OPERATION KOMOREBI

REALIZATION CALCULUS



CORE SERIES X

Universal Residual Seeds

Deformation forests, transport categories, and conditional system completion

---

Lean and evidence guide

Komorebi P.

Verification edition / 20 October 2026

REALIZATION CALCULUS

Abstract

This guide explains the source contracts behind Paper X rather than treating its terminal signature as a stand-in for the mathematics. It provides a concordance of the twenty source records, follows the two independent construction branches, and distinguishes arithmetic occurrence, finite trace recovery, carrier presentations, local comparison data, and nonzero occupancy. Four complete production modules are printed from preserved source. Their saved verification and the separate October 20 production replay remain explicit.

Contents

1 The October 20 production verification **3**

2 How to read the formal evidence **4**

2.1 Three different records 4

2.2 What each kind of check can establish 4

3 The twenty source records **5**

4 Residual roots and reconstruction **7**

4.1 A point in an inverse limit is a coherent family 7

4.2 The corrected digit formula 7

5 Counting finite points without inventing limit points **8**

5.1 The singular count is not a count of infinite branches 8

6 Cohomology packets and directed transport **9**

6.1 A transport graph is data before it is an optimization problem 9

6.2 Transport must respect every reduction square 9

7 Horizontal completion, hypothesis by hypothesis **10**

7.1 The local profile is not hidden in the word reconstruction 10

7.2 What changes when coefficient places are indexed 10

8 From arithmetic occurrence to a finite separator **11**

8.1 What the separator accepts 11

8.2 The finite character-table model 11

8.3 Exact recovery can still lose finite precision 11

9 Carrier recovery is a construction inside supplied data **12**

9.1 Three tests after solving a linear equation 12

9.2 What happens to local data 12

10 Occupancy and the missing nonzero premise **13**

10.1 Why the terminal assembly permits an empty selection 13

11 The final conjunction and its exact logical strength **14**

11.1 Read the six proof lines 14

11.2 A practical application contract 14

12 Source provenance and replay boundaries **15**

12.1 The compact preservation archive 15

12.2 Interpreting the saved controls 15

13 The recorded endpoint environment **16**

14 The complete terminal module **17**

15 Direct dependency: horizontal transport **22**

16 Direct dependency: constructed arithmetic separation **28**

17 Direct dependency: strict compatible carrier closure **29**

18 Companion records and limits **36**

1 The October 20 production verification

The selected source is the recovered October 10 archive, with SHA-256 `29e504f3508d54e72bd891bdd77c8df8ded52448b9f5aa161c1ebae1716ce58d`. It is an explicitly selected baseline for this new run. The October 16–17 packages were not recovered; their reported 122-module, 39-test run remains unauthenticated. `verification/2026-10-20/SOURCE_SELECTION.json` records this distinction.

Two fresh production builds. The twenty reading routes and printed endpoint resolve to an exact closure of 122 project and retained support modules. Both clean builds passed, including a replay from a relocated source directory. All 392 generated artifacts matched byte for byte. No inherited project proof objects were seeded. The terminal-only closure has 97 modules and is a subset of the 122. The pinned Lean 4.34.0-rc2 executable and nine dependency identities were checked. External Mathlib caches were reused; this is not a cold dependency rebuild. The portable runtime's executable-path adapter and its exact hashes are recorded.

Owned declarations and explicit targets. The fresh audit checked all 2,846 declarations owned by the 122 selected modules, including 2,127 theorems, 444 private declarations, and generated declarations. All 45 explicit target contracts resolved with the required kind and owner. Their complete elaborated types are retained. The union of transitive axioms was exactly `propext`, `Classical.choice`, and `Quot.sound`. No forbidden dependency was found. The unchanged checker first hit its 4 GB memory cap; the successful serial run raised the cap to 6 GB. Failed attempts remain recorded. The 18 compiler fixtures and 64 unit tests check tooling and are separate from these production counts.

Correspondence requires mathematical reading. The twenty-row register is a reading concordance. Its three incorrect source routes have been corrected to the actual residual-obstruction, reconstruction, and empty-level declarations. The independent bounded source review checked the amended terminal composition and finite-copy carrier closure under their explicit inputs. It is not a new proof audit of every historical theorem. Classical Inputs 3.1 and 4.1 and the conjectures remain outside the twenty-row concordance. Compiler acceptance and name resolution alone do not prove manuscript equivalence.

Finite checks have separate bounds. The current replay passed the original 28 companion tests and 13 reader regression methods, including 3,123 canonical residue towers through depth four over bases 2, 3, 5 and 7. The visual suite passed 263 positive assertions and rejected 14 false controls. Six further finite suites retain their actual bounds, seeds and logs separately. These counts are not combined into a theorem count.

The mathematical boundary is unchanged. The transported target family and recovered carrier summand are independent existential conclusions. Their identification requires another comparison. The literal constructed arithmetic separator is over `GQ`. Local comparisons, carrier presentations and source-family data remain inputs. Zero multiplicities are admitted; nonzero occupancy needs its own premise. Universal occupancy and natural finite seed rank remain conjectures. The later stronger residual-system attempt remains separately preserved and unpromoted.

2 How to read the formal evidence

Paper X has a finite and formal core, a substantial arithmetic development, and a terminal conditional assembly. These are not three names for one assertion. The source register below gives the twenty numbered proof records in the main manuscript. The checked endpoint concerns their recorded amended scope. Later arithmetic prose can still require hypotheses which an application has not supplied.

A Lean declaration proves its type, including implicit parameters and typeclass assumptions. A short proof can therefore depend on a long contract. Here the terminal proof is only six lines: it constructs a conjunction and invokes two previous theorems. The mathematical content is distributed across those theorems and their dependencies. Reading only the final six lines would miss the source family, all-depth naturality, inverse-limit closure, carrier presentations, and constituent local comparisons.

2.1 Three different records

Record	Interpretation
Block 27 and scalar/Mellin	Historical 17/20, with Claims 9.1, 9.2, and 11.1 still partial in those source revisions.
Conditional assembly	Recorded 20/20 under an explicitly amended contract. The target family and carrier summand are independent conclusions.
Later residual attempt	19/20 and an unpromoted terminal claim under a stronger contract. Its unavailable pinned compiler is not a rejection of the earlier checked module.

The corresponding records are preserved independently. This edition does not replace the later failed attempt by the earlier successful theorem, and it does not demote the successful theorem merely because the later attempt was incomplete.

2.2 What each kind of check can establish

An exact Python calculation verifies the displayed finite instance or enumerated range. A source hash verifies identity of bytes. A successful typeset build verifies that the document can be generated. A saved compiler log reports a past compilation. None of these is a fresh proof of every theorem in the collection. A transitive axiom report also has a specific declaration set and environment. Its scope is not enlarged by binding it into a longer monograph.

The present edition performs a new reader and release pass. It preserves the formal sources and distinguishes inherited compiler evidence from checks actually executed for this delivery. The dated final review and machine-readable verification record list the execution results. They are the authoritative place for current run counts, rather than numbers copied from an earlier checkpoint.

3 The twenty source records

The numbered entries below are the main manuscript's twenty theorem, proposition, and corollary records. Classical Inputs 3.1 and 4.1 are not counted as new proof records. Definitions, examples, numerical certificates, and the conjectural finite-seed-rank statement are also excluded from this count.

This is a reading concordance, not a claim that twenty lines of a table replace twenty theorem-to-source audits. The earlier seventeen entries retain their hypotheses. The final three arithmetic entries must be read through the amended source contract explained in Sections 7–11. Links target named destinations in the companion monograph. Keep the three PDFs in the same directory for relative file links to resolve.

Record	Statement and source route	Hypothesis to retain
2.1	Residual partition. `Residual`	A supplied, single-valued reduction map.
2.2	No crossing under positive-depth perturbation. `Residual`, `SeedConsequences`	Root-preserving operations. The infinite statement assumes convergence.
2.3	Zeroth-order obstruction. `Residual.zeroth_order_obstruction`	Different roots cannot be reached by genuine positive-depth operations.
3.2	Formal inverse-limit reconstruction. `Reconstruction.represented_local_algebra_reconstruction`	Represented complete local problems with compatible finite maps.
3.3	Finite-branch compactness. `Compactness`, `RepresentationForests`	Finite nonempty levels and the stated total reduction maps.
3.4	Smooth formal branch count. `SmoothMaps`, `IntegralCharts`	An actual power-series chart and coefficient-preserving local maps.
4.2	Cohomology packet decomposition. `ComplexCohomology`, `PacketCohomology`	A finite complete orthogonal family of chain projectors.
5.3	Weighted smooth-chart count. `Counting`, `FormalCharts`	Independent coordinates with the stated valuation lower bounds.
6.2	Smooth seed zeta function. `Zeta`, `SmoothModels`	The smooth-chart count, as an infinite formal power series.
6.3	Weighted seed zeta function. `Zeta`	The stated ordered positive weights, with finite initial levels separated.
6.4	Singular power seed. `Singular`, `SingularZeta`	The scalar equation x to the e equals zero at finite precision.
6.6	Fractional singular dimension. `BranchingDimension`, `Singular`	A finite-count growth exponent, not the dimension of a limit space.
7.3	Directed minimum-seed theorem. `DirectedSeeds`	A finite supplied graph. Arithmetic admissibility of edges is separate.
8.2	Adelic finite-depth completion. `ClassCommonSpine`, `SystemCompletion`	Common data, finite-depth nonemptiness, and inverse-limit-closed conditions.
8.3	Residual companion obstruction. `SystemCompletion.system_empty_level_obstruction`	An empty required finite level rules out the specified realization.
9.1	Finite Frobenius separation. `Claim91ConstructedClosure`	Later constructed rational-base separator, with its literal compatible-family inputs.

Record	Statement and source route	Hypothesis to retain
9.2	Compatible-carrier summand completion. `StrictCompatibleCarrierCompletion`	Explicit copy presentations, natural multiplicities within capacity, and supplied local data.
10.1	Exact finite occupancy trace. `SpectralProjectors`, `JointOccupancy`	Commuting split semisimple operators. The positive-trace interpretation uses characteristic zero.
11.1	Residual-system completion criterion. `ConditionalOccupiedSystemCompletion`	The later amended two-conjunct contract, not universal residual existence.
11.2	Exact location of the remaining existence problem. `SpineExistenceCriterion`, `RequiredPacketOccupancy`	All required finite levels and all supplied packet occupancy premises.

The monograph retains the general-base mathematical discussion. In contrast, the printed constructed arithmetic separator and the terminal assembly use `GQ`, the rational absolute Galois group. That literal base is part of the formal statement. An arbitrary-number-field version requires its own attachment and proof, not a change in a heading.

4 Residual roots and reconstruction

The fiber decomposition begins with a function, the reduction map. For an object x in a coefficient algebra A, its residual root is $\mathrm{red}_A(x)$. Distinct fibers are disjoint because a function has one value at each input. No topological or representation-theoretic theorem is needed for that partition.

The no-crossing theorem adds a hypothesis about the allowed operation. A positive-depth correction is trivial after residual reduction. A finite sequence of such corrections therefore stays in one fiber. An infinite sequence needs a convergence assumption. The sequence $0, 3, 0, 3, \ldots$ has one residue modulo three but does not converge modulo nine. Residual invariance is weaker than being Cauchy.

4.1 A point in an inverse limit is a coherent family

For a represented complete deformation problem, compatible finite maps determine one map into the inverse-limit coefficient ring. The argument does not assert that every finite object can be extended. Nor does it produce a map when the finite levels are empty. Reconstruction and existence remain distinct.

The finite-branch argument supplies existence when all levels are finite and nonempty and the transition maps are the stated reductions. Surjectivity of every transition map is unnecessary. Some nodes may die. The surviving branch is obtained from nodes with descendants at arbitrarily large depths. Finiteness prevents indefinitely moving the root to a new unrelated component.

Atlas V9 displays the infinite-level countermodel $X_m = \{n \geq m\}$ with inclusion maps. Each level is nonempty, but a compatible family would have to be one integer n lying in every X_m. No such integer exists. Its infinite levels violate the compactness hypotheses. This is a countermodel to dropping finiteness, not to the finite-level theorem.

4.2 The corrected digit formula

For compatible canonical residues $0 \leq x_m < p^m$, the digit in position $m-1$ is

$$a_{m-1} = \left\lfloor \frac{x_m}{p^{m-1}} \right\rfloor \pmod{p}.$$

Set $x_0 = 0$. Compatibility also makes

$$a_{m-1} = \frac{x_m - x_{m-1}}{p^{m-1}} \in \{0, \ldots, p-1\}.$$

The earlier algorithm display omitted the floor. The executable implementation already used integer division. For the base-five residues $4, 19$, the new digit is 3. Dividing 19 by 5 in the rationals does not recover a digit. The new regression checks compare the displayed formulas with the inherited executable on exhaustive bounded towers, including noncanonical integer representatives for the floor formula.

5 Counting finite points without inventing limit points

In a smooth chart $\mathcal{O}[[t_1, \dots, t_d]]$, local maps into $\mathcal{O}/\varpi^m$ send each parameter into the maximal ideal. There are q^{m-1} possibilities for each independent coordinate, giving $q^{d(m-1)}$ maps. The word independent is justified by the power-series presentation, not by the appearance of a grid in a figure.

Weighted coordinates with $t_i \in \varpi^{w_i}\mathcal{O}$ have $\max(m - w_i, 0)$ free residue digits. Thus

$$N_{\mathbf{w}}(m) = q^{\sum_i \max(m-w_i,0)}.$$

For weights $(1, 2, 4)$ over $\mathbb{Z}_2$, the counts begin $1, 2, 8, 32, 256, 2048$. The coordinate of weight four is not nontrivial at depth four. It first has a free digit at depth five. The weight and first visible depth are different quantities.

5.1 The singular count is not a count of infinite branches

For $x^e = 0$ modulo ϖ^m, the condition is $e\, v_\varpi(x) \geq m$. Its finite cardinality is $q^{m-\lceil m/e \rceil}$. Splitting $m = ek + r$, $1 \leq r \leq e$, gives

$$Z_e(T) = \frac{\sum_{r=1}^{e} q^{r-1}T^r}{1 - q^{e-1}T^e}.$$

This is an identity of formal power series. Its growth exponent $1 - 1/e$ measures the finite thickenings. It does not imply that the complete solution space has a fractional geometric dimension. In the integral domain $\mathcal{O}$, the only complete root of $x^e = 0$ is zero. Equivalently, the reduction at depth m of a root at depth em is zero. Every coherent infinite branch is therefore the zero branch.

The smooth, weighted, and singular cases teach different tests. A complete chart can have many genuine branches. A weighted chart delays their finite visibility. A singular thickening can have many finite roots and only one complete branch. The generating series must always be interpreted for the object it actually counts.

6 Cohomology packets and directed transport

A degreewise idempotent is not automatically a map of complexes. The required identity is $d_r P_j^r = P_j^{r+1} d_r$. Together with orthogonality and a sum equal to the identity, this makes the images into subcomplexes. The induced maps on cycles and boundaries then descend to cohomology. Their inverse identities give

$$H^r(C^\bullet) \simeq \bigoplus_j H^r(P_j C^\bullet).$$

The comparison with tangent or obstruction spaces of a particular deformation problem is an additional identification. The packet theorem itself is a theorem about a complex with supplied chain projectors.

Atlas V10 and V11 use a complex over $\mathbb{F}_5$ with dimensions $(2, 4, 2)$. The valid packet family gives cohomology dimensions $(1, 2, 1)$. A modified idempotent deletes the image of a retained source vector and fails the chain-map identity. All its individual matrices remain idempotent. The missing condition is compatibility with the differential, not idempotence.

6.1 A transport graph is data before it is an optimization problem

For a finite directed graph, contract strongly connected components. The resulting graph is acyclic. Every generating seed set must meet each source component, since no other component can reach into that source. Conversely, one vertex from every source reaches all components. This proves the directed minimum-seed formula.

The theorem optimizes a supplied graph. An arithmetic twist, induction, or coefficient change is an edge only after its admissibility has been established. Adding an unjustified arrow can lower the computed seed rank without proving anything about the original arithmetic problem. Likewise, a finite graph theorem does not settle the conjectural finite seed rank of an unrestricted arithmetic transport category.

6.2 Transport must respect every reduction square

For roots of $x^2 + 1$ modulo powers of five, negation preserves the equation and commutes with every reduction. Translation by one maps the root two to the root three modulo five, but maps three to four, which is not a root. One successful pair of endpoints does not define a natural operation on the whole tower. The formal interface in the next section expresses exactly this distinction.

7 Horizontal completion, hypothesis by hypothesis

The module `HorizontalTransportCompletion` works with actual change-of-basis classes. Its structure `AdmissibleClassTransport` has three fields: a map at every precision, preservation of allowed classes, and compatibility with reduction. Those fields are the data of a map of inverse towers.

A completed source representation ρ first gives a coherent branch of its finite classes. Applying the transport at each depth gives a coherent target branch. The predecessor reconstruction theorem then supplies an actual target representation. This is stronger than mapping one residual root, but it is still conditional on the existence of the completed source representation.

7.1 The local profile is not hidden in the word reconstruction

`TransportedAlong` records, for every depth, equality between the reduced target class and the transported source class. An arbitrary semantic property of the complete representation does not follow from that equality alone. The theorem therefore accepts `InverseLimitClosed` for the selected target profile and finite-level conditions. Applying the closure theorem to the reconstructed target establishes that profile.

For example, nontriviality is not a closed condition under the relevant limit. The matrices

$$\rho_n(g) = \begin{pmatrix} 1 & 3^n g \\ 0 & 1 \end{pmatrix}$$

are nontrivial representations of the additive integers, while they converge pointwise to the trivial representation in the three-adic topology. A proof of nontriviality for every approximation is not a proof of nontriviality for its limit. This example does not decide the closedness of any particular arithmetic local condition. That condition needs its own theorem.

7.2 What changes when coefficient places are indexed

The indexed theorem takes a source member at every λ. It first proves an existence statement at each place, then uses choice to select a family of target members. The source code explicitly verifies three outputs at every place: finite-depth admissibility, the relation `TransportedAlong`, and membership in the target profile.

This is an indexed family statement. Any interpretation as a compatible arithmetic system depends on the shared data encoded by the conditions and profiles. Merely placing unrelated local objects in one function indexed by λ is not a new compatibility theorem. The abstract horizontal module is general in the source group, but the terminal assembly printed here specializes that group to `GQ`.

8 From arithmetic occurrence to a finite separator

The constructed Claim 9.1 module has two declarations. The first provides its `ExistentialArithmeticCoverage` proposition by calling the constructed Chebotarev bridge. The second applies the arithmetic system-separation theorem to that proposition. In this later source there is no free external coverage parameter in the final separator signature. The earlier scalar/Mellin checkpoint had retained that parameter. Both sources remain in the archive.

8.1 What the separator accepts

The family consists of finite-dimensional irreducible representations, compatible character values in a common field, continuous characters at each realization, and pairwise nonisomorphism in the stated memberwise sense. At each coefficient place the bad-prime set is finite. The caller selects a distinguished place and may forbid an additional finite set of rational primes.

The result selects distinct allowed rational primes. The common sample matrix has nonzero determinant, and that remains true after each coefficient-field embedding. Matrix entries identify with the representation characters wherever the required good-prime condition holds. The terminal wrapper then converts this matrix identity into the literal trace determinant needed by the carrier theorem.

Distinct primes alone would not suffice. Samples could repeat one character row and leave the matrix singular. Nonisomorphism, arithmetic occurrence, and the finite-dimensional separation argument do different work.

8.2 The finite character-table model

With columns trivial, sign, and standard, use

$$C = \begin{pmatrix} 1 & 1 & 2 \\ 1 & -1 & 0 \\ 1 & 1 & -1 \end{pmatrix}, \qquad \det C = 6, \qquad C \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \\ 1 \end{pmatrix}.$$

The table explains multiplicity recovery after separating samples exist. It does not produce rational primes with prescribed Frobenius behavior. The arithmetic occurrence construction is represented by the preserved source and its inherited verification record, not by this finite example.

8.3 Exact recovery can still lose finite precision

Over characteristic zero, C is invertible. Modulo eight, $\delta m = (4, 4, 0)$ is nonzero but $C\delta m = (8, 0, 8)$ vanishes. The denominator valuation consumes one binary digit. Traces modulo sixteen distinguish this particular ambiguity. The example separates invertibility over a field from invertibility over a residue ring. Atlas V17 gives the complete small model.

9 Carrier recovery is a construction inside supplied data

Let the supplied carrier be equivariantly presented as $\bigoplus_i W_i^{\oplus c_i}$, and let the chosen multiplicities satisfy $m_i \le c_i$. The copy inclusion and projection give maps j and r with $r \circ j = \mathrm{id}$. Transporting these through the carrier presentation produces an actual split summand. An arbitrary vector of traces does not provide that presentation.

The initial subrepresentation is also supplied with an equivariant presentation by the same selected copies. The nonsingular sample matrix proves that no other natural multiplicity vector has those sample traces. Thus the theorem constructs splittings and proves uniqueness within the declared constituent family. It does not discover an unknown irreducible decomposition from raw observations.

9.1 Three tests after solving a linear equation

For capacities $(2, 1, 1)$, the vector $(2, 0, 1)$ is admissible. The trace vector $(1, 0, 0)$ instead gives $(1/6, 1/6, 1/3)$, which is not integral. The vector $(-1, 1, 1)$ is integral but negative. The vector $(3, 0, 1)$ exceeds capacity. Each has a perfectly well-defined rational trace vector under the same matrix. A nonsingular system alone does not turn its solution into a representation.

9.2 What happens to local data

The source constructs finite copies of supplied local packets. Characteristic polynomials multiply with multiplicities. Relevant dimensions and traces add. The supplied period comparisons similarly produce the selected finite-copy comparison and weight-multiplicity formulas.

These identities do not attach Weil–Deligne or period data to an arbitrary new representation. The data and comparisons are input. An extra semantic condition is handled separately by `FiniteCopyClosed`. The terminal `CarrierRecoveryAt` packages specific polynomial, local, and period identities. It does not universally quantify over every possible local predicate. The separate `selected_local_condition` theorem applies only after its finite-copy-closure premise has been proved.

A fixed nontrivial determinant character illustrates the distinction. Two one-dimensional constituents can each have determinant χ, while their direct sum has determinant χ^2, not χ. A determinant target must be combined according to its actual product law. The word local does not make a condition stable under arbitrary sums.

10 Occupancy and the missing nonzero premise

The identity $\mathrm{Tr}(P) = \mathrm{rank}(P)$ for a projector over a characteristic-zero field is interpreted through the natural-number rank cast into that field. Positivity means positivity of this integer, not an order on an arbitrary coefficient field. In positive characteristic, a nonzero projector can have field-valued trace zero. The rank remains the correct nonemptiness test.

For commuting split semisimple operators, products of their exact spectral projectors detect joint eigenspaces. Pairwise occupancy need not imply joint occupancy. Atlas V18 uses six binary labels for which each pair of the target conditions $(0, 1, 1)$ is witnessed, while the complete triple is absent. The product projector is zero although all pairwise tests pass.

10.1 Why the terminal assembly permits an empty selection

The multiplicities are natural numbers. The vector $m_i = 0$ satisfies $m_i \le c_i$ for every carrier. Its selected copy space is zero, its splitting is valid, and its polynomial products are empty products with value one. All of this is consistent with the formal theorem.

To infer a nonzero summand, an application needs an index i with $m_i > 0$ and a nonzero constituent W_i. Under the finite-dimensional field assumptions,

$$\dim\Big(\bigoplus_i W_i^{\oplus m_i}\Big) = \sum_i m_i \dim W_i > 0$$

then gives the desired conclusion. Dropping that premise would strengthen the theorem to a false statement.

The indexed required-packet criterion is a different statement. It requires a nonzero vector in every supplied required packet. A finite sample of packet indices need not test them all. It also does not produce one common vector belonging to several unrelated carriers. Quantifiers and carrier types must remain visible in the application.

11 The final conjunction and its exact logical strength

Write $A(\sigma)$ for the transported-family conclusion and $B(S)$ for carrier recovery. The assembly has the form

$$(\exists \sigma\ A(\sigma))\ \wedge\ (\exists S\ B(S)).$$

It does not assert an isomorphism between the two witnesses. In a finite logical model, A may hold only at zero and B only at one. Both existential statements are true, but no object satisfies both. This elementary countermodel explains why conjoining proofs cannot create a missing comparison.

In the actual source, the separation is stronger than different names: the transported family is represented by matrices over coefficient rings, whereas carrier recovery is formulated through constituents and presentations over coefficient fields. A comparison may need scalar extension, an equivariant isomorphism, and compatibility with the local structures. There is no canonical identification merely because both outputs are called a system or a packet.

11.1 Read the six proof lines

The terminal declaration first uses `constructor`. The left branch calls `exists_transported_system_of_inverseLimitClosed`. The right branch calls `carrier_recovery_from_arithmetic_separator`. The latter obtains a trace separator and invokes the strict compatible-carrier theorem. The dependency pattern is acyclic and the two branches share only the parameters they actually use.

The carrier statement is at an arbitrary distinguished coefficient place. Repeating the theorem at all places requires the corresponding carrier and initial-subrepresentation presentations there. Choice can select witnesses once those pointwise existence statements have been proved. It cannot fill a place at which their hypotheses are unavailable.

11.2 A practical application contract

An application should first identify its complete source members and the actual finite-level transports. It must then prove the target profile's closure and supply the constituent family, carrier presentations, and compatible sample data. Only after applying the two construction branches should it attempt any desired identification or nonzero conclusion. This order keeps construction, comparison, and occupancy separate rather than using one as an unstated substitute for another.

12 Source provenance and replay boundaries

The three direct dependency modules printed below are copied from the `conditional_checked` archive tree. Their full leaf paths, nested-container lineage, byte lengths, and SHA-256 values are recorded in `SOURCE_EXCERPT_LINEAGE.json`. The terminal production module is also retained as executable UTF-8 source. The printed listing is a view of that source, not an alternate executable file.

12.1 The compact preservation archive

The object store deduplicates leaf payloads by SHA-256 and records each occurrence in its original archive tree. Nested ZIP contents can be restored as directories. This preserves their leaf bytes and lineage, not the original ZIP compression streams or central-directory bytes. A container hash in the map identifies the historical container. It is not a claim that a newly reconstructed ZIP has that hash.

From the release root, the following commands verify and restore the preserved conditional tree. The output directory must be new or empty.

```
python formal_preservation/restore.py --verify
python formal_preservation/restore.py --root conditional_checked --output restored_conditional
```

The original reproduction scripts and saved logs remain under the restored lineage and readable records. They can contain original execution-workspace paths. A fresh Lean replay requires the pinned toolchain, matching dependencies, and corrected local paths. No such replay was performed for this reader edition.

12.2 Interpreting the saved controls

The historical verification reports two warnings-as-errors builds from separate frozen-cache copies, matching corresponding proof objects, and a transitive axiom audit of the exposed endpoint declarations. The allowed axioms are `propext`, `Classical.choice`, and `Quot.sound`. The false-proof and placeholder controls were rejected. Omitting closure or separator premises was also rejected. The injected custom axiom compiled and was then detected by the audit policy. It is inaccurate to say that Lean rejected it as a malformed proof.

The pinned dependency cache and predecessor proof objects were trusted inputs to that saved endpoint replay. Their existence in the archive does not mean this publication rebuilt the full dependency graph. Source identity, saved compilation, and a new cold build are separate claims.

13 The recorded endpoint environment

The pinned toolchain in the saved endpoint report is Lean 4.34.0-rc2, commit `6a10ac8c22beadecabdbb0919c2b50214762f91d`. The Mathlib revision is `aeb72f1e9981b6d35b426b16dca7759be3e7ba06`. The same pinned identifiers were recovered for the separate October 20 production run; this section retains the original endpoint evidence context.

The source record says the four exposed endpoint declarations depend only on `propext`, `Classical.choice`, and `Quot.sound`. A separate October 5 audit reports a 30-module Paper X compilation. Those inherited reports are not counted as new compiler work; the fresh run is reported separately above.

The complete production endpoint has SHA-256 `ba15fd1ea175fecfcc705243c75d1061c8bdf500221dafe86f29fe1c5d21c5e0`. The next section prints it in full, followed by the three direct dependency modules. Those three modules import further dependencies. They are not a self-contained four-file Lean project. The full preserved archive carries the remaining sources and saved evidence.

Typography and executable source. Listings retain original line numbers and source order. Long lines can wrap in print. The quotient-ring slash and direct-sum symbol are rendered with mathematical glyphs when the monospaced font does not supply its glyph. The UTF-8 `.lean` files, not PDF text extraction, are the source of record. The generated rendering files and hashes are described by `PRINTED_SOURCE_MANIFEST.json`.

14 The complete terminal module

File: `ConditionalOccupiedSystemCompletion.lean`. SHA-256: `ba15fd1ea175fecfcc705243c75d1061c8bdf500221dafe86f29fe1c5d21c5e0`.

```
import PaperX.HorizontalTransportCompletion
import PaperX.Claim91ConstructedClosure
import PaperX.StrictCompatibleCarrierCompletion

/-!
# Conditional occupied-system completion

This is the final assembly theorem for Paper X.  It keeps the two conclusions
of the manuscript separate:

* vertical/horizontal completion constructs a simultaneous target family with
  the declared inverse-limit-closed profile;
* arithmetic Frobenius separation and compatible-carrier closure recover the
  declared constituent packet in the known carrier.

No residual companion, carrier, or local comparison is manufactured by this
assembly theorem.  Those are exactly the explicit hypotheses of the conditional
source statement.
-/
set_option autoImplicit false
noncomputable section

namespace PaperX.ConditionalOccupiedSystemCompletion

open scoped BigOperators
open Polynomial
open PaperX.RepresentationLimits
open PaperX.RepresentationClasses
open PaperX.ClassCompletion
open PaperX.InverseLimitLocalConditions
open PaperX.HorizontalTransportCompletion
open PaperX.AbsoluteFrobeniusWitnesses
open PaperX.AbsoluteIntegralFrobenius
open PaperX.CompatibleSystemRigidity
open PaperX.FiniteSeparation
open PaperX.TraceRecovery
open PaperX.CarrierSummands
open PaperX.WeilData
open PaperX.WeilCopies
open PaperX.WeilSmooth
open PaperX.WeilEuler
open PaperX.WeilProfileClosure
open PaperX.SemilinearTateComparison
open PaperX.SemilinearPeriods

universe u z uB

/-- The exact local carrier conclusion proved by Claim 9.2 at one arbitrary
coefficient realization.  Since the coefficient place is arbitrary and the
same `m` is used, this is the pointwise form of the compatible subsystem. -/
def CarrierRecoveryAt
    {K : Type u} [Field K] [CharZero K] [TopologicalSpace K]
    {GG : Type} [Monoid GG] [TopologicalSpace GG]
    {CIdx : Type} [Fintype CIdx] [DecidableEq CIdx]
    {V : CIdx → Type u}
    [∀ i, AddCommGroup (V i)] [∀ i, Module K (V i)]
    [∀ i, FiniteDimensional K (V i)] [∀ i, TopologicalSpace (V i)]
    (ρ : ∀ i, Representation K GG (V i))
    {W : Type z} [AddCommGroup W] [Module K W]
```

```
    (carrier : Representation K GG W)
    (m : CIdx → ℕ)
    {U₀ : Type z} [AddCommGroup U₀] [Module K U₀]
    [FiniteDimensional K U₀]
    (ρU₀ : Representation K GG U₀)
    (sample : CIdx → GG)
    {E : Type uB} [CommRing E] (σE : E →+* K)
    (g : GG) (P : CIdx → E[X])
    {GW : Type} [Group GW] [TopologicalSpace GW]
    {ν : GW →* Kˣ} (H : Subgroup GW) [H.Normal]
    (Dwd : ∀ i, InertialDatum (ν := ν) H (V i)) (φ : GW)
    {Bper : Type uB} [Field Bper] [Algebra K Bper]
    {GP : Type} [Group GP]
    (σB : GP →* (Bper ≃ₐ[K] Bper)) (tate : ℤ → GP →* Kˣ)
    (ρp : ∀ i, Representation K GP (V i))
    (Cper : ∀ i, Comparison σB tate (ρp i)) : Prop :=
  (∃ j : Representation.IntertwiningMap ρU₀ carrier,
    ∃ r : Representation.IntertwiningMap carrier ρU₀,
      ∀ x, r (j x) = x) ∧
  (∀ n : CIdx → ℕ,
    (∀ j, LinearMap.trace K U₀ (ρU₀ (sample j)) =
      LinearMap.trace K _ (copiesRep ρ n (sample j))) → n = m) ∧
  (∃ j : Representation.IntertwiningMap (copiesRep ρ m) carrier,
    ∃ r : Representation.IntertwiningMap carrier (copiesRep ρ m),
      (∀ x, r (j x) = x) ∧
      Continuous (fun gx : GG × Copies V m => copiesRep ρ m gx.1 gx.2) ∧
      (copiesRep ρ m g).charpoly = (∏ i, P i ^ m i).map σE) ∧
  (let S := copiesInertialDatum H Dwd m;
    S.toDatum = copiesDatum (fun i => (Dwd i).toDatum) m ∧
    localCharpoly S.toDatum H φ =
      ∏ i, localCharpoly (Dwd i).toDatum H φ ^ m i ∧
    eulerPolynomial S.toDatum H φ =
      ∏ i, eulerPolynomial (Dwd i).toDatum H φ ^ m i ∧
    Module.finrank K (fixedKernel S.toDatum H) =
      ∑ i, m i * Module.finrank K (fixedKernel (Dwd i).toDatum H) ∧
    LinearMap.trace K _ (fixedRep S.toDatum H φ) =
      ∑ i, (m i : K) * LinearMap.trace K _ (fixedRep (Dwd i).toDatum H φ)) ∧
  (∃ Csel : Comparison σB tate (copiesRep ρp m),
    (∀ q : ℤ,
      weightMultiplicity σB tate Csel q =
        ∑ i, m i * weightMultiplicity σB tate (Cper i) q) ∧
    (∀ q : ℤ, Nonempty (
      periodSpace σB tate (copiesRep ρp m) q ≃ₗ[K]
        Copies (fun i => periodSpace σB tate (ρp i) q) m)))

/-- Claim 9.1's constructed arithmetic separator gives the literal nonsingular
trace sample required by Claim 9.2 at any selected coefficient place. -/
theorem finite_trace_separator_at
    {Λ CIdx : Type} {E : Type uB} [Field E]
    [Fintype CIdx] [DecidableEq CIdx]
    {K : Λ → Type u} [∀ l, Field (K l)] [∀ l, CharZero (K l)]
    [∀ l, TopologicalSpace (K l)] [∀ l, IsTopologicalRing (K l)]
    [∀ l, T2Space (K l)]
    {V : Λ → CIdx → Type u}
    [∀ l i, AddCommGroup (V l i)] [∀ l i, Module (K l) (V l i)]
    [∀ l i, FiniteDimensional (K l) (V l i)]
    (ρ : (l : Λ) → (i : CIdx) → Representation (K l) GQ (V l i))
    [∀ l i, (ρ l i).IsIrreducible]
    (hc : ∀ l i, Continuous (ρ l i).character)
    (emb : (l : Λ) → E →+* K l)
    (t : CIdx → RationalPrime → E)
    (good : Λ → RationalPrime → Prop)
    (hgood : ∀ l, Set.Finite {p | ¬ good l p})
    (compat : ∀ l i p, good l p →
      (ρ l i).character (chosenWitness p).element = emb l (t i p))
    (hne : ∀ i j, MemberwiseEquiv ρ i j → i = j)
```

```
    (l₀ : Λ) (bad : Finset RationalPrime) :
    ∃ v : CIdx → RationalPrime,
      Function.Injective v ∧
      (∀ i, good l₀ (v i)) ∧
      (∀ i, v i ∉ bad) ∧
      Matrix.det (fun j i =>
        LinearMap.trace (K l₀) (V l₀ i)
          (ρ l₀ i (chosenWitness (v j)).element)) ≠ 0 := by
  obtain ⟨v, hv, hvgood, hvbad, _, hmapdet, hentries⟩ :=
    PaperX.Claim91ConstructedClosure.finite_arithmetic_system_separator
      ρ hc emb t good hgood compat hne l₀ bad
  refine ⟨v, hv, hvgood, hvbad, ?_⟩
  have hmatrix :
      (emb l₀).mapMatrix (sampleMatrix t v) =
        (fun j i => LinearMap.trace (K l₀) (V l₀ i)
          (ρ l₀ i (chosenWitness (v j)).element)) := by
    ext j i
    simpa [Representation.character] using hentries l₀ j i (hvgood j)
  rw [← hmatrix]
  exact hmapdet l₀

/-- Claims 9.1 and 9.2 compose: arithmetic occurrence constructs the finite
separator, and the compatible-carrier theorem then recovers the declared packet
and all supplied strict-compatible local data. -/
theorem carrier_recovery_from_arithmetic_separator
    {Λ CIdx : Type} {E : Type uB} [Field E]
    [Fintype CIdx] [DecidableEq CIdx]
    {K : Λ → Type u} [∀ l, Field (K l)] [∀ l, CharZero (K l)]
    [∀ l, TopologicalSpace (K l)] [∀ l, IsTopologicalRing (K l)]
    [∀ l, T2Space (K l)]
    {V : Λ → CIdx → Type u}
    [∀ l i, AddCommGroup (V l i)] [∀ l i, Module (K l) (V l i)]
    [∀ l i, FiniteDimensional (K l) (V l i)]
    [∀ l i, TopologicalSpace (V l i)]
    (ρ : (l : Λ) → (i : CIdx) → Representation (K l) GQ (V l i))
    [∀ l i, (ρ l i).IsIrreducible]
    (hc : ∀ l i, Continuous (ρ l i).character)
    (emb : (l : Λ) → E →+* K l)
    (t : CIdx → RationalPrime → E)
    (good : Λ → RationalPrime → Prop)
    (hgood : ∀ l, Set.Finite {p | ¬ good l p})
    (compat : ∀ l i p, good l p →
      (ρ l i).character (chosenWitness p).element = emb l (t i p))
    (hne : ∀ i j, MemberwiseEquiv ρ i j → i = j)
    (l₀ : Λ) (bad : Finset RationalPrime)
    (hρ : ∀ i, Continuous (fun gx : GQ × V l₀ i => ρ l₀ i gx.1 gx.2))
    {W : Type z} [AddCommGroup W] [Module (K l₀) W]
    (carrier : Representation (K l₀) GQ W)
    (m c : CIdx → ℕ) (hmc : ∀ i, m i ≤ c i)
    (Bcar : W ≃ₗ[K l₀] Copies (V l₀) c)
    (hBcar : ∀ g x, Bcar (carrier g x) = copiesRep (ρ l₀) c g (Bcar x))
    {U₀ : Type z} [AddCommGroup U₀] [Module (K l₀) U₀]
    [FiniteDimensional (K l₀) U₀]
    (ρU₀ : Representation (K l₀) GQ U₀)
    (e₀ : U₀ ≃ₗ[K l₀] Copies (V l₀) m)
    (he₀ : ∀ g x, e₀ (ρU₀ g x) = copiesRep (ρ l₀) m g (e₀ x))
    (g : GQ) (P : CIdx → E[X])
    (hP : ∀ i, (ρ l₀ i g).charpoly = (P i).map (emb l₀))
    {GW : Type} [Group GW] [TopologicalSpace GW]
    {ν : GW →* (K l₀)ˣ} (H : Subgroup GW) [H.Normal]
    (Dwd : ∀ i, InertialDatum (ν := ν) H (V l₀ i)) (φ : GW)
    {Bper : Type uB} [Field Bper] [Algebra (K l₀) Bper]
    {GP : Type} [Group GP]
    (σB : GP →* (Bper ≃ₐ[K l₀] Bper))
    (tate : ℤ → GP →* (K l₀)ˣ)
    (ρp : ∀ i, Representation (K l₀) GP (V l₀ i))
```

```
    (Cper : ∀ i, Comparison σB tate (ρp i)) :
    ∃ v : CIdx → RationalPrime,
      Function.Injective v ∧
      (∀ i, good l₀ (v i)) ∧
      (∀ i, v i ∉ bad) ∧
      CarrierRecoveryAt (K := K l₀) (GG := GQ) (CIdx := CIdx) (V := V l₀)
        (W := W) (U₀ := U₀) (E := E) (GW := GW) (ν := ν)
        (Bper := Bper) (GP := GP)
        (ρ l₀) carrier m ρU₀ (fun j => (chosenWitness (v j)).element)
        (emb l₀) g P H Dwd φ σB tate ρp Cper := by
  obtain ⟨v, hv, hvgood, hvbad, hdet⟩ :=
    finite_trace_separator_at ρ hc emb t good hgood compat hne l₀ bad
  refine ⟨v, hv, hvgood, hvbad, ?_⟩
  simpa [CarrierRecoveryAt] using
    (PaperX.StrictCompatibleCarrierCompletion.compatible_carrier_summand_completion
      (ρ l₀) hp carrier m c hmc Bcar hBcar ρU₀ e₀ he₀
      (fun j => (chosenWitness (v j)).element) hdet
      (emb l₀) g P hP H Dwd φ σB tate ρp Cper)

/-- Final Paper X assembly at an arbitrary coefficient place.  The first output
is the simultaneous family supplied by vertical/horizontal completion and
inverse-limit-closed local conditions.  The second, logically independent,
output is the compatible carrier summand recovered by finite Frobenius traces.
This matches the two-clause conclusion of the manuscript's conditional
occupied-system completion theorem. -/
theorem conditional_occupied_system_completion
    {Λ CIdx : Type} {E : Type uB} [Field E]
    {ι : Type} [Fintype ι] [DecidableEq ι]
    (O : Λ → Type) [∀ l, CommRing (O l)]
    (J : (l : Λ) → Ideal (O l)) [∀ l, IsAdicComplete (J l) (O l)]
    (Csrc Ctgt : (l : Λ) → ClassConditions (G := GQ) (ι := ι) (J l))
    (finiteO : ∀ l k, Finite (O l ⧸ (J l) ^ k))
    (T : ∀ l, AdmissibleClassTransport (J l) (Csrc l) (Ctgt l))
    (TargetProfile : (l : Λ) → MatrixRep (O l) GQ ι → Prop)
    (closed : ∀ l, InverseLimitClosed (J l) (Ctgt l) (TargetProfile l))
    (ρsrc : (l : Λ) → MatrixRep (O l) GQ ι)
    (hρsrc : ∀ l k,
      (Csrc l).allowed k (classOf (reduce (J l) (ρsrc l) k)))
    [Fintype CIdx] [DecidableEq CIdx]
    {K : Λ → Type u} [∀ l, Field (K l)] [∀ l, CharZero (K l)]
    [∀ l, TopologicalSpace (K l)] [∀ l, IsTopologicalRing (K l)]
    [∀ l, T2Space (K l)]
    {V : Λ → CIdx → Type u}
    [∀ l i, AddCommGroup (V l i)] [∀ l i, Module (K l) (V l i)]
    [∀ l i, FiniteDimensional (K l) (V l i)]
    [∀ l i, TopologicalSpace (V l i)]
    (ρ : (l : Λ) → (i : CIdx) → Representation (K l) GQ (V l i))
    [∀ l i, (ρ l i).IsIrreducible]
    (hc : ∀ l i, Continuous (ρ l i).character)
    (emb : (l : Λ) → E →+* K l)
    (t : CIdx → RationalPrime → E)
    (good : Λ → RationalPrime → Prop)
    (hgood : ∀ l, Set.Finite {p | ¬ good l p})
    (compat : ∀ l i p, good l p →
      (ρ l i).character (chosenWitness p).element = emb l (t i p))
    (hne : ∀ i j, MemberwiseEquiv ρ i j → i = j)
    (l₀ : Λ) (bad : Finset RationalPrime)
    (hp : ∀ i, Continuous (fun gx : GQ × V l₀ i => ρ l₀ i gx.1 gx.2))
    {W : Type z} [AddCommGroup W] [Module (K l₀) W]
    (carrier : Representation (K l₀) GQ W)
    (m c : CIdx → ℕ) (hmc : ∀ i, m i ≤ c i)
    (Bcar : W ≃ₗ[K l₀] Copies (V l₀) c)
    (hBcar : ∀ g x, Bcar (carrier g x) = copiesRep (ρ l₀) c g (Bcar x))
    {U₀ : Type z} [AddCommGroup U₀] [Module (K l₀) U₀]
    [FiniteDimensional (K l₀) U₀]
    (ρU₀ : Representation (K l₀) GQ U₀)
```

```
    (e₀ : U₀ ≃ₗ[K l₀] Copies (V l₀) m)
    (he₀ : ∀ g x, e₀ (ρU₀ g x) = copiesRep (ρ l₀) m g (e₀ x))
    (g : GQ) (P : CIdx → E[X])
    (hP : ∀ i, (ρ l₀ i g).charpoly = (P i).map (emb l₀))
    {GW : Type} [Group GW] [TopologicalSpace GW]
    {ν : GW →* (K l₀)ˣ} (H : Subgroup GW) [H.Normal]
    (Dwd : ∀ i, InertialDatum (ν := ν) H (V l₀ i)) (φ : GW)
    {Bper : Type uB} [Field Bper] [Algebra (K l₀) Bper]
    {GP : Type} [Group GP]
    (σB : GP →* (Bper ≃ₐ[K l₀] Bper))
    (tate : ℤ → GP →* (K l₀)ˣ)
    (ρp : ∀ i, Representation (K l₀) GP (V l₀ i))
    (Cper : ∀ i, Comparison σB tate (ρp i)) :
    (∃ σ : (l : Λ) → MatrixRep (O l) GQ ι,
      (∀ l k, (Ctgt l).allowed k (classOf (reduce (J l) (σ l) k))) ∧
      (∀ l, TransportedAlong (J l) (T l) (ρsrc l) (σ l)) ∧
      ∀ l, TargetProfile l (σ l)) ∧
    (∃ ν : CIdx → RationalPrime,
      Function.Injective ν ∧
      (∀ i, good l₀ (ν i)) ∧
      (∀ i, ν i ∉ bad) ∧
      CarrierRecoveryAt (K := K l₀) (GG := GQ) (CIdx := CIdx) (V := V l₀)
        (W := W) (U₀ := U₀) (E := E) (GW := GW) (ν := ν)
        (Bper := Bper) (GP := GP)
        (ρ l₀) carrier m ρU₀ (fun j => (chosenWitness (ν j)).element)
        (emb l₀) g P H Dwd φ σB tate ρp Cper) := by
  constructor
  · exact exists_transported_system_of_inverseLimitClosed O J Csrc Ctgt finiteO T
      TargetProfile closed ρsrc hρsrc
  · exact carrier_recovery_from_arithmetic_separator ρ hc emb t good hgood compat hne
      l₀ bad hρ carrier m c hmc Bcar hBcar ρU₀ e₀ he₀ g P hP
      H Dwd φ σB tate ρp Cper

end PaperX.ConditionalOccupiedSystemCompletion
```

15 Direct dependency: horizontal transport

File: source/HorizontalTransportCompletion.lean. SHA-256: 91c7652afda255943a57ac983414bd6ee2e786e26899175e51e6b390e1fbbfda.

```
import PaperX.ClassBranchLifting
import PaperX.ClassCompletion
import PaperX.FiniteFiberBranches
import PaperX.InverseLimitLocalConditions

/-!
Paper X: horizontal transport completion block.

This formalizes the finite-depth content of an admissible residual transport:
a map on representation classes at every precision, preservation of the selected
admissible conditions, and naturality with respect to coefficient reduction.

The main theorem proves that such a transport sends a completed source branch
to a compatible target branch and, over an adically complete coefficient ring
with finite coefficient quotients, reconstructs an actual target
representation realizing that transported branch.  No arithmetic transport is
asserted unless supplied through the `allowed` predicates and the transport
fields below.
-/
set_option autoImplicit false

namespace PaperX.HorizontalTransportCompletion

open PaperX.Compactness
open PaperX.FiniteFiberBranches
open PaperX.RepresentationLimits
open PaperX.RepresentationClasses
open PaperX.ClassCompletion
open PaperX.ClassBranchLifting
open PaperX.InverseLimitLocalConditions

variable {R G ι : Type} [CommRing R] [Group G]
  [Fintype ι] [DecidableEq ι]
variable (I : Ideal R)

/-- Finite-depth form of an admissible residual transport.  It acts on actual
change-of-basis classes, carries every selected source condition to the target
condition, and commutes with reduction between arbitrary precisions. -/
structure AdmissibleClassTransport
    (C D : ClassConditions (G := G) (ι := ι) I) where
  map : (k : ℕ) → ClassRep (R ⧸ I ^ k) G ι → ClassRep (R ⧸ I ^ k) G ι
  allowed : ∀ k c, C.allowed k c → D.allowed k (map k c)
  reduction : ∀ {j k : ℕ} (h : j ≤ k) c,
    transitionClass I h (map k c) = map j (transitionClass I h c)

/-- The inverse system of admissible representation classes for one selected
condition family. -/
def conditionTower (C : ClassConditions (G := G) (ι := ι) I) : Tower where
  Level k := {c : ClassRep (R ⧸ I ^ k) G ι // C.allowed k c}
  parent k c :=
    ⟨transitionClass I (Nat.le_succ k) c.val,
      C.downward (Nat.le_succ k) c.val c.property⟩

/-- An admissible transport is literally a map of the corresponding inverse
towers. -/
def AdmissibleClassTransport.asTowerMap
    {C D : ClassConditions (G := G) (ι := ι) I}
    (T : AdmissibleClassTransport I C D) :
    TowerMap (conditionTower I C) (conditionTower I D) where
```

```
  map k c := ⟨T.map k c.val, T.allowed k c.val c.property⟩
  parent_commutes k c := by
    apply Subtype.ext
    exact (T.reduction (Nat.le_succ k) c.val).symm

/-- Identity is an admissible transport. -/
def AdmissibleClassTransport.id
    (C : ClassConditions (G := G) (ι := ι) I) :
    AdmissibleClassTransport I C C where
  map := fun _ c => c
  allowed := fun _ _ h => h
  reduction := by
    intro j k h c
    rfl

/-- Admissible transports compose without losing reduction naturality. -/
def AdmissibleClassTransport.comp
    {C D E : ClassConditions (G := G) (ι := ι) I}
    (U : AdmissibleClassTransport I D E)
    (T : AdmissibleClassTransport I C D) :
    AdmissibleClassTransport I C E where
  map := fun k c => U.map k (T.map k c)
  allowed := fun k c hc => U.allowed k (T.map k c) (T.allowed k c hc)
  reduction := by
    intro j k h c
    calc
      transitionClass I h (U.map k (T.map k c))
          = U.map j (transitionClass I h (T.map k c)) :=
            U.reduction h (T.map k c)
      _ = U.map j (T.map j (transitionClass I h c)) :=
            congrArg (U.map j) (T.reduction h c)

/-- A completed integral representation determines its exact admissible class
branch. -/
def realizationBranch
    (C : ClassConditions (G := G) (ι := ι) I)
    (ρ : MatrixRep R G ι)
    (hρ : ∀ k, C.allowed k (classOf (reduce I ρ k))) :
    Branch (conditionTower I C) where
  value k := ⟨classOf (reduce I ρ k), hρ k⟩
  compatible k := by
    apply Subtype.ext
    simp [conditionTower]

/-- Transport an entire compatible branch, not merely one residual root. -/
def transportBranch
    {C D : ClassConditions (G := G) (ι := ι) I}
    (T : AdmissibleClassTransport I C D)
    (b : Branch (conditionTower I C)) : Branch (conditionTower I D) :=
  mapBranch (T.asTowerMap I) b

@[simp] theorem transportBranch_value
    {C D : ClassConditions (G := G) (ι := ι) I}
    (T : AdmissibleClassTransport I C D)
    (b : Branch (conditionTower I C)) (k : ℕ) :
    (transportBranch I T b).value k =
      ⟨T.map k (b.value k).val, T.allowed k (b.value k).val (b.value k).property⟩ :=
  rfl

/-- Forget the admissibility witness while retaining the compatible class
branch. -/
def forgetConditionTower
    (C : ClassConditions (G := G) (ι := ι) I) :
    TowerMap (conditionTower I C) (classTower (G := G) (ι := ι) I) where
  map := fun _ c => c.val
  parent_commutes := by
```

```
    intro k c
    rfl

/-- The transported admissible branch viewed as an ordinary branch of
representation classes. -/
def transportedClassBranch
    {C D : ClassConditions (G := G) (ι := ι) I}
    (T : AdmissibleClassTransport I C D)
    (ρ : MatrixRep R G ι)
    (hρ : ∀ k, C.allowed k (classOf (reduce I ρ k))) :
    Branch (classTower (G := G) (ι := ι) I) :=
  mapBranch (forgetConditionTower I D)
    (transportBranch I T (realizationBranch I C ρ hρ))

@[simp] theorem transportedClassBranch_value
    {C D : ClassConditions (G := G) (ι := ι) I}
    (T : AdmissibleClassTransport I C D)
    (ρ : MatrixRep R G ι)
    (hρ : ∀ k, C.allowed k (classOf (reduce I ρ k))) (k : ℕ) :
    (transportedClassBranch I T ρ hρ).value k =
      T.map k (classOf (reduce I ρ k)) :=
  rfl

/-- Horizontal transport plus vertical completeness reconstructs an actual
integral target realization.  The conclusion remembers the exact transported
class at every finite precision; it is stronger than target nonemptiness. -/
theorem exists_transported_realization [IsAdicComplete I R]
    {C D : ClassConditions (G := G) (ι := ι) I}
    (finiteR : ∀ k, Finite (R ⧸ I ^ k))
    (T : AdmissibleClassTransport I C D)
    (ρ : MatrixRep R G ι)
    (hρ : ∀ k, C.allowed k (classOf (reduce I ρ k))) :
    ∃ σ : MatrixRep R G ι,
      (∀ k, D.allowed k (classOf (reduce I σ k))) ∧
      ∀ k, classOf (reduce I σ k) = T.map k (classOf (reduce I ρ k)) := by
  let b := transportedClassBranch I T ρ hρ
  obtain ⟨σ, hσ⟩ := realization_of_class_branch I finiteR b
  refine ⟨σ, ?_, ?_⟩
  · intro k
    rw [hσ k]
    exact ((transportBranch I T (realizationBranch I C ρ hρ)).value k).property
  · intro k
    exact hσ k

variable [TopologicalSpace G]

/-- If every finite representative of an allowed target class is continuous,
the transported realization can be chosen adically continuous as well. -/
theorem exists_continuous_transported_realization [IsAdicComplete I R]
    {C D : ClassConditions (G := G) (ι := ι) I}
    (finiteR : ∀ k, Finite (R ⧸ I ^ k))
    (T : AdmissibleClassTransport I C D)
    (ρ : MatrixRep R G ι)
    (hρ : ∀ k, C.allowed k (classOf (reduce I ρ k)))
    (continuous : ∀ k (τ : MatrixRep (R ⧸ I ^ k) G ι),
      D.allowed k (classOf τ) → PaperX.RepresentationTopology.FiniteRepContinuous τ) :
    ∃ σ : MatrixRep R G ι,
      (∀ k, D.allowed k (classOf (reduce I σ k))) ∧
      PaperX.RepresentationTopology.AdicRepContinuous I σ ∧
      ∀ k, classOf (reduce I σ k) = T.map k (classOf (reduce I ρ k)) := by
  let b := transportedClassBranch I T ρ hρ
  obtain ⟨σ, hσ, hc⟩ := continuous_realization_of_class_branch I finiteR b
    (fun k τ hτ => continuous k τ (by
      rw [hτ]
      exact ((transportBranch I T (realizationBranch I C ρ hρ)).value k).property))
  refine ⟨σ, ?_, hc, ?_⟩
```

```
  · intro k
    rw [hσ k]
    exact ((transportBranch I T (realizationBranch I C ρ hρ)).value k).property
  · intro k
    exact hσ k

end PaperX.HorizontalTransportCompletion

namespace PaperX.HorizontalTransportCompletion

open PaperX.InverseLimitLocalConditions

open PaperX.RepresentationLimits
open PaperX.RepresentationClasses
open PaperX.ClassCompletion

variable {R G ι : Type} [CommRing R] [Group G]
  [Fintype ι] [DecidableEq ι]
variable (I : Ideal R)

/-- Exact relation between complete realizations induced by a finite-depth
horizontal transport.  It records the whole compatible system of transported
classes, not only the residual class. -/
def TransportedAlong
    {C D : ClassConditions (G := G) (ι := ι) I}
    (T : AdmissibleClassTransport I C D)
    (ρ σ : MatrixRep R G ι) : Prop :=
  ∀ k, classOf (reduce I σ k) = T.map k (classOf (reduce I ρ k))


/-- If the target complete property is known to be inverse-limit closed for the
selected target finite-level conditions, the transported reconstruction satisfies
that property automatically.  This is the literal closure hypothesis appearing
in Claim 11.1; no separate complete-level transport axiom is needed. -/
theorem exists_transported_realization_of_inverseLimitClosed [IsAdicComplete I R]
    {C D : ClassConditions (G := G) (ι := ι) I}
    (finiteR : ∀ k, Finite (R ⧸ I ^ k))
    (T : AdmissibleClassTransport I C D)
    (TargetProfile : MatrixRep R G ι → Prop)
    (closed : InverseLimitClosed I D TargetProfile)
    (ρ : MatrixRep R G ι)
    (hρ : ∀ k, C.allowed k (classOf (reduce I ρ k))) :
    ∃ σ : MatrixRep R G ι,
      (∀ k, D.allowed k (classOf (reduce I σ k))) ∧
      TransportedAlong I T ρ σ ∧ TargetProfile σ := by
  obtain ⟨σ, hD, hT⟩ := exists_transported_realization I finiteR T ρ hρ
  exact ⟨σ, hD, hT,
    (inverseLimitClosed_iff_realizations I D TargetProfile).mp closed σ hD⟩

/-- Arithmetic semantics are intentionally separated from the class-level
transport.  A selected complete profile is preserved only when this implication
is actually supplied/proved for the intended arithmetic realization. -/
def PreservesCompleteProfile
    {C D : ClassConditions (G := G) (ι := ι) I}
    (T : AdmissibleClassTransport I C D)
    (SourceProfile TargetProfile : MatrixRep R G ι → Prop) : Prop :=
  ∀ ρ σ, TransportedAlong I T ρ σ → SourceProfile ρ → TargetProfile σ

/-- The class transport and an explicit complete-profile preservation theorem
compose without conflating finite congruence data with arithmetic semantics. -/
theorem exists_transported_realization_with_profile [IsAdicComplete I R]
    {C D : ClassConditions (G := G) (ι := ι) I}
    (finiteR : ∀ k, Finite (R ⧸ I ^ k))
    (T : AdmissibleClassTransport I C D)
    (SourceProfile TargetProfile : MatrixRep R G ι → Prop)
    (preserves : PreservesCompleteProfile I T SourceProfile TargetProfile)
```

```
    (ρ : MatrixRep R G ι)
    (hρ : ∀ k, C.allowed k (classOf (reduce I ρ k)))
    (hSource : SourceProfile ρ) :
    ∃ σ : MatrixRep R G ι,
      (∀ k, D.allowed k (classOf (reduce I σ k))) ∧
      TransportedAlong I T ρ σ ∧ TargetProfile σ := by
  obtain ⟨σ, hD, hT⟩ := exists_transported_realization I finiteR T ρ hρ
  exact ⟨σ, hD, hT, preserves ρ σ hT hSource⟩

section Indexed

variable {Λ : Type}
variable (O : Λ → Type) [∀ l, CommRing (O l)]
variable {G ι : Type} [Group G] [Fintype ι] [DecidableEq ι]
variable (J : (l : Λ) → Ideal (O l))

/-- Coefficient-indexed horizontal completion.  Once one completed source
member is supplied at each coefficient place, admissible horizontal transports
produce actual target members at every place with exact finite-depth class
matching.  This is the system-level algebraic content of the horizontal half of
Claim 11.1; no Frobenius separation or WD/Hodge--Tate comparison is hidden in
this statement. -/
theorem exists_transported_system
    [∀ l, IsAdicComplete (J l) (O l)]
    (C D : (l : Λ) → ClassConditions (G := G) (ι := ι) (J l))
    (finiteO : ∀ l k, Finite (O l ⧸ (J l) ^ k))
    (T : ∀ l, AdmissibleClassTransport (J l) (C l) (D l))
    (ρ : (l : Λ) → MatrixRep (O l) G ι)
    (hρ : ∀ l k, (C l).allowed k (classOf (reduce (J l) (ρ l) k))) :
    ∃ σ : (l : Λ) → MatrixRep (O l) G ι,
      (∀ l k, (D l).allowed k (classOf (reduce (J l) (σ l) k))) ∧
      ∀ l, TransportedAlong (J l) (T l) (ρ l) (σ l) := by
  classical
  have H : ∀ l, ∃ σ : MatrixRep (O l) G ι,
      (∀ k, (D l).allowed k (classOf (reduce (J l) σ k))) ∧
      TransportedAlong (J l) (T l) (ρ l) σ := by
    intro l
    exact exists_transported_realization (J l) (finiteO l) (T l) (ρ l) (hρ l)
  refine ⟨fun l => Classical.choose (H l), ?_, ?_⟩
  · intro l k
    exact (Classical.choose_spec (H l)).1 k
  · intro l
    exact (Classical.choose_spec (H l)).2


/-- Coefficient-indexed form using exactly the inverse-limit-closure premise of
Claim 11.1.  Every target member reconstructed from transported finite classes
inherits the supplied closed local/arithmetic predicate. -/
theorem exists_transported_system_of_inverseLimitClosed
    [∀ l, IsAdicComplete (J l) (O l)]
    (C D : (l : Λ) → ClassConditions (G := G) (ι := ι) (J l))
    (finiteO : ∀ l k, Finite (O l ⧸ (J l) ^ k))
    (T : ∀ l, AdmissibleClassTransport (J l) (C l) (D l))
    (TargetProfile : (l : Λ) → MatrixRep (O l) G ι → Prop)
    (closed : ∀ l, InverseLimitClosed (J l) (D l) (TargetProfile l))
    (ρ : (l : Λ) → MatrixRep (O l) G ι)
    (hρ : ∀ l k, (C l).allowed k (classOf (reduce (J l) (ρ l) k))) :
    ∃ σ : (l : Λ) → MatrixRep (O l) G ι,
      (∀ l k, (D l).allowed k (classOf (reduce (J l) (σ l) k))) ∧
      (∀ l, TransportedAlong (J l) (T l) (ρ l) (σ l)) ∧
      ∀ l, TargetProfile l (σ l) := by
  classical
  have H : ∀ l, ∃ σ : MatrixRep (O l) G ι,
      (∀ k, (D l).allowed k (classOf (reduce (J l) σ k))) ∧
      TransportedAlong (J l) (T l) (ρ l) σ ∧ TargetProfile l σ := by
    intro l
```

```
    exact exists_transported_realization_of_inverseLimitClosed (J l) (finiteO l)
      (T l) (TargetProfile l) (closed l) (ρ l) (hρ l)
  refine ⟨fun l => Classical.choose (H l), ?_, ?_, ?_⟩
  · intro l k
    exact (Classical.choose_spec (H l)).1 k
  · intro l
    exact (Classical.choose_spec (H l)).2.1
  · intro l
    exact (Classical.choose_spec (H l)).2.2

/-- Indexed version with an explicit arithmetic/local semantic profile.  The
only semantic premise is the named preservation implication; the compactness
and reconstruction layer does not manufacture Hodge--Tate, Weil--Deligne,
fixed-determinant, or other deformation properties. -/
theorem exists_transported_system_with_profile
    [∀ l, IsAdicComplete (J l) (O l)]
    (C D : (l : Λ) → ClassConditions (G := G) (ι := ι) (J l))
    (finiteO : ∀ l k, Finite (O l ⧸ (J l) ^ k))
    (T : ∀ l, AdmissibleClassTransport (J l) (C l) (D l))
    (SourceProfile TargetProfile : (l : Λ) → MatrixRep (O l) G ι → Prop)
    (preserves : ∀ l,
      PreservesCompleteProfile (J l) (T l) (SourceProfile l) (TargetProfile l))
    (ρ : (l : Λ) → MatrixRep (O l) G ι)
    (hρ : ∀ l k, (C l).allowed k (classOf (reduce (J l) (ρ l) k)))
    (hSource : ∀ l, SourceProfile l (ρ l)) :
    ∃ σ : (l : Λ) → MatrixRep (O l) G ι,
      (∀ l k, (D l).allowed k (classOf (reduce (J l) (σ l) k))) ∧
      (∀ l, TransportedAlong (J l) (T l) (ρ l) (σ l)) ∧
      ∀ l, TargetProfile l (σ l) := by
  classical
  have H : ∀ l, ∃ σ : MatrixRep (O l) G ι,
      (∀ k, (D l).allowed k (classOf (reduce (J l) σ k))) ∧
      TransportedAlong (J l) (T l) (ρ l) σ ∧ TargetProfile l σ := by
    intro l
    exact exists_transported_realization_with_profile (J l) (finiteO l) (T l)
      (SourceProfile l) (TargetProfile l) (preserves l) (ρ l) (hρ l) (hSource l)
  refine ⟨fun l => Classical.choose (H l), ?_, ?_, ?_⟩
  · intro l k
    exact (Classical.choose_spec (H l)).1 k
  · intro l
    exact (Classical.choose_spec (H l)).2.1
  · intro l
    exact (Classical.choose_spec (H l)).2.2

end Indexed

end PaperX.HorizontalTransportCompletion
```

16 Direct dependency: constructed arithmetic separation

File: source/Claim91ConstructedClosure.lean. SHA-256: f2d9bca535fe4fc8c2c6702e64e034cc3447de698cd29c0931f8ceb4642d751a.

```
import PaperX.ConstructedChebotarevBridge
import PaperX.ArithmeticSystemSeparation

/-!
# Constructed Claim 9.1 endpoint
The original Block37 separator statement, with the external API parameter
discharged by the actual general Frobenius occurrence construction.
-/
set_option autoImplicit false
noncomputable section

namespace PaperX.Claim91ConstructedClosure

open PaperX.AbsoluteFrobeniusWitnesses
open PaperX.AbsoluteIntegralFrobenius
open PaperX.ChebotarevExternalAPIAdapter
open PaperX.CompatibleSystemRigidity
open PaperX.FiniteSeparation PaperX.TraceRecovery

/-- The constructed Chebotarev implementation closes the exact
arithmetic occurrence obligation in Paper X Claim 9.1. -/
theorem existentialArithmeticCoverage :
    PaperX.GeneralArithmeticCoverage.ExistentialArithmeticCoverage :=
  PaperX.ConstructedChebotarevBridge.existentialArithmeticCoverage

variable {Λ I E : Type*} [Field E]
variable [Fintype I] [DecidableEq I]
variable {K : Λ → Type*} [∀ l, Field (K l)] [∀ l, CharZero (K l)]
  [∀ l, TopologicalSpace (K l)] [∀ l, IsTopologicalRing (K l)] [∀ l, T2Space (K l)]
variable {W : Λ → I → Type*} [∀ l i, AddCommGroup (W l i)]
  [∀ l i, Module (K l) (W l i)] [∀ l i, FiniteDimensional (K l) (W l i)]
variable (ρ : (l : Λ) → (i : I) → Representation (K l) GQ (W l i))
variable [∀ l i, (ρ l i).IsIrreducible]
variable (hc : ∀ l i, Continuous (ρ l i).character)
variable (emb : (l : Λ) → E →+* K l)
variable (t : I → RationalPrime → E)
variable (good : Λ → RationalPrime → Prop)
variable (hgood : ∀ l, Set.Finite {p | ¬ good l p})
variable (compat : ∀ l i p, good l p →
  (ρ l i).character (chosenWitness p).element = emb l (t i p))
variable (hne : ∀ i j, MemberwiseEquiv ρ i j → i = j)

include hc hgood compat hne in
/-- Full recorded Claim 9.1 separator conclusion with no external API premise.  The selected rational primes avoid every
prescribed finite forbidden set. -/
theorem finite_arithmetic_system_separator
    (l₀ : Λ) (bad : Finset RationalPrime) :
    ∃ v : I → RationalPrime, Function.Injective v ∧
      (∀ i, good l₀ (v i)) ∧ (∀ i, v i ∉ bad) ∧
      (sampleMatrix t v).det ≠ 0 ∧
      (∀ l, ((emb l).mapMatrix (sampleMatrix t v)).det ≠ 0) ∧
      ∀ l j i, good l (v j) →
        (emb l).mapMatrix (sampleMatrix t v) j i =
          (ρ l i).character (chosenWitness (v j)).element := by
  exact PaperX.ArithmeticSystemSeparation.finite_arithmetic_system_separator
    ρ hc emb t good hgood compat hne
    existentialArithmeticCoverage l₀ bad

end PaperX.Claim91ConstructedClosure
```

17 Direct dependency: strict compatible carrier closure

File: source/StrictCompatibleCarrierCompletion.lean. SHA-256: 4fa6b2fd0c212d75e1158049e777f94217b1dab5ce5f84c2690dfcb826406388.

```
import PaperX.CarrierCompatibility
import PaperX.WeilProfileClosure
import PaperX.SemilinearTateComparison
import PaperX.SemilinearPeriods

/-!
# Compatible-carrier summand completion

This is the source-faithful closure theorem for Paper X Claim 9.2.  The
constituent representations, their strict local Weil packets, and their genuine
semilinear Tate comparisons are inputs.  The selected multiplicities are then
used at every coefficient realization.  The theorem constructs the actual split
summand and proves closure of all local data under those finite copies.

It does not construct a compatible carrier or prove that an arbitrary p-adic
representation is Hodge--Tate.  Those claims are not part of the conditional
carrier theorem.
-/
set_option autoImplicit false
noncomputable section

namespace PaperX.StrictCompatibleCarrierCompletion

open scoped BigOperators
open Polynomial
open PaperX.CarrierSummands
open PaperX.CarrierCompatibility
open PaperX.WeilData
open PaperX.WeilCopies
open PaperX.WeilSmooth
open PaperX.WeilEuler
open PaperX.WeilProfileClosure
open PaperX.SemilinearTateComparison
open PaperX.SemilinearPeriods

universe u v w z uB


/-- A semantic local condition on representations over `K`. -/
def LocalCondition (K : Type u) (G : Type v) [CommRing K] [Monoid G] :=
  ∀ (X : Type u), [AddCommGroup X] → [Module K X] → Representation K G X → Prop

/-- A local condition is finite-copy closed when constituent instances imply
its instance on every finite copy carrier.  This explicit hypothesis is
necessary for conditions, such as fixed determinant, which are not automatically
stable under direct sum. -/
def FiniteCopyClosed
    {K : Type u} {G : Type v} [CommRing K] [Monoid G]
    (Q : LocalCondition K G) : Prop :=
  ∀ {I : Type} [Fintype I] {V : I → Type u}
    [∀ i, AddCommGroup (V i)] [∀ i, Module K (V i)]
    (ρ : ∀ i, Representation K G (V i)) (m : I → ℕ),
      (∀ i, Q (V i) (ρ i)) → Q (Copies V m) (copiesRep ρ m)

/-- Every explicitly finite-copy-closed local condition is inherited by the
selected compatible summand. -/
theorem selected_local_condition
    {K : Type u} {G : Type v} [CommRing K] [Monoid G]
    {I : Type} [Fintype I] {V : I → Type u}
```

```
    [∀ i, AddCommGroup (V i)] [∀ i, Module K (V i)]
    (ρ : ∀ i, Representation K G (V i)) (m : I → ℕ)
    (Q : LocalCondition K G) (hclosed : FiniteCopyClosed Q)
    (hconst : ∀ i, Q (V i) (ρ i)) :
    Q (Copies V m) (copiesRep ρ m) :=
  hclosed ρ m hconst

/-- At one coefficient realization, a multiplicity vector bounded by the
carrier capacities produces an actual equivariant split summand, a continuous
selected action, and the expected common Frobenius polynomial. -/
theorem selected_global_carrier
    {K : Type u} [Field K] [TopologicalSpace K]
    {G : Type v} [Monoid G] [TopologicalSpace G]
    {I : Type} [Fintype I]
    {V : I → Type u}
    [∀ i, AddCommGroup (V i)] [∀ i, Module K (V i)]
    [∀ i, FiniteDimensional K (V i)] [∀ i, TopologicalSpace (V i)]
    (ρ : ∀ i, Representation K G (V i))
    (hρ : ∀ i, Continuous (fun gx : G × V i => ρ i gx.1 gx.2))
    {W : Type z} [AddCommGroup W] [Module K W]
    (carrier : Representation K G W)
    (m c : I → ℕ) (hmc : ∀ i, m i ≤ c i)
    (B : W ≃ₗ[K] Copies V c)
    (hB : ∀ g x, B (carrier g x) = copiesRep ρ c g (B x))
    {E : Type uB} [CommRing E] (σE : E →+* K)
    (g : G) (P : I → E[X])
    (hP : ∀ i, (ρ i g).charpoly = (P i).map σE) :
    ∃ j : Representation.IntertwiningMap (copiesRep ρ m) carrier,
      ∃ r : Representation.IntertwiningMap carrier (copiesRep ρ m),
        (∀ x, r (j x) = x) ∧
        Continuous (fun gx : G × Copies V m => copiesRep ρ m gx.1 gx.2) ∧
        (copiesRep ρ m g).charpoly = (∏ i, P i ^ m i).map σE := by
  let jlin : Copies V m →ₗ[K] W := transportedInclusion hmc B
  let rlin : W →ₗ[K] Copies V m := transportedRetraction hmc B
  let j : Representation.IntertwiningMap (copiesRep ρ m) carrier :=
    jlin.intertwiningMap_of_isIntertwiningMap _ _ (by
      intro h x
      exact transported_inclusion_equivariant ρ carrier hmc B hB h x)
  let r : Representation.IntertwiningMap carrier (copiesRep ρ m) :=
    rlin.intertwiningMap_of_isIntertwiningMap _ _ (by
      intro h x
      exact transported_retraction_equivariant ρ carrier hmc B hB h x)
  refine ⟨j, r, ?_, ?_, ?_⟩
  · intro x
    exact transported_split hmc B x
  · exact continuous_copies_action ρ m hρ
  · exact common_polynomial σE ρ m g P hP

/-- Under an explicit nonsingular character sample, the coefficient-independent
multiplicity vector used by the completion is unique.  This is the formal
version of uniqueness in the semisimple carrier decomposition. -/
theorem selected_multiplicities_unique
    {K : Type u} [Field K] [CharZero K]
    {G : Type v} [Monoid G]
    {I : Type} [Fintype I] [DecidableEq I]
    {V : I → Type u}
    [∀ i, AddCommGroup (V i)] [∀ i, Module K (V i)]
    [∀ i, FiniteDimensional K (V i)]
    (ρ : ∀ i, Representation K G (V i))
    (m : I → ℕ) (sample : I → G)
    (hdet : Matrix.det (fun j i => LinearMap.trace K _ (ρ i (sample j))) ≠ 0) :
    ∀ n : I → ℕ,
      (∀ j, LinearMap.trace K _ (copiesRep ρ m (sample j)) =
        LinearMap.trace K _ (copiesRep ρ n (sample j))) → n = m := by
  intro n htrace
  exact (multiplicities_unique_from_actual_traces ρ m n sample hdet htrace).symm
```

```
126
127 /-- Supplied constituent Weil--monodromy packets are closed under the selected
128 finite copies, including smooth inertia and the exact local polynomial,
129 dimension, and trace formulas. -/
130 theorem selected_weil_packets
131     {K : Type u} [Field K]
132     {GW : Type v} [Group GW] [TopologicalSpace GW]
133     {I : Type} [Fintype I]
134     {ν : GW →* Kˣ}
135     {V : I → Type u}
136     [∀ i, AddCommGroup (V i)] [∀ i, Module K (V i)]
137     [∀ i, FiniteDimensional K (V i)]
138     (H : Subgroup GW) [H.Normal]
139     (D : ∀ i, InertialDatum (ν := ν) H (V i))
140     (m : I → ℕ) (φ : GW) :
141     let S := copiesInertialDatum H D m
142     S.toDatum = copiesDatum (fun i => (D i).toDatum) m ∧
143     localCharpoly S.toDatum H φ =
144       ∏ i, localCharpoly (D i).toDatum H φ ^ m i ∧
145     eulerPolynomial S.toDatum H φ =
146       ∏ i, eulerPolynomial (D i).toDatum H φ ^ m i ∧
147     Module.finrank K (fixedKernel S.toDatum H) =
148       ∑ i, m i * Module.finrank K (fixedKernel (D i).toDatum H) ∧
149     LinearMap.trace K _ (fixedRep S.toDatum H φ) =
150       ∑ i, (m i : K) * LinearMap.trace K _ (fixedRep (D i).toDatum H φ) := by
151   exact copies_local_profile H D m φ
152
153 /-- Genuine semilinear comparison data on every constituent produce the actual
154 comparison on the selected finite copies.  The same construction gives the
155 intrinsic period-space equivalence, so the result is not merely a census of
156 formal weight labels. -/
157 theorem selected_semilinear_period_packets
158     {R : Type u} {B : Type uB} {GP : Type v}
159     [Field R] [Field B] [Algebra R B] [Group GP]
160     (σB : GP →* (B ≃ₐ[R] B)) (tate : ℤ → GP →* Rˣ)
161     {I : Type} [Fintype I]
162     {V : I → Type u}
163     [∀ i, AddCommGroup (V i)] [∀ i, Module R (V i)]
164     (ρ : ∀ i, Representation R GP (V i))
165     (C : ∀ i, Comparison σB tate (ρ i))
166     (m : I → ℕ) :
167     ∃ Csel : Comparison σB tate (copiesRep ρ m),
168       (∀ z : ℤ,
169         weightMultiplicity σB tate Csel z =
170           ∑ i, m i * weightMultiplicity σB tate (C i) z) ∧
171       (∀ z : ℤ, Nonempty (
172         periodSpace σB tate (copiesRep ρ m) z ≃ₗ[R]
173           Copies (fun i => periodSpace σB tate (ρ i) z) m)) := by
174   refine ⟨copiesComparison σB tate ρ C m, ?_, ?_⟩
175   · intro z
176     exact copies_weightMultiplicity σB tate ρ C m z
177   · intro z
178     exact ⟨periodCopiesEquiv σB tate ρ m z⟩
179
180 /-- One-place capstone.  It combines the actual global split summand with the
181 constructed Weil packet and the genuine semilinear comparison.  Applying this
182 pointwise at every coefficient place, with the same `m`, is exactly the
183 compatible-carrier completion theorem. -/
184 theorem one_place_completion
185     {K : Type u} [Field K] [TopologicalSpace K]
186     {GG : Type v} [Monoid GG] [TopologicalSpace GG]
187     {I : Type} [Fintype I]
188     {V : I → Type u}
189     [∀ i, AddCommGroup (V i)] [∀ i, Module K (V i)]
190     [∀ i, FiniteDimensional K (V i)] [∀ i, TopologicalSpace (V i)]
191     (ρ : ∀ i, Representation K GG (V i))
```

```
    (hρ : ∀ i, Continuous (fun gx : GG × V i => ρ i gx.1 gx.2))
    {W : Type z} [AddCommGroup W] [Module K W]
    (carrier : Representation K GG W)
    (m c : I → ℕ) (hmc : ∀ i, m i ≤ c i)
    (Bcar : W ≃ₗ[K] Copies V c)
    (hBcar : ∀ g x, Bcar (carrier g x) = copiesRep ρ c g (Bcar x))
    {E : Type uB} [CommRing E] (σE : E →+* K)
    (g : GG) (P : I → E[X])
    (hP : ∀ i, (ρ i g).charpoly = (P i).map σE)
    {GW : Type v} [Group GW] [TopologicalSpace GW]
    {ν : GW →* Kˣ} (H : Subgroup GW) [H.Normal]
    (D : ∀ i, InertialDatum (ν := ν) H (V i)) (φ : GW)
    {Bper : Type uB} [Field Bper] [Algebra K Bper]
    {GP : Type v} [Group GP]
    (σB : GP →* (Bper ≃ₐ[K] Bper)) (tate : ℤ → GP →* Kˣ)
    (ρp : ∀ i, Representation K GP (V i))
    (C : ∀ i, Comparison σB tate (ρp i)) :
    (∃ j : Representation.IntertwiningMap (copiesRep ρ m) carrier,
      ∃ r : Representation.IntertwiningMap carrier (copiesRep ρ m),
        (∀ x, r (j x) = x) ∧
        Continuous (fun gx : GG × Copies V m => copiesRep ρ m gx.1 gx.2) ∧
        (copiesRep ρ m g).charpoly = (∏ i, P i ^ m i).map σE) ∧
    (let S := copiesInertialDatum H D m;
      S.toDatum = copiesDatum (fun i => (D i).toDatum) m ∧
      localCharpoly S.toDatum H φ =
        ∏ i, localCharpoly (D i).toDatum H φ ^ m i ∧
      eulerPolynomial S.toDatum H φ =
        ∏ i, eulerPolynomial (D i).toDatum H φ ^ m i ∧
      Module.finrank K (fixedKernel S.toDatum H) =
        ∑ i, m i * Module.finrank K (fixedKernel (D i).toDatum H) ∧
      LinearMap.trace K _ (fixedRep S.toDatum H φ) =
        ∑ i, (m i : K) * LinearMap.trace K _ (fixedRep (D i).toDatum H φ)) ∧
    (∃ Csel : Comparison σB tate (copiesRep ρp m),
      (∀ z : ℤ,
        weightMultiplicity σB tate Csel z =
          ∑ i, m i * weightMultiplicity σB tate (C i) z) ∧
      (∀ z : ℤ, Nonempty (
        periodSpace σB tate (copiesRep ρp m) z ≃ₗ[K]
          Copies (fun i => periodSpace σB tate (ρp i) z) m))) := by
  refine ⟨?_, ?_, ?_⟩
  · exact selected_global_carrier ρ hρ carrier m c hmc Bcar hBcar σE g P hP
  · exact selected_weil_packets H D m φ
  · exact selected_semilinear_period_packets σB tate ρp C m


/-- The supplied initial semisimple summand is transported to the canonical
selected-copy realization through its explicit equivariant decomposition.  This
is the formal version of the source theorem's statement
`U_{λ₀} ≃ ⨁ᵢ W_{i,λ₀}^{⊕ mᵢ}`. -/
theorem initial_summand_transport
    {K : Type u} [Field K]
    {G : Type v} [Monoid G]
    {I : Type} [Fintype I]
    {V : I → Type u}
    [∀ i, AddCommGroup (V i)] [∀ i, Module K (V i)]
    (ρ : ∀ i, Representation K G (V i))
    {W : Type z} [AddCommGroup W] [Module K W]
    (carrier : Representation K G W)
    (m c : I → ℕ) (hmc : ∀ i, m i ≤ c i)
    (Bcar : W ≃ₗ[K] Copies V c)
    (hBcar : ∀ g x, Bcar (carrier g x) = copiesRep ρ c g (Bcar x))
    {U₀ : Type z} [AddCommGroup U₀] [Module K U₀]
    (ρU₀ : Representation K G U₀)
    (e₀ : U₀ ≃ₗ[K] Copies V m)
    (he₀ : ∀ g x, e₀ (ρU₀ g x) = copiesRep ρ m g (e₀ x)) :
    ∃ j : Representation.IntertwiningMap ρU₀ carrier,
```

```
      ∃ r : Representation.IntertwiningMap carrier ρU₀,
        ∀ x, r (j x) = x := by
  let jlin : U₀ →ₗ[K] W :=
    (transportedInclusion hmc Bcar).comp e₀.toLinearMap
  let rlin : W →ₗ[K] U₀ :=
    e₀.symm.toLinearMap.comp (transportedRetraction hmc Bcar)
  let j : Representation.IntertwiningMap ρU₀ carrier :=
    jlin.intertwiningMap_of_isIntertwiningMap _ _ (by
      intro g x
      change transportedInclusion hmc Bcar (e₀ (ρU₀ g x)) =
        carrier g (transportedInclusion hmc Bcar (e₀ x))
      rw [he₀]
      exact transported_inclusion_equivariant ρ carrier hmc Bcar hBcar g (e₀ x))
  let r : Representation.IntertwiningMap carrier ρU₀ :=
    rlin.intertwiningMap_of_isIntertwiningMap _ _ (by
      intro g x
      apply e₀.injective
      simpa only [rlin, LinearMap.comp_apply, LinearEquiv.coe_coe,
        e₀.apply_symm_apply, he₀] using
        transported_retraction_equivariant ρ carrier hmc Bcar hBcar g x)
  refine (j, r, ?_)
  intro x
  change rlin (jlin x) = x
  change e₀.symm (transportedRetraction hmc Bcar
      (transportedInclusion hmc Bcar (e₀ x))) = x
  rw [transported_split, e₀.symm_apply_apply]

/-- Frobenius traces of the supplied initial summand determine the same unique
multiplicity packet.  The trace comparison is proved from the explicit
intertwining linear equivalence rather than assumed separately. -/
theorem initial_summand_multiplicities_unique
    {K : Type u} [Field K] [CharZero K]
    {G : Type v} [Monoid G]
    {I : Type} [Fintype I] [DecidableEq I]
    {V : I → Type u}
    [∀ i, AddCommGroup (V i)] [∀ i, Module K (V i)]
    [∀ i, FiniteDimensional K (V i)]
    (ρ : ∀ i, Representation K G (V i))
    (m : I → ℕ)
    {U₀ : Type z} [AddCommGroup U₀] [Module K U₀]
    [FiniteDimensional K U₀]
    (ρU₀ : Representation K G U₀)
    (e₀ : U₀ ≃ₗ[K] Copies V m)
    (he₀ : ∀ g x, e₀ (ρU₀ g x) = copiesRep ρ m g (e₀ x))
    (sample : I → G)
    (hdet : Matrix.det (fun j i => LinearMap.trace K _ (ρ i (sample j))) ≠ 0) :
    ∀ n : I → ℕ,
      (∀ j, LinearMap.trace K U₀ (ρU₀ (sample j)) =
        LinearMap.trace K _ (copiesRep ρ n (sample j))) → n = m := by
  intro n htrace
  apply selected_multiplicities_unique ρ m sample hdet n
  intro j
  have hconj : e₀.conj (ρU₀ (sample j)) = copiesRep ρ m (sample j) := by
    apply LinearMap.ext
    intro x
    change e₀ (ρU₀ (sample j) (e₀.symm x)) = copiesRep ρ m (sample j) x
    rw [he₀, e₀.apply_symm_apply]
  calc
    LinearMap.trace K _ (copiesRep ρ m (sample j)) =
        LinearMap.trace K _ (e₀.conj (ρU₀ (sample j))) := by rw [hconj]
    _ = LinearMap.trace K U₀ (ρU₀ (sample j)) :=
      LinearMap.trace_conj' (ρU₀ (sample j)) e₀
    _ = LinearMap.trace K _ (copiesRep ρ n (sample j)) := htrace j


/-- Source-level capstone for Paper X Claim 9.2.  The initial semisimple
```

```
summand is supplied in the explicit constituent-copy form stated in the amended
manuscript.  The theorem proves its split realization in the carrier, uniqueness
of the multiplicity vector from the finite Frobenius separator, and preservation
of the strict-compatible local packets at an arbitrary coefficient place.
Applying the universally quantified local conclusion at every coefficient place
with the same `m` constructs the compatible subsystem. -/
theorem compatible_carrier_summand_completion
    {K : Type u} [Field K] [CharZero K] [TopologicalSpace K]
    {GG : Type v} [Monoid GG] [TopologicalSpace GG]
    {I : Type} [Fintype I] [DecidableEq I]
    {V : I → Type u}
    [∀ i, AddCommGroup (V i)] [∀ i, Module K (V i)]
    [∀ i, FiniteDimensional K (V i)] [∀ i, TopologicalSpace (V i)]
    (ρ : ∀ i, Representation K GG (V i))
    (hρ : ∀ i, Continuous (fun gx : GG × V i => ρ i gx.1 gx.2))
    {W : Type z} [AddCommGroup W] [Module K W]
    (carrier : Representation K GG W)
    (m c : I → ℕ) (hmc : ∀ i, m i ≤ c i)
    (Bcar : W ≃ₗ[K] Copies V c)
    (hBcar : ∀ g x, Bcar (carrier g x) = copiesRep ρ c g (Bcar x))
    {U₀ : Type z} [AddCommGroup U₀] [Module K U₀]
    [FiniteDimensional K U₀]
    (ρU₀ : Representation K GG U₀)
    (e₀ : U₀ ≃ₗ[K] Copies V m)
    (he₀ : ∀ g x, e₀ (ρU₀ g x) = copiesRep ρ m g (e₀ x))
    (sample : I → GG)
    (hdet : Matrix.det (fun j i => LinearMap.trace K _ (ρ i (sample j))) ≠ 0)
    {E : Type uB} [CommRing E] (σE : E →+* K)
    (g : GG) (P : I → E[X])
    (hP : ∀ i, (ρ i g).charpoly = (P i).map σE)
    {GW : Type v} [Group GW] [TopologicalSpace GW]
    {ν : GW →* Kˣ} (H : Subgroup GW) [H.Normal]
    (D : ∀ i, InertialDatum (ν := ν) H (V i)) (φ : GW)
    {Bper : Type uB} [Field Bper] [Algebra K Bper]
    {GP : Type v} [Group GP]
    (σB : GP →* (Bper ≃ₐ[K] Bper)) (tate : ℤ → GP →* Kˣ)
    (ρp : ∀ i, Representation K GP (V i))
    (C : ∀ i, Comparison σB tate (ρp i)) :
    (∃ j : Representation.IntertwiningMap ρU₀ carrier,
      ∃ r : Representation.IntertwiningMap carrier ρU₀,
        ∀ x, r (j x) = x) ∧
    (∀ n : I → ℕ,
      (∀ j, LinearMap.trace K U₀ (ρU₀ (sample j)) =
        LinearMap.trace K _ (copiesRep ρ n (sample j))) → n = m) ∧
    (∃ j : Representation.IntertwiningMap (copiesRep ρ m) carrier,
      ∃ r : Representation.IntertwiningMap carrier (copiesRep ρ m),
        (∀ x, r (j x) = x) ∧
        Continuous (fun gx : GG × Copies V m => copiesRep ρ m gx.1 gx.2) ∧
        (copiesRep ρ m g).charpoly = (∏ i, P i ^ m i).map σE) ∧
    (let S := copiesInertialDatum H D m;
      S.toDatum = copiesDatum (fun i => (D i).toDatum) m ∧
      localCharpoly S.toDatum H φ =
        ∏ i, localCharpoly (D i).toDatum H φ ^ m i ∧
      eulerPolynomial S.toDatum H φ =
        ∏ i, eulerPolynomial (D i).toDatum H φ ^ m i ∧
      Module.finrank K (fixedKernel S.toDatum H) =
        ∑ i, m i * Module.finrank K (fixedKernel (D i).toDatum H) ∧
      LinearMap.trace K _ (fixedRep S.toDatum H φ) =
        ∑ i, (m i : K) * LinearMap.trace K _ (fixedRep (D i).toDatum H φ)) ∧
    (∃ Csel : Comparison σB tate (copiesRep ρp m),
      (∀ z : ℤ,
        weightMultiplicity σB tate Csel z =
          ∑ i, m i * weightMultiplicity σB tate (C i) z) ∧
      (∀ z : ℤ, Nonempty (
        periodSpace σB tate (copiesRep ρp m) z ≃ₗ[K]
          Copies (fun i => periodSpace σB tate (ρp i) z) m))) := by
```

```
  refine (?_, ?_, ?_, ?_, ?_)
  · exact initial_summand_transport ρ carrier m c hmc Bcar hBcar ρU₀ e₀ he₀
  · exact initial_summand_multiplicities_unique ρ m ρU₀ e₀ he₀ sample hdet
  · exact selected_global_carrier ρ hρ carrier m c hmc Bcar hBcar σE g P hP
  · exact selected_weil_packets H D m φ
  · exact selected_semilinear_period_packets σB tate ρp C m

end PaperX.StrictCompatibleCarrierCompletion
```

18 Companion records and limits

Read `REPORT.md`, `PROOF_AUDIT.md`, `CLAIM_MAP.md`, `CLAIM_STATUS.json`, and `REPRODUCE.md` under `formal_preservation/readable/checked_conditional/`. The later residual attempt remains separately under `formal_preservation/readable/later_residual_attempt/` and `status_records/`. The inherited Block 27 and residue-counting materials remain in `inherited_baseline/`.

The principal reader scripts are `publication/build_publication.py` and `verification/run_checks.py`. Package integrity is checked by `verify_release.py`. The dated final review records failed attempts as well as completed checks. A successful package verification does not replace a mathematical proof review or a new pinned Lean build.

The current monograph includes the recovered 23-entry bibliography with separately recorded identity checks. This guide adds exact internal source routes; bibliography identity checks do not certify every external theorem. Universal companion existence, unknown carrier existence, unrestricted local attachment, and universal automorphic occupancy are outside the delivered conditional theorem. No new result about those existence problems is claimed.

The recovered original historical source and fourteen vector figures are under `original_historical_source/`. Original code and machine evidence use MIT; original writing and figures use CC BY 4.0. Third-party terms, including the Apache-licensed arithmetic support, remain applicable.