit.schmidt_rank_eq_matrix_rank	308, support
KomorebiPaperVII.SeedOrbit.local_orbit_is_tensor_span	319, support
KomorebiPaperVII.SeedOrbit.local_orbit_schmidt_dimension	328, support
KomorebiPaperVII.SeedOrbit.partial_trace_gram	336, support
KomorebiPaperVII.SeedOrbit.reduced_rank_eq_schmidt	343, support
KomorebiPaperVII.SeedOrbit.reduced_positive	348, support
KomorebiPaperVII.SeedOrbit.reduced_full_support_iff	352, support
KomorebiPaperVII.SeedOrbit.local_orbit_full_iff	357, support
KomorebiPaperVII.SeedOrbit.local_orbit_reduced_support	364, Core 2.3

10.24 SyndromeRecovery: 18 theorem declarations

Source: `formalization/SyndromeRecovery.lean`.

Named declaration	Source line
KomorebiPaperVII.Recovery.OrthogonalSyndromes.packet_selfAdjoint	37, support
KomorebiPaperVII.Recovery.OrthogonalSyndromes.packet_mul	41, support
KomorebiPaperVII.Recovery.OrthogonalSyndromes.total_projection	53, support
KomorebiPaperVII.Recovery.OrthogonalSyndromes.complement_projection	61, support
KomorebiPaperVII.Recovery.OrthogonalSyndromes.total_error	64, support
KomorebiPaperVII.Recovery.OrthogonalSyndromes.complement_error	68, support
KomorebiPaperVII.Recovery.OrthogonalSyndromes.error_adjoint_complement	71, support
KomorebiPaperVII.Recovery.OrthogonalSyndromes.recoveryKraus_complete	84, support
KomorebiPaperVII.Recovery.OrthogonalSyndromes.decoder_eq_kraus	92, support
KomorebiPaperVII.Recovery.OrthogonalSyndromes.decoder_add	97, support
KomorebiPaperVII.Recovery.OrthogonalSyndromes.decoder_cross_error	103, support
KomorebiPaperVII.Recovery.OrthogonalSyndromes.decoder_smul	129, support
KomorebiPaperVII.Recovery.OrthogonalSyndromes.decoder_weighted_noise	138, support
KomorebiPaperVII.Recovery.OrthogonalSyndromes.decoder_error_superposition	149, support
KomorebiPaperVII.Recovery.recovery_positive	180, support
KomorebiPaperVII.Recovery.recovery_trace_preserving	186, support

Named declaration	Source line
`KomorebiPaperVII.Recovery.recovery_complete_positivity`	197, support
`KomorebiPaperVII.Recovery.recovery_density_matrix`	212, support

10.25 TensorRefinement: 11 theorem declarations

Source: `formalization/TensorRefinement.lean`.

Named declaration	Source line
`KomorebiPaperVII.TensorRefinement.tensor_mul`	20, support
`KomorebiPaperVII.TensorRefinement.tensor_one`	30, support
`KomorebiPaperVII.TensorRefinement.tensor_sum`	42, support
`KomorebiPaperVII.TensorRefinement.tensor_adjoint`	54, support
`KomorebiPaperVII.TensorRefinement.fine_mul`	67, support
`KomorebiPaperVII.TensorRefinement.fine_complete`	86, support
`KomorebiPaperVII.TensorRefinement.fine_selfAdjoint`	94, support
`KomorebiPaperVII.TensorRefinement.coarse_mul`	108, support
`KomorebiPaperVII.TensorRefinement.coarse_complete`	126, support
`KomorebiPaperVII.TensorRefinement.coarse_selfAdjoint`	133, support
`KomorebiPaperVII.TensorRefinement.tensor_polyphase_refinement`	139, Core 3.2

11 Module dependencies, source identities, and pins

11.1 Current local module graph

The dependencies below are read from public imports and commands of the form import all. Duplicate imports count once. Importing all declarations matters in this Lean module edition because downstream proofs can require private implementation details. A module with no local import still imports the pinned Mathlib files named in its source.

Module	Direct current local imports
`CanonicalConnection`	Mathlib only
`ComputationalSeed`	Mathlib only
`ConstraintAudit`	Mathlib only
`CyclicProjectors`	Mathlib only
`ErrorSpanRecovery`	`KnillLaflammeNecessity`, `KnillLaflammeSufficiency`, `QuantumStateBridge`, `SyndromeRecovery`
`EuclideanTransfer`	Mathlib only
`GlobalProjectedConnection`	`GlobalRangeBundle`, `LocalProjectedCurvature`, `ProjectedCurvature`
`GlobalRangeBundle`	Mathlib only
`KnillLaflammeExamples`	`ErrorSpanRecovery`, `KnillLaflammeNecessity`, `KnillLaflammeSufficiency`, `QuantumStateBridge`
`KnillLaflammeNecessity`	`KnillLaflammeSufficiency`
`KnillLaflammeSufficiency`	`SyndromeRecovery`
`KrausChannels`	Mathlib only
`LeakageBound`	Mathlib only
`LocalProjectedCurvature`	`ProjectedCurvature`
`LowModeConvergence`	Mathlib only
`ManifoldProjectedCurvature`	`GlobalProjectedConnection`, `ProjectedCurvature`
`MovingFrames`	Mathlib only
`ProjectedBundleCompletion`	`ManifoldProjectedCurvature`
`ProjectedCurvature`	Mathlib only
`QuantumPacketCore`	Mathlib only
`QuantumStateBridge`	`KnillLaflammeNecessity`, `KnillLaflammeSufficiency`
`SchurFeshbach`	Mathlib only
`SeedOrbit`	Mathlib only
`SyndromeRecovery`	Mathlib only
`TensorRefinement`	Mathlib only

11.2 Current source SHA-256 identities

CanonicalConnection

`f2b433bdd432e18c9cb7af45403a1adcdd0426f5e9115d098317744e4b0fc3ed`

ComputationalSeed

`6b92535db98ff199d4b5fd3307c2c3935d5fe6c769761eae9fe4af71bb64c6c4`

ConstraintAudit

`e00fe71eb73e8ed8a652839d4f927e282784001d6c454f8cb2fe66f866de0371`

CyclicProjectors

`f0f88121acd4fc2f270de7e0ad7ca8d9e273945987a869935df670248e5d206b`

ErrorSpanRecovery

`f8a02ecda26d3a7d55378c06599597ada186b801750af09be0bbb2c3ae9191ed`

EuclideanTransfer

`5a259e9245571bf84048a3ff2f7d332cff29c8c2c9b3fcac515dceb43f594bad`

GlobalProjectedConnection

`ec618cd64d17d1bca4dee9ea0fae3c7f943c3347f4b1c8be063f3dd8517e0beb`

GlobalRangeBundle

`cbd9b611d6160a91a76332cad47d246f89cd82f02d38854089bf7f4eaa988b33`

KnillLaflammeExamples

`52f6ea5ccfc664ba41150c026015bec8a99d9eeb82880206b26456dee4abae05`

KnillLaflammeNecessity

`ff7b077fc8d55474c3a3b8ad6ae1017f4082e16dd9282ca07ecc5240f759c65d`

KnillLaflammeSufficiency

`3b71fbe77e55d8f603580678ada7ce4fd6303e8e1f1f5e028afb68c6bfc2e017`

KrausChannels

`915ebc9806b5f47ec8f958a72bfafe02723c5dcc47fceccdade1bcd3579e078f`

LeakageBound

`859f23f52f1274d485c2496828938cf48a8b5bf5a574264a7f04107b5b089887`

LocalProjectedCurvature

`6f59e61c3b60fca127a7cf9fdce369e26b421e2f20843c263201019cdf2af1c0`

LowModeConvergence

`7e077b24ccdfb970b349171d55551f39127a4556b2e2d247c47f6a4bc0774595`

ManifoldProjectedCurvature

`9803d99d975303179ccecbbfae1133bcab47496cee452cf3e130f3c4870a1e12`

MovingFrames

`8961ac05b66911ad82c84bc49ade3cac6eac2a84728898cc86a8286599f0ddef`

ProjectedBundleCompletion

`29afe46959d4538fc2833de8e014faf8be2203f811654796847d1b7597868d77`

ProjectedCurvature

`ad7b8cd20af860c3f455874c0009b3d18f40a5fe937531cd6798007406316686`

QuantumPacketCore

`92e49bb023877ed1f1c2bb34968aa81ee7724fa3a8e1801a831cfc70f1de3595`

QuantumStateBridge

`b3a76f530fb11075d5494f45a94d102c9084125e291c2bcd9ffa5210f6e8bdb2`

SchurFeshbach

`2ce60c9300b85ff06064d296cdf773f7cf9d2e0eafb7acbda15973fe851bb479`

SeedOrbit

`0b1da8c7e74ccc7bbca8e2b7c0c26bce01bc95d4089616ba95c1e4fe2f147773`

SyndromeRecovery

`23a176970c7adb9e995005d50722eebf953889147802f35ea9024b0081d6af84`

TensorRefinement

`b0587b32f55d50125790345695b777786d23e3f46141e7056049d3ef5a11e0f2`

11.3 Pinned compiler and dependency revisions

Compiler: `leanprover/lean4:v4.34.0-rc2`. Saved runtime commit: `6a10ac8c22beadecabdbb0919c2b50214762f91d`. The following are the nine packages in the manifest. Replay should use the locked revisions, including inherited dependencies.

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

12 Exact excerpt register

Each excerpt is a contiguous block of the unchanged source, retaining its full declaration and proof. The printed code uses explicit font fallback only where the monospaced font lacks a glyph. The corresponding `.lean.txt` file preserves raw UTF-8, without editorial substitutions. The source and excerpt hashes below bind the printed selection to its original context. The assumptions declared in a module remain in force when an excerpt is read. The teaching chapters state the relevant type classes and hypotheses.

row_equiv

Source: `formalization/SeedOrbit.lean`.

Lines 45 to 52. Declaration `orbitRowEquiv`.

Source SHA-256: `0b1da8c7e74ccc7bbca8e2b7c0c26bce01bc95d4089616ba95c1e4fe2f147773`.

Excerpt SHA-256: `a4d1d840bdcebbbffc42ae57380b479bc57d2afe07de79b1fe9e34bdda086c28`.

orbit_dimension

Source: `formalization/SeedOrbit.lean`.

Lines 68 to 71. Declaration `span_orbit_dimension`.

Source SHA-256: `0b1da8c7e74ccc7bbca8e2b7c0c26bce01bc95d4089616ba95c1e4fe2f147773`.

Excerpt SHA-256: `365ce586b2f58c4a3574bdc7af540baa1094dd028e25cd712a1334bb6782eb24`.

cyclic_rank

Source: `formalization/SeedOrbit.lean`.

Lines 73 to 80. Declaration `cyclic_iff_full_rank`.

Source SHA-256: `0b1da8c7e74ccc7bbca8e2b7c0c26bce01bc95d4089616ba95c1e4fe2f147773`.

Excerpt SHA-256: `f82dfbcc44084af989f282f53c4380f8b99bd21b2ced9722c556a65551ab7fdc`.

seed_generate

Source: `formalization/SeedOrbit.lean`.

Lines 127 to 128. Declaration `Generates`.

Source SHA-256: `0b1da8c7e74ccc7bbca8e2b7c0c26bce01bc95d4089616ba95c1e4fe2f147773`.

Excerpt SHA-256: `f5f2e4255c8ba1ace030b5315a2b6be0db3fc24fe4beda857449e94a5b41b6de`.

stack_rank

Source: `formalization/SeedOrbit.lean`.

Lines 148 to 150. Declaration `generates_iff_rank`.

Source SHA-256: `0b1da8c7e74ccc7bbca8e2b7c0c26bce01bc95d4089616ba95c1e4fe2f147773`.

Excerpt SHA-256: `f490fd86582b597c2a514ab1effe39cab73d59a96ca7a6b9923d52eb8c02702c`.

full_column

Source: `formalization/SeedOrbit.lean`.

Lines 158 to 171. Declaration `exists_full_column_rank`.

Source SHA-256: `0b1da8c7e74ccc7bbca8e2b7c0c26bce01bc95d4089616ba95c1e4fe2f147773`.

Excerpt SHA-256: `63d1eecf433f1ddc3dbfc779101bca490c43fcc6709bfa193f2c7e5ee31ee615`.

seed_minimum

Source: `formalization/SeedOrbit.lean`.

Lines 239 to 246. Declaration `quantum_seed_rank`.

Source SHA-256: `0b1da8c7e74ccc7bbca8e2b7c0c26bce01bc95d4089616ba95c1e4fe2f147773`.

Excerpt SHA-256: `89ee9cb7e0961e27954fe763e202dd0c3f44282f2b5fa3cc2abbf7bc71b316c5`.

schmidt_definition

Source: `formalization/SeedOrbit.lean`.

Lines 303 to 306. Declaration `schmidtRank`.

Source SHA-256: `0b1da8c7e74ccc7bbca8e2b7c0c26bce01bc95d4089616ba95c1e4fe2f147773`.

Excerpt SHA-256: `1279ef8ea57e64959c0d7d67193ccbb143617830b9007b93e5aae3a6de863f42`.

schmidt_minimum

Source: `formalization/SeedOrbit.lean`.

Lines 298 to 300. Declaration `product_length_minimum`.

Source SHA-256: `0b1da8c7e74ccc7bbca8e2b7c0c26bce01bc95d4089616ba95c1e4fe2f147773`.

Excerpt SHA-256: `435e38531c7e19906f0ce32c7a69346a9990726ce7d9e697e8b6443d9f95942c`.

local_action

Source: `formalization/SeedOrbit.lean`.

Lines 261 to 265. Declaration `local_tensor_action`.

Source SHA-256: `0b1da8c7e74ccc7bbca8e2b7c0c26bce01bc95d4089616ba95c1e4fe2f147773`.

Excerpt SHA-256: `8130668bea5e0e3ab38489f1a45a4a7b7d0593996ab788615ef7bbc36ed52ab8`.

reduced_support

Source: `formalization/SeedOrbit.lean`.

Lines 364 to 366. Declaration `local_orbit_reduced_support`.

Source SHA-256: `0b1da8c7e74ccc7bbca8e2b7c0c26bce01bc95d4089616ba95c1e4fe2f147773`.

Excerpt SHA-256: `147a14439c7983ad43f62238f014efeebb80b403a15160f3c8a64c23af1bdc6d`.

seed_basis

Source: `formalization/ComputationalSeed.lean`.

Lines 80 to 82. Declaration `circuit_seed_basis`.

Source SHA-256: `6b92535db98ff199d4b5fd3307c2c3935d5fe6c769761eae9fe4af71bb64c6c4`.

Excerpt SHA-256: `e228b9b252604336965a67975ccbebbe2c014dadc8da35b9e38f9457fad20162`.

seed_translation

Source: `formalization/ComputationalSeed.lean`.

Lines 72 to 77. Declaration `circuit_eq_translation`.

Source SHA-256: `6b92535db98ff199d4b5fd3307c2c3935d5fe6c769761eae9fe4af71bb64c6c4`.

Excerpt SHA-256: `42c2842952870bb25e00b99c1d316542ccb39edcc35b9bf5282899c130919b8c`.

cyclic_character

Source: `formalization/CyclicProjectors.lean`.

Lines 22 to 26. Declaration `character_sum`.

Source SHA-256: `f0f88121acd4fc2f270de7e0ad7ca8d9e273945987a869935df670248e5d206b`.

Excerpt SHA-256: `6182ca8cbfb31865af273e0554bd4027e66efedb0195c6e5548ff6041df9fbad`.

cyclic_mul

Source: `formalization/CyclicProjectors.lean`.

Lines 44 to 58. Declaration `packet_mul`.

Source SHA-256: `f0f88121acd4fc2f270de7e0ad7ca8d9e273945987a869935df670248e5d206b`.

Excerpt SHA-256: `5d5eb96a4f93624594bcc443e22a1fd3b50078f6219ca12787f9492f83e149f5`.

cyclic_matrix

Source: `formalization/CyclicProjectors.lean`.

Lines 157 to 170. Declaration `finite_cyclic_projectors`.

Source SHA-256: `f0f88121acd4fc2f270de7e0ad7ca8d9e273945987a869935df670248e5d206b`.

Excerpt SHA-256: `314f68975d510c4ac2dbf381e55d004877797f7e011199f51b0743c4231fe886`.

tensor_mul

Source: `formalization/TensorRefinement.lean`.

Lines 20 to 27. Declaration `tensor_mul`.

Source SHA-256: `b0587b32f55d50125790345695b777786d23e3f46141e7056049d3ef5a11e0f2`.

Excerpt SHA-256: `840d1c4953c639cdfb51054f3e6524dd7e5e58c9f56328e253b43677ef4950d9`.

tensor_refinement

Source: `formalization/TensorRefinement.lean`.

Lines 139 to 154. Declaration `tensor_polyphase_refinement`.

Source SHA-256: `b0587b32f55d50125790345695b777786d23e3f46141e7056049d3ef5a11e0f2`.

Excerpt SHA-256: `2f58f8936bffee8ed36834f4618bcb5de3ef8454cffcd992cf9bd226fe21bb07`.

packet_structure

Source: `formalization/QuantumPacketCore.lean`.

Lines 22 to 25. Declaration `PacketFamily`.

Source SHA-256: `92e49bb023877ed1f1c2bb34968aa81ee7724fa3a8e1801a831cfc70f1de3595`.

Excerpt SHA-256: `94b93d351a97d973bf990a4a8d1fcd4a8d04e12ae60192ab2eaf58fd05722bde`.

compose_blocks

Source: `formalization/QuantumPacketCore.lean`.

Lines 41 to 50. Declaration `compose_blocks`.

Source SHA-256: `92e49bb023877ed1f1c2bb34968aa81ee7724fa3a8e1801a831cfc70f1de3595`.

Excerpt SHA-256: `d62cbe9447e98700c0387ade81a7695945d771ac4aad6dc2e445fb925825dec1`.

circuit_path

Source: `formalization/QuantumPacketCore.lean`.

Lines 63 to 68. Declaration `circuit_path_expansion`.

Source SHA-256: `92e49bb023877ed1f1c2bb34968aa81ee7724fa3a8e1801a831cfc70f1de3595`.

Excerpt SHA-256: `8a7447d90ccb095429646dbfcb867b148a3807779708cc2e65b183c3691fc338`.

unitarity_columns

Source: `formalization/QuantumPacketCore.lean`.

Lines 119 to 131. Declaration `unitarity_columns`.

Source SHA-256: `92e49bb023877ed1f1c2bb34968aa81ee7724fa3a8e1801a831cfc70f1de3595`.

Excerpt SHA-256: `54be9086567756fa2f7a0f2136a49ed2b27eb5289098ad7171c6e4db4ca7b8ba`.

symmetry_exp

Source: `formalization/QuantumPacketCore.lean`.

Lines 168 to 171. Declaration `symmetry_exponential_no_transfer`.

Source SHA-256: `92e49bb023877ed1f1c2bb34968aa81ee7724fa3a8e1801a831cfc70f1de3595`.

Excerpt SHA-256: `39905e80eae8288643f906f8bce86131420f979f90a22972af30af14907e2fdb`.

measurement

Source: `formalization/QuantumPacketCore.lean`.

Lines 208 to 211. Declaration `measurement_invariance`.

Source SHA-256: `92e49bb023877ed1f1c2bb34968aa81ee7724fa3a8e1801a831cfc70f1de3595`.

Excerpt SHA-256: `d98f2c05feb20e8ec9edc53e090c36e404cc45977db5bf30b82abb62d4b2cad2`.

transport_derivative

Source: `formalization/LeakageBound.lean`.

Lines 31 to 38. Declaration `transported_derivative`.

Source SHA-256: `859f23f52f1274d485c2496828938cf48a8b5bf5a574264a7f04107b5b089887`.

Excerpt SHA-256: `4f6cf06917d31eef4a0cb83ee091863c72390050b998c3ec82f81ed4715b0536`.

transport_lipschitz

Source: `formalization/LeakageBound.lean`.

Lines 46 to 54. Declaration `transported_lipschitz`.

Source SHA-256: `859f23f52f1274d485c2496828938cf48a8b5bf5a574264a7f04107b5b089887`.

Excerpt SHA-256: `f7a3612f65196115dc1c6dc0b6721a1a057db6f39fedd361e43acbe872a31cf7`.

leakage_matrix

Source: `formalization/LeakageBound.lean`.

Lines 167 to 171. Declaration `matrix_commutator_leakage_bound`.

Source SHA-256: `859f23f52f1274d485c2496828938cf48a8b5bf5a574264a7f04107b5b089887`.

Excerpt SHA-256: `2da717dabf6f741c4b0d5d435b27898cd4360e6a75bd401e9b9ca08bbea787b1`.

leakage_perturbation

Source: `formalization/LeakageBound.lean`.

Lines 151 to 164. Declaration `perturbation_leakage_bound`.

Source SHA-256: `859f23f52f1274d485c2496828938cf48a8b5bf5a574264a7f04107b5b089887`.

Excerpt SHA-256: `6c29b54826e1a17a70cc88a993747995a66d98bc4de068407fe606ab14f39e84`.

frame_product

Source: `formalization/MovingFrames.lean`.

Lines 23 to 29. Declaration `matrix_vector_derivative`.

Source SHA-256: `8961ac05b66911ad82c84bc49ade3cac6eac2a84728898cc86a8286599f0ddef`.

Excerpt SHA-256: `b541e7998198257db7256977da20097d88c1093f7a302793f133b501727fb886`.

moving_coordinates

Source: `formalization/MovingFrames.lean`.

Lines 110 to 132. Declaration `moving_frame_coordinates`.

Source SHA-256: `8961ac05b66911ad82c84bc49ade3cac6eac2a84728898cc86a8286599f0ddef`.

Excerpt SHA-256: `a54d0ad271dcb2a4a32e0800cb13e680e61dc3f6fa699184ba31ae9c5d81b095`.

connection_jet

Source: `formalization/CanonicalConnection.lean`.

Lines 78 to 95. Declaration `differentiated_orthogonality`.

Source SHA-256: `f2b433bdd432e18c9cb7af45403a1adcdd0426f5e9115d098317744e4b0fc3ed`.

Excerpt SHA-256: `eb57f9d5f332db8c25c84c3e5a6bb54c1cd10454626ef4f3d200a3c3878d1c2e`.

canonical_connection

Source: `formalization/CanonicalConnection.lean`.

Lines 113 to 131. Declaration `canonical_connection_of_hasDerivAt`.

Source SHA-256: `f2b433bdd432e18c9cb7af45403a1adcdd0426f5e9115d098317744e4b0fc3ed`.

Excerpt SHA-256: `ee12fd06d9eaa15681457221c5d4aca3d1aaff2b2487d0b6389cedebd82bc40e`.

sandwich_zero

Source: `formalization/ProjectedCurvature.lean`.

Lines 22 to 26. Declaration `jet_sandwich_zero`.

Source SHA-256: `ad7b8cd20af860c3f455874c0009b3d18f40a5fe937531cd6798007406316686`.

Excerpt SHA-256: `7dc40b766cb348d7f859e22f3264fe7832a0243c2573b3c72b2e0670f962ae77`.

local_curvature

Source: `formalization/LocalProjectedCurvature.lean`.

Lines 79 to 94. Declaration `projected_connection_curvature`.

Source SHA-256: `6f59e61c3b60fca127a7cf9fdce369e26b421e2f20843c263201019cdf2af1c0`.

Excerpt SHA-256: `536eabe9e44ef8fea3518c58157a34c7e2485e4bd4cd031aecced8c419426013`.

range_transport

Source: `formalization/GlobalRangeBundle.lean`.

Lines 43 to 43. Declaration `transport`.

Source SHA-256: `cbd9b611d6160a91a76332cad47d246f89cd82f02d38854089bf7f4eaa988b33`.

Excerpt SHA-256: `dfd3703c4861cefd45738415feb43fd6d139ae903e22fcfa54a52f3ae3166b59`.

range_intertwines

Source: `formalization/GlobalRangeBundle.lean`.

Lines 49 to 54. Declaration `transport_intertwines`.

Source SHA-256: `cbd9b611d6160a91a76332cad47d246f89cd82f02d38854089bf7f4eaa988b33`.

Excerpt SHA-256: `270a82bb94c15f6815e19aacca566c805b4c02366ca5aa7f0af3a21edeff6ea7`.

range_product

Source: `formalization/GlobalRangeBundle.lean`.

Lines 180 to 209. Declaration `localProduct`.

Source SHA-256: `cbd9b611d6160a91a76332cad47d246f89cd82f02d38854089bf7f4eaa988b33`.

Excerpt SHA-256: `1f6345cfbb46c8c9cfd05c107d4c065ff4203c2f6579ab9ad3f77f677446c0a7`.

range_atlas

Source: `formalization/GlobalRangeBundle.lean`.

Lines 241 to 252. Declaration `smooth_range_bundle_atlas`.

Source SHA-256: `cbd9b611d6160a91a76332cad47d246f89cd82f02d38854089bf7f4eaa988b33`.

Excerpt SHA-256: `8e70b5873ddcf4e12a7c8f010839e666e3eed7810e9e45ac18fe8694cf61d9fd`.

global_nabla

Source: `formalization/GlobalProjectedConnection.lean`.

Lines 29 to 30. Declaration `nabla`.

Source SHA-256: `ec618cd64d17d1bca4dee9ea0fae3c7f943c3347f4b1c8be063f3dd8517e0beb`.

Excerpt SHA-256: `21fd6220b421762dd36a043194975b4cb173eb8ecba9904f1e82558148f56361`.

global_leibniz

Source: `formalization/GlobalProjectedConnection.lean`.

Lines 59 to 63. Declaration `nabla_leibniz`.

Source SHA-256: `ec618cd64d17d1bca4dee9ea0fae3c7f943c3347f4b1c8be063f3dd8517e0beb`.

Excerpt SHA-256: `7c4fed39fa233c060d4c3cb834aec746aa20118fac168fe896e2b97d1b71e20e`.

global_glue

Source: `formalization/GlobalProjectedConnection.lean`.

Lines 115 to 124. Declaration `local_connections_glue`.

Source SHA-256: `ec618cd64d17d1bca4dee9ea0fae3c7f943c3347f4b1c8be063f3dd8517e0beb`.

Excerpt SHA-256: `44e2c6deb0f447d93e739a761bebd05a7202663add66936ed4ca8b04f8140f56`.

manifold_curvature

Source: `formalization/ManifoldProjectedCurvature.lean`.

Lines 214 to 232. Declaration `projected_connection_curvature`.

Source SHA-256: `9803d99d975303179ccecbbfae1133bcab47496cee452cf3e130f3c4870a1e12`.

Excerpt SHA-256: `b16a27faaca85a392b119d8a58b11243610f490f85808d3f503f924be9d4b759`.

smooth_section

Source: `formalization/ProjectedBundleCompletion.lean`.

Lines 40 to 53. Declaration `smooth_section_iff`.

Source SHA-256: `29afe46959d4538fc2833de8e014faf8be2203f811654796847d1b7597868d77`.

Excerpt SHA-256: `4d880ef0afe4f2904521de95d6987431163c41a619192455f0d4963381a109ab`.

gauge_law

Source: `formalization/ProjectedBundleCompletion.lean`.

Lines 57 to 70. Declaration `local_connections_gauge`.

Source SHA-256: `29afe46959d4538fc2833de8e014faf8be2203f811654796847d1b7597868d77`.

Excerpt SHA-256: `3515991be601fc80774ba245cf2fd97b415f112cf14f1303201a2f488392eca3`.

kraus_amplify

Source: `formalization/KrausChannels.lean`.

Lines 48 to 56. Declaration `amplification_eq_kraus`.

Source SHA-256: `915ebc9806b5f47ec8f958a72bfafe02723c5dcc47fceccdade1bcd3579e078f`.

Excerpt SHA-256: `1323a35ef1a6247fba9aa829edaebc571ab8045b4e0362a24ca12abd66386fa6`.

kraus_cp

Source: `formalization/KrausChannels.lean`.

Lines 58 to 62. Declaration `channel_completely_positive`.

Source SHA-256: `915ebc9806b5f47ec8f958a72bfafe02723c5dcc47fceccdade1bcd3579e078f`.

Excerpt SHA-256: `aecdfd6e1b918aac49f394badbf080146717c04ebf3d94a03f06528db544435e`.

syndrome_structure

Source: `formalization/SyndromeRecovery.lean`.

Lines 21 to 27. Declaration `OrthogonalSyndromes`.

Source SHA-256: `23a176970c7adb9e995005d50722eebf953889147802f35ea9024b0081d6af84`.

Excerpt SHA-256: `73f2e29a5c2b47648bf6729fa98b330f3532a0f288038c89b1388338a3042e0d`.

syndrome_complete

Source: `formalization/SyndromeRecovery.lean`.

Lines 84 to 90. Declaration `recoveryKraus_complete`.

Source SHA-256: `23a176970c7adb9e995005d50722eebf953889147802f35ea9024b0081d6af84`.

Excerpt SHA-256: `ee0a642c16544ee89b34db86766f682bfc4789777d83f2b7d51b749b359521e7`.

syndrome_cross

Source: `formalization/SyndromeRecovery.lean`.

Lines 103 to 124. Declaration `decoder_cross_error`.

Source SHA-256: `23a176970c7adb9e995005d50722eebf953889147802f35ea9024b0081d6af84`.

Excerpt SHA-256: `c5524fa7c2b23fb404c2cc2b7c66c4d1be519b0fc2c8189e0b6cd5cbb0aacb15`.

kl_data

Source: `formalization/KnillLaflammeSufficiency.lean`.

Lines 187 to 193. Declaration `Data`.

Source SHA-256: `3b71fbe77e55d8f603580678ada7ce4fd6303e8e1f1f5e028afb68c6bfc2e017`.

Excerpt SHA-256: `ee0fe883d0c774bf87c6bb168c1419d60e29142fc7711f195c500c9fa12187db`.

kl_diagonal

Source: `formalization/KnillLaflammeSufficiency.lean`.

Lines 204 to 214. Declaration `diagonal`.

Source SHA-256: `3b71fbe77e55d8f603580678ada7ce4fd6303e8e1f1f5e028afb68c6bfc2e017`.

Excerpt SHA-256: `db1af099e2586af393a0b1574fb59cc0f8464f63a93001e6a52e70d3e62db4a8`.

kl_null

Source: `formalization/KnillLaflammeSufficiency.lean`.

Lines 107 to 110. Declaration `zero_weight_error`.

Source SHA-256: `3b71fbe77e55d8f603580678ada7ce4fd6303e8e1f1f5e028afb68c6bfc2e017`.

Excerpt SHA-256: `062b1f1c10c1db14ed3d3fa301585f6746b2a816af2a875256e671740ef79ddc`.

kl_corrects

Source: `formalization/KnillLaflammeSufficiency.lean`.

Lines 252 to 259. Declaration `recovery_corrects`.

Source SHA-256: `3b71fbe77e55d8f603580678ada7ce4fd6303e8e1f1f5e028afb68c6bfc2e017`.

Excerpt SHA-256: `0edf5306e8febc4c8d0dd4bf808c41f3498f9c568687a547926941d315f33bf5`.

kl_rigidity

Source: `formalization/KnillLaflammeNecessity.lean`.

Lines 19 to 45. Declaration `rank_one_gram_rigidity`.

Source SHA-256: `ff7b077fc8d55474c3a3b8ad6ae1017f4082e16dd9282ca07ecc5240f759c65d`.

Excerpt SHA-256: `c7b1e822236c6453ea334c28938823c2af3318444fef3cf82ee5311d7310c0da`.

kl_necessity

Source: `formalization/KnillLaflammeNecessity.lean`.

Lines 86 to 105. Declaration `gram_of_recovery`.

Source SHA-256: `ff7b077fc8d55474c3a3b8ad6ae1017f4082e16dd9282ca07ecc5240f759c65d`.

Excerpt SHA-256: `b3955cdc9224b5c2af85de359794d01818749364c809e7ce3d991b9c109ad40f`.

density_bridge

Source: `formalization/QuantumStateBridge.lean`.

Lines 119 to 129. Declaration `density_corrects_iff_operators`.

Source SHA-256: `b3a76f530fb11075d5494f45a94d102c9084125e291c2bcd9ffa5210f6e8bdb2`.

Excerpt SHA-256: `ae811fac94c396d9ee2052588ea8668a2e69f5d24c3bcdd26560c4567b6fe95f`.

kl_criterion

Source: `formalization/QuantumStateBridge.lean`.

Lines 139 to 151. Declaration `state_correctable_iff_gram`.

Source SHA-256: `b3a76f530fb11075d5494f45a94d102c9084125e291c2bcd9ffa5210f6e8bdb2`.

Excerpt SHA-256: `3e7dd0d4356f6ce599db17a3d9a4049a86fb612e0bb8254114f6f5e15df70dcb`.

kl_reference

Source: `formalization/QuantumStateBridge.lean`.

Lines 164 to 182. Declaration `recovery_with_reference`.

Source SHA-256: `b3a76f530fb11075d5494f45a94d102c9084125e291c2bcd9ffa5210f6e8bdb2`.

Excerpt SHA-256: `e6301010a1248fffde60cbd86f25278f5615775f2ef4fc02bb5a57276b510366`.

kl_span

Source: `formalization/ErrorSpanRecovery.lean`.

Lines 138 to 165. Declaration `recovery_span_channel`.

Source SHA-256: `f8a02ecda26d3a7d55378c06599597ada186b801750af09be0bbb2c3ae9191ed`.

Excerpt SHA-256: `108c56b74be5789d672f77a69cf5d11fac6376476eda291ca96358ad2f9c5aba`.

kl_family

Source: `formalization/ErrorSpanRecovery.lean`.

Lines 188 to 209. Declaration `error_family_correctable_iff_gram`.

Source SHA-256: `f8a02ecda26d3a7d55378c06599597ada186b801750af09be0bbb2c3ae9191ed`.

Excerpt SHA-256: `5d027f08e3b3e93ee21eda17960f631aadd79cae3e6d0e47df52bdced92e4df7`.

feshbach

Source: `formalization/SchurFeshbach.lean`.

Lines 27 to 44. Declaration `projected_resolvent`.

Source SHA-256: `2ce60c9300b85ff06064d296cdf773f7cf9d2e0eafb7acbda15973fe851bb479`.

Excerpt SHA-256: `d84ccc733fea567da198271006256848b4afa3112df09490598de9968c00ca70`.

feshbach_counterexample

Source: `formalization/SchurFeshbach.lean`.

Lines 59 to 64. Declaration `eliminated_invertible_full_singular`.

Source SHA-256: `2ce60c9300b85ff06064d296cdf773f7cf9d2e0eafb7acbda15973fe851bb479`.

Excerpt SHA-256: `377aa16415eeb74cce76d96a5bc1e33ac627d32dbccdcb9465f52bb456552283`.

euclidean_cfc

Source: `formalization/EuclideanTransfer.lean`.

Lines 39 to 47. Declaration `transfer_cfc`.

Source SHA-256: `5a259e9245571bf84048a3ff2f7d332cff29c8c2c9b3fcac515dceb43f594bad`.

Excerpt SHA-256: `9eb0a3b8ac13e09838ab8074571b5149b2040c810140f3518039b8edb8915e61`.

euclidean_endpoint

Source: `formalization/EuclideanTransfer.lean`.

Lines 89 to 96. Declaration `finite_euclidean_transfer`.

Source SHA-256: `5a259e9245571bf84048a3ff2f7d332cff29c8c2c9b3fcac515dceb43f594bad`.

Excerpt SHA-256: `ee9c869877cc94941a4308d727f4d422bb8d5c13864aa4c33755d0f5a1ef3d8a`.

low_mode

Source: `formalization/LowModeConvergence.lean`.

Lines 32 to 45. Declaration `low_mode_error`.

Source SHA-256: `7e077b24ccdfb970b349171d55551f39127a4556b2e2d247c47f6a4bc0774595`.

Excerpt SHA-256: `7de00df6c8e15233a57e9432f25072c725e03efb58d3b6e0cfa6a72d61a0b97c`.

constraint_kernel

Source: `formalization/ConstraintAudit.lean`.

Lines 31 to 34. Declaration `compressed_anomaly_zero`.

Source SHA-256: `e00fe71eb73e8ed8a652839d4f927e282784001d6c454f8cb2fe66f866de0371`.

Excerpt SHA-256: `121148fa69ae880879db5578d139bcff070684810952dc73d45f9fe7106ce78d`.

constraint_nonclosure

Source: `formalization/ConstraintAudit.lean`.

Lines 56 to 60. Declaration `commutator_nonzero`.

Source SHA-256: `e00fe71eb73e8ed8a652839d4f927e282784001d6c454f8cb2fe66f866de0371`.

Excerpt SHA-256: `2a18c6988e67ff03d9308f041e90dc1e3713f983dc36f07f9b1f28f91454d3bd`.