

CORE SERIES XII

Patching Support and Hecke Packets

Conditional automorphic completion

Lean and evidence guide

Komorebi P.

Reading release 1.0.0 / 10 October 2026

The formal record and the reader

Reading release 1.0.0 synchronizes this guide with the main monograph and the visual companion. The production source and theorem binders are unchanged. Its 96 modules and 1,229 public named declarations are historical source-census objects, not a claim of 1,229 new discoveries.

Three contracts remain distinct

Record	What it establishes
Revised internal claims	The inherited release maps all 49 revised claims to checked production targets.
Original wording	The historical census remains 41 complete, seven partial, and one unchecked.
This publication release	Source, evidence, finite-example, reference, layout, and packaging checks. No fresh Lean/Lake execution.

The full 49-row table in Appendix B now links to the actual main statements. Each mapped target still has its own displayed binders and source location. A link does not establish definitional equivalence between prose and code.

Read the last two constructions carefully

The companion-completion theorem assumes finite occupied classes, complete separated coefficients, continuous representatives, integral polynomial data, and relations valid in every admissible frame. Visual lesson V23 makes the frame distinction concrete.

The five-gate theorem requires a compatible Hecke point, actual ordinary or derived carriers, and realization at the selected eigensystem. Visual lesson V17 displays the separate premises. Propagation compares two supplied compatible families for the same automorphic object.

Historical dates are not new compiler runs

September 2026 compiler receipts remain September receipts. The 6 October source audits and 8 October standardized edition are retained as predecessors. The current release has its own dated verification reports, with fresh execution separated from inherited records.

Abstract

This guide explains the complete production Lean source accompanying Paper XII, with a source catalog, declaration census, import audit, exact source excerpts, and a claim crosswalk. It separates constructed algebraic and topological objects from hypotheses supplied by an arithmetic application. The checked revised contract comprises 49 internal claims mapped to 291 distinct production targets. The original wording remains a separate ledger: 41 complete, seven partial, and one unchecked. The guide preserves all 96 production modules and their inherited verification records. Its new checks are source, provenance, and document checks; no fresh Lean compilation was available in the finalization environment.

Contents

1 What this guide certifies and how to use it 3

2 Independently checked counts and their definitions 3

3 Source organization, dependency order and hypothesis notation 4

4 Claim 3.1: construct the tensor, then complete it 5

5 Completion and quotient: actual maps, completeness and inverses 7

6 Locality, universal maps and the edge cases 10

7 Claim 3.3: completed global base change and its points 12

8 Ordinary support, patching and intrinsic Fitting ideals 15

9 Derived support uses actual perfect models 16

10 Artin packets, residual augmentation and finite adequacy 18

11 Occupancy, stable cores and adic completion of classes 19

12 Conditional realization and the five separate gates 21

13 External arithmetic inputs and editorial boundaries 24

14 Replay prerequisites, pins and audit procedure 25

15 Package contents, Markdown files and rebuilding this document 26

A Full source catalog 26

B All 49 revised internal claims 29

C A practical reading and inspection sequence 30

1 What this guide certifies and how to use it

The reference production tree is the `formal_release/lean/` tree from the complete v0.17.1 bundle. Its revised manuscript is the 36-page `Paper_XII_Revised_Formalization_Complete_v0.17.0.tex`; the v0.17.1 release reverified its unchanged production sources. This reading edition is dated 2026-10-10 and describes those exact sources, alongside the final manuscript's explicit editorial clarifications. It does not silently replace any production theorem or historical verification record.

Read Sections 4–7 for the completed local and global constructions, Sections 8–12 for support, packets, occupancy and conditional realization, and Section 14 for replay. Appendices list every source module and every revised internal claim. The machine-readable catalog gives full SHA-256 hashes, source locations, imports, declaration names and line numbers. The full production source is supplied; printing every line in this PDF is unnecessary for reproducibility.

Three distinct kinds of evidence must be kept separate. A Lean kernel check verifies the exact formal proposition under its hypotheses and dependencies. An ordinary mathematical proof explains a proposition in mathematical prose, with its own statement and citation obligations. A finite computation verifies the tested cases, numerical identities or model bounds stated by that computation. Neither a finite numerical certificate nor a source-token scan is a substitute for kernel elaboration of the production proof.

Evidence	Exact scope
Inherited Lean replay	September 12, 2026: all 96 production modules compiled; 1,229 named declarations axiom-audited; three invalid controls rejected. Pinned Mathlib cache used.
Finalization source audit	Production hashes, lexical census, direct imports, claim targets, source preservation, and inherited report consistency checked on 2026-10-06.
New guide build	Portable PDF build, embedded-font/glyph/reference checks, and independent fresh-directory PDF/text/page comparison.
Numerical verification	Separate finite and exact computation records explain each tested input and bound. No universal arithmetic statement follows from those tests.

No fresh Lean compilation is claimed in this finalization. The pinned Lean executable, Lake and Elan were unavailable in the finalization environment. Preserved compiler logs and matching source/object hashes are inherited evidence, not a new run of the kernel. The replay commands below enable a reader with the separately supplied pinned runtime to make such a run.

2 Independently checked counts and their definitions

The registered manifest contains exactly 96 distinct modules, and its source path set equals the set of all production `.lean` files. There are 91 files under `PaperXII/` and five under `PaperX/`. Directory names alone do not identify originality: `PaperXII.ReuseCompactness` and `PaperXII.ReuseFiniteSeparation` retain Paper X namespaces and 18 reused theorem declarations. Together with 63 theorem declarations in the five Paper X files, these account for the 81 reused theorems. Paper XII's own public theorem count is 748.

The public named lexical census is 1,229: 829 theorems, 319 definitions, 47 abbreviations, 17 structures and 17 named instances. One private theorem helper, `PaperXII.DeterminantalIdeals.option_factor`, is excluded from the public count. The 829 theorem declarations divide into 748 own and 81 reused; they are supporting lemmas and interfaces as well as theorems implementing numbered manuscript claims. This is not a historical novelty count or a twelve-paper total.

The independent scanner blanks nested block comments, line comments and strings while preserving line numbers. It recognizes anchored named declarations in each module's unique namespace and resolves their qualified names. Its declaration set agrees exactly with the inherited axiom target set. It finds no production `sorry`, `admit`, `axiom`, `unsafe`, or `native_decide` token outside comments/strings. These are lexical findings; elaboration and transitive axioms remain the role of the compiler records.

The 49 revised claim records comprise 27 theorems, 14 corollaries, seven propositions and one lemma. Three worked examples, definitions, remarks and eight external standard facts are outside that internal census. All 291 distinct mapped targets occur in the 1,229-name inherited audit. A target can support several claims; a claim can require several targets. Target count is therefore neither theorem count nor claim count.

The original ledger has 41 complete, seven partial and one unchecked claim. Its eight inherited amendments are 6.12, 9.4, 9.5, 9.6, 10.5, 12.2, 12.4 and 12.6. The revised coverage is 49 complete with zero partial and zero unchecked. Both ledgers are supplied without relabeling the original statements as proved. The former 47/49 textual retraction is superseded by the recovered full production archive and successful September replay; it is not an outstanding formal obligation.

3 Source organization, dependency order and hypothesis notation

The source audit finds 113 direct internal import edges. Every internal import names a registered module and precedes its consumer in `checks/modules.json`. The external direct imports are Mathlib modules and `Std`; the complete list is in `data/census.json`. This checks source-level dependency order, not the entire transitive Mathlib import graph. The production tree reuses Paper X, not Paper XI.

The original 71-module block, ten Fitting modules and 15 completion modules make up the 96-module cumulative release. The 15-module completion block contains 146 public theorem declarations and 225 public named declarations. These are preserved existing work, not newly proved results in this guide. Source preservation and the declaration census are computed against the actual files.

In Lean, square-bracket arguments such as `[CommRingR]` and `[CompleteSpaceP]` are typeclass hypotheses. A proposition-valued ordinary argument such as `(hB:IsAdicmB)` is also a hypothesis. A `variable` command may contribute ambient arguments to later declarations; `include` and `omit` control that contribution. Reading only the line beginning `theorem` can therefore hide essential assumptions. The full exact excerpt modules preserve these commands, imports, namespace and section boundaries.

The source excerpts delivered as `excerpts/*.lean` are byte-identical full modules, including their ambient hypotheses. PDF excerpts select line ranges and retain original line numbers. To avoid missing glyphs in portable code typography, the display copies replace eleven symbols by readable forms, for example `iSup` for the indexed supremum and `/` for Lean's ideal-quotient slash. The complete replacement map is in `excerpts/INDEX.json`. This is a documented display convention only; production source and exact excerpts retain every original Unicode character.

4 Claim 3.1: construct the tensor, then complete it

For a finite index type ι, commutative coefficient ring O, and commutative O-algebras R_i, the raw tensor is Mathlib's actual `PiTensorProduct OR`. It is not a record accepting an already supplied tensor equivalence. The factor inclusion is `PiTensorProduct.singleAlgHom`; the family lift is constructed from a multilinear product and the tensor's universal map.

Given ideals $m_O \subseteq O$, $m_i \subseteq R_i$, and condition ideals I_i, define

$$S = \bigotimes_{i\in\iota,O} R_i, \qquad \mathfrak{a} = m_O S + \sum_i m_i S, \qquad L = \sum_i I_i S.$$

In Lean the finite sums are indexed suprema of mapped ideals. The scalar term is present even if ι is empty. Removing it would change the empty-family topology to the wrong normalization.

Source: `PaperXII/FiniteTensorQuotient.lean`, lines 11–26.

```
variable {O ι : Type*} [CommRing O] [Fintype ι] [DecidableEq ι]
variable (R : ι → Type*) [∀ i, CommRing (R i)] [∀ i, Algebra O (R i)]

abbrev Raw := PiTensorProduct O R

/-- The canonical inclusion of one factor in the actual finite tensor product. -/
def includeFactor (i : ι) : R i →ₐ[O] Raw (O := O) R := PiTensorProduct.singleAlgHom i

/-- The universal algebra map, constructed by the multilinear product of the factor maps. -/
def liftFamily {A : Type*} [CommRing A] [Algebra O A] (f : ∀ i, R i →ₐ[O] A) :
    Raw (O := O) R →ₐ[O] A :=
  PiTensorProduct.liftAlgHom
    ((MultilinearMap.mkPiAlgebra O ι A).compLinearMap (fun i ↦ (f i).toLinearMap))
    (by simp [MultilinearMap.compLinearMap_apply, MultilinearMap.mkPiAlgebra_apply])
    (by intro x y; simp [MultilinearMap.compLinearMap_apply, MultilinearMap.mkPiAlgebra_apply,
      Pi.mul_apply, Finset.prod_mul_distrib])
```

Source: `PaperXII/FiniteTensorQuotient.lean`, lines 60–75.

```
def conditions (J : ∀ i, Ideal (R i)) : Ideal (Raw (O := O) R) :=
  iSup i, (J i).map (includeFactor R i).toRingHom

/-- The defining ideal also retains the base ideal, necessary for the empty family. -/
def definingIdeal (mO : Ideal O) (m : ∀ i, Ideal (R i)) : Ideal (Raw (O := O) R) :=
  mO.map (algebraMap O (Raw (O := O) R)) ⊔ conditions R m

variable (J : ∀ i, Ideal (R i))

/-- Apply the actual quotient map in every factor. -/
def quotientMap : Raw (O := O) R →ₐ[O] Raw (O := O) (fun i ↦ R i ⧸ J i) :=
  liftFamily R (fun i ↦ (includeFactor (fun i ↦ R i ⧸ J i) i).comp (Ideal.Quotient.mkₐ O (J i)))

@[simp] theorem quotientMap_include (i : ι) (x : R i) :
    quotientMap (O := O) R J (includeFactor R i x) =
      includeFactor (fun i ↦ R i ⧸ J i) i (Ideal.Quotient.mk (J i) x) := by
```

The ordinary quotient map is built by applying the ideal quotient in each factor. Its kernel contains L; quotient lifts construct both directions between S/L and $\bigotimes_i (R_i/I_i)$. Tensor extensionality and quotient extensionality prove the inverse identities. This produces an ordinary algebra equivalence, with a

proved exact image for the defining ideal. Ordinary tensor right exactness alone would not establish the completed theorem.

`AdicModel.CarrierRI` is a type synonym that installs the specified ideal-adic topology on R. Its `Completed` object is the actual Hausdorff uniform completion of that carrier. The forgetful raw ring equivalence does not assert continuity into an arbitrarily chosen topology. This prevents the topology from being replaced silently.

Source: `PaperXII/AdicModel.lean`, lines 12–42.

```
12 def Carrier (R : Type*) [CommRing R] (_I : Ideal R) := R
13
14 instance carrierCommRing (R : Type*) [CommRing R] (I : Ideal R) : CommRing (Carrier R I) :=
15   inferInstanceAs (CommRing R)
16
17 /-- Forget the type synonym, without asserting continuity into an arbitrary topology. -/
18 def rawEquiv (R : Type*) [CommRing R] (I : Ideal R) : Carrier R I ≃+* R := RingEquiv.refl R
19
20 instance carrierWithIdeal (R : Type*) [CommRing R] (I : Ideal R) : WithIdeal (Carrier R I) :=
21   ⟨I⟩
22
23 theorem carrier_isAdic (R : Type*) [CommRing R] (I : Ideal R) :
24     IsAdic (WithIdeal.i : Ideal (Carrier R I)) := rfl
25
26 instance carrier_countablyGenerated (R : Type*) [CommRing R] (I : Ideal R) :
27     Filter.IsCountablyGenerated (uniformity (Carrier R I)) := by
28   have := (carrier_isAdic R I).hasBasis_nhds_zero.isCountablyGenerated
29   exact IsUniformAddGroup.uniformity_countably_generated
30
31 /-- This is an actual completion, not a supplied completion-interface object. -/
32 abbrev Completed (R : Type*) [CommRing R] (I : Ideal R) := Completion (Carrier R I)
33
34 def embed (R : Type*) [CommRing R] (I : Ideal R) : R →+* Completed R I :=
35   Completion.coeRingHom.comp (rawEquiv R I).symm.toRingHom
36
37 @[simp] theorem embed_apply (R : Type*) [CommRing R] (I : Ideal R) (x : R) :
38     embed R I x = ((rawEquiv R I).symm x : Completed R I) := rfl
39
40 /-- A ring map carrying the defining ideal into the target ideal is uniformly continuous. -/
41 theorem map_continuous {R S : Type*} [CommRing R] [CommRing S]
42     (I : Ideal R) (J : Ideal S) (f : R →+* S) (h : I.map f ≤ J) :
```

Thus `CompletedTensorRmOm` is the Hausdorff completion of S in the literal $\mathfrak{a}$-adic additive uniformity. Its O-algebra structure comes from the dense embedding of S, and its completed factor maps compose that embedding with the raw factor inclusions. The condition ideal is the closure of the algebraically extended L.

Source: `PaperXII/CompletedFiniteTensor.lean`, lines 16–47.

```
16 namespace PaperXII.FiniteTensor
17
18 variable {O ι : Type*} [CommRing O] [Fintype ι] [DecidableEq ι]
19 variable (R : ι → Type*) [∀ i, CommRing (R i)] [∀ i, Algebra O (R i)]
20 variable (mO : Ideal O) (m : ∀ i, Ideal (R i))
21
22 /-- The actual finite completed tensor product. -/
23 abbrev CompletedTensor := AdicModel.Completed (Raw (O := O) R) (definingIdeal R mO m)
24
25 /-- The algebra structure is induced by the canonical dense embedding. -/
26 instance completedTensorAlgebra : Algebra O (CompletedTensor R mO m) :=
```

```
  ((AdicModel.embed (Raw (O := O) R) (definingIdeal R mO m)).comp
    (algebraMap O (Raw (O := O) R))).toAlgebra

/-- Actual dense tensor inclusion, as an algebra homomorphism. -/
def tensorEmbed : Raw (O := O) R →ₐ[O] CompletedTensor R mO m where
  __ := AdicModel.embed (Raw (O := O) R) (definingIdeal R mO m)
  commutes' _ := rfl

/-- Actual completed factor map. -/
def completedInclude (i : ι) : R i →ₐ[O] CompletedTensor R mO m :=
  (tensorEmbed R mO m).comp (includeFactor R i)

/-- The closed ideal generated by all the supplied local condition ideals. -/
def closedConditions (J : ∀ i, Ideal (R i)) : Ideal (CompletedTensor R mO m) :=
  CompletionQuotient.closedExtension
    (AdicModel.carrierIdeal (definingIdeal R mO m) (conditions R J))

omit [Fintype ι] in
/-- This is precisely the literal closed ideal of the completed factor images. -/
theorem closedConditions_eq (J : ∀ i, Ideal (R i)) :
    closedConditions R mO m J =
```

5 Completion and quotient: actual maps, completeness and inverses

The reusable comparison is

$$\widehat{R}\big/\overline{I\widehat{R}} \;\cong\; \widehat{R/I}.$$

Its ambient assumptions are a commutative ring, a uniform space with uniform additive group, and a topological ring. Countable generation of the additive uniformity is used to prove completeness of the quotient of the completion. There is no assumption supplying completeness of that quotient, an inverse comparison map, or comparison surjectivity. The closure makes the quotient Hausdorff.

Source: `PaperXII/CompletionQuotient.lean`, *lines 18–61.*

```
set_option autoImplicit false
namespace PaperXII.CompletionQuotient

variable {R : Type*} [CommRing R] [UniformSpace R] [IsUniformAddGroup R]
  [IsTopologicalRing R]

/-- Countable uniform bases survive actual Hausdorff completion. -/
theorem completion_countablyGenerated {α : Type*} [UniformSpace α]
    [Filter.IsCountablyGenerated (uniformity α)] :
    Filter.IsCountablyGenerated (uniformity (Completion α)) := by
  have : Filter.IsCountablyGenerated (uniformity (CauchyFilter α)) := by
    obtain ⟨s, hs⟩ := Filter.isCountablyGenerated_iff_exists_antitone_basis.mp
      (inferInstance : Filter.IsCountablyGenerated (uniformity α))
    exact (CauchyFilter.basis_uniformity hs.toHasBasis).isCountablyGenerated
  change Filter.IsCountablyGenerated (uniformity (SeparationQuotient (CauchyFilter α)))
  rw [SeparationQuotient.uniformity_eq]
  infer_instance

/-- The uniformity associated with the actual quotient topology. -/
@[instance_reducible] def quotientUniformSpace {S : Type*} [CommRing S] [TopologicalSpace S]
    [IsTopologicalRing S] (I : Ideal S) : UniformSpace (S ⧸ I) :=
  IsTopologicalAddGroup.rightUniformSpace (S ⧸ I)

attribute [local instance] quotientUniformSpace

instance quotientUniformAddGroup {S : Type*} [CommRing S] [TopologicalSpace S]
    [IsTopologicalRing S] (I : Ideal S) : IsUniformAddGroup (S ⧸ I) :=
```

```
  isUniformAddGroup_of_addCommGroup

/-- Close the extended ideal in the actual completion. -/
def closedExtension (I : Ideal R) : Ideal (Completion R) :=
  (I.map (Completion.coeRingHom : R →+* Completion R)).closure

instance closedExtension_isClosed (I : Ideal R) :
    IsClosed (closedExtension I : Set (Completion R)) := isClosed_closure

/-- The quotient of the completion is complete; no finite-generation assumption. -/
theorem quotient_complete (I : Ideal R) [Filter.IsCountablyGenerated (uniformity R)] :
    CompleteSpace (Completion R / closedExtension I) := by
  have := completion_countablyGenerated (α := R)
  exact QuotientAddGroup.completeSpace_right (Completion R) (closedExtension I).toAddSubgroup

/-- The completion of the quotient receives the canonical completed quotient map. -/
def forwardMap (I : Ideal R) : Completion R →+* Completion (R / I) :=
```

The forward map extends the actual quotient map. Its continuity and the target's separation prove that it kills the closed extended ideal. The backward map first descends the dense raw representatives, then extends from R/I to its completion. The source quotient has its actual quotient topology and the associated additive uniformity. `PaperXII.AdicQuotientTopology` separately proves that this quotient topology is the topology of the image defining ideal.

The inverse laws are proved on dense representatives and extended by continuity. The second law uses quotient surjectivity after completion induction. The ring equivalence and its homeomorphism are then constructed from these proved maps.

Source: `PaperXII/CompletionQuotient.lean`*, lines 121–180.*

```
attribute [local instance] quotient_complete

/-- The reverse comparison extends the quotient map to its actual completion. -/
def backward (I : Ideal R) :
    Completion (R / I) →+* (Completion R / closedExtension I) :=
  Completion.extensionHom (quotientBaseMap I) (quotientBaseMap_continuous I)

theorem backward_continuous (I : Ideal R) : Continuous (backward I) :=
  Completion.continuous_extension

@[simp] theorem backward_coe (I : Ideal R) (x : R / I) :
    backward I (x : Completion (R / I)) = quotientBaseMap I x :=
  Completion.extensionHom_coe _ _ x

/-- First inverse law, established on dense representatives then by continuity. -/
theorem forward_backward (I : Ideal R) (x : Completion (R / I)) :
    forward I (backward I x) = x := by
  refine Completion.induction_on x
    (isClosed_eq ((forward_continuous I).comp (backward_continuous I)) continuous_id) ?_
  intro x
  obtain ⟨r, rfl⟩ := Ideal.Quotient.mk_surjective x
  simp

/-- Second inverse law; the dense representative argument precedes quotient induction. -/
theorem backward_forwardMap (I : Ideal R) (x : Completion R) :
    backward I (forwardMap I x) = Ideal.Quotient.mk (closedExtension I) x := by
  refine Completion.induction_on x
    (isClosed_eq ((backward_continuous I).comp (forwardMap_continuous I))
      (QuotientRing.isOpenQuotientMap_mk _).continuous) ?_
  intro r
  simp

```

```
theorem backward_forward (I : Ideal R) (x : Completion R / closedExtension I) :
    backward I (forward I x) = x := by
  obtain ⟨x, rfl⟩ := Ideal.Quotient.mk_surjective x
  exact backward_forwardMap I x

/-- Completion commutes with quotient by the closed extended ideal. -/
def equiv (I : Ideal R) :
    (Completion R / closedExtension I) ≃+* Completion (R / I) :=
  RingEquiv.ofRingHom (forward I) (backward I)
    (RingHom.ext (forward_backward I)) (RingHom.ext (backward_forward I))

theorem equiv_continuous (I : Ideal R) : Continuous (equiv I) := forward_continuous I

theorem equiv_symm_continuous (I : Ideal R) : Continuous (equiv I).symm :=
  backward_continuous I

@[simp] theorem equiv_mk_coe (I : Ideal R) (r : R) :
    equiv I (Ideal.Quotient.mk (closedExtension I) (r : Completion R)) =
      (Ideal.Quotient.mk I r : Completion (R / I)) := forwardMap_coe I r

/-- A bundled homeomorphism accompanying the ring equivalence. -/
def homeomorph (I : Ideal R) :
    (Completion R / closedExtension I) ≃ₜ Completion (R / I) where
  toEquiv := (equiv I).toEquiv
  continuous_toFun := equiv_continuous I
  continuous_invFun := equiv_symm_continuous I

end Countable
```

Applying this to the raw finite tensor quotient gives the displayed O-algebra equivalence of Claim 3.1. Its forward direction in the manuscript is the inverse of the construction direction. Both maps are continuous, and every factor identity is proved with the actual inclusions.

Source: `PaperXII/CompletedFiniteTensor.lean`, lines 91–133.

```
    CompletedTensor (fun i ↦ R i / J i) mO
      (fun i ↦ (m i).map (Ideal.Quotient.mk (J i))) ≃ₐ[O]
      (CompletedTensor R mO m / closedConditions R mO m J) :=
  (show (CompletedTensor R mO m / closedConditions R mO m J) ≃ₐ[O]
      CompletedTensor (fun i ↦ R i / J i) mO
        (fun i ↦ (m i).map (Ideal.Quotient.mk (J i))) from
    { __ := completedQuotientEquiv R mO m J
      commutes' r := by
        change completedQuotientEquiv R mO m J
          (Ideal.Quotient.mk _ (tensorEmbed R mO m (algebraMap O (Raw (O := O) R) r))) = _
        rw [completedQuotientEquiv_mk_embed, (quotientMap R J).commutes]
        rfl }).symm

/-- Both maps of the completed tensor quotient equivalence are continuous. -/
theorem completed_tensor_quotient_continuous :
    Continuous (completed_tensor_quotient R mO m J) ∧
      Continuous (completed_tensor_quotient R mO m J).symm :=
  ⟨completedQuotientEquiv_symm_continuous R mO m J,
    completedQuotientEquiv_continuous R mO m J⟩

/-- The isomorphism respects every actual canonical factor map. -/
@[simp] theorem completed_tensor_quotient_factor (i : ι) (x : R i) :
    completed_tensor_quotient R mO m J
      (completedInclude (fun i ↦ R i / J i) mO
        (fun i ↦ (m i).map (Ideal.Quotient.mk (J i))) i (Ideal.Quotient.mk (J i) x)) =
      Ideal.Quotient.mk _ (completedInclude R mO m i x) := by
  apply (completedQuotientEquiv R mO m J).injective
  change completedQuotientEquiv R mO m J
    ((completedQuotientEquiv R mO m J).symm _) = _
```

```
  rw [RingEquiv.apply_symm_apply]
  symm
  simpa only [quotientMap_include, completedInclude, AlgHom.comp_apply] using
    completedQuotientEquiv_mk_embed R mO m J (includeFactor R i x)

attribute [local instance] CompletionQuotient.quotientUniformSpace

omit [Fintype ι] in
/-- The right-hand quotient is itself a genuine complete Hausdorff ring. -/
theorem completed_tensor_quotient_complete :
    CompleteSpace (CompletedTensor R mO m / closedConditions R mO m J) ∧
      T2Space (CompletedTensor R mO m / closedConditions R mO m J) :=
  (CompletionQuotient.quotient_complete _, inferInstance)

```

The source theorem `closedConditions_eq` identifies the ideal with the literal closure of the ideal generated by the completed factor images. The supplementary consumer `exact_completed_kernel` additionally asserts exact equality of the canonical completed-map kernel and the stated closed ideal. That consumer is separate from the production theorem census.

The general comparison requires neither locality nor Noetherianity of the raw tensor ring. `PaperXII.AdicCompletionComparison` independently identifies the uniform completion with Mathlib's algebraic inverse-limit adic completion. Finite generation controls ideal powers and identifies the completed topology with the topology of its extended defining ideal. These steps support the subsequent local-ring specialization.

6 Locality, universal maps and the edge cases

For the manuscript specialization, supply a field k, an O-algebra structure on k, residue algebra maps $R_i \to k$, and surjectivity of $O \to k$. The defining ideals are their kernels. `PaperXII.FiniteTensorLocality` constructs $S/\mathfrak{a} \cong k$, proves $\mathfrak{a}$ maximal, and derives finite generation from Noetherianity of O and each R_i. It then proves locality and the maximal-ideal-adic topology of the actual completion. These residue maps are explicit mathematical data, not an implicit assertion that all residue fields somehow agree.

Source: `PaperXII/FiniteTensorLocality.lean`, lines 100–138.

```
    (fgO : mO.FG) (fg : ∀ i, (m i).FG) : (definingIdeal (O := O) R mO m).FG := by
  exact Submodule.FG.sup (fgO.map _) (Submodule.fg_iSup _ (fun i ↦ (fg i).map _))

/-- The completed finite tensor product is local under the actual common-residue data. -/
theorem completedTensor_isLocalRing (hO : Surjective (algebraMap O k))
    (fgO : (RingHom.ker (algebraMap O k)).FG)
    (fg : ∀ i, (RingHom.ker (res i)).FG) :
    IsLocalRing (CompletedTensor (O := O) R (RingHom.ker (algebraMap O k))
      (fun i ↦ RingHom.ker (res i))) := by
  have := residueIdeal_isMaximal R res hO
  exact AdicModel.completed_isLocalRing _ (definingIdeal_fg R _ _ fgO fg)

/-- The manuscript's Noetherian hypotheses discharge finite generation internally. -/
theorem completedTensor_isLocalRing_noetherian [IsNoetherianRing O]
    [∀ i, IsNoetherianRing (R i)] (hO : Surjective (algebraMap O k)) :
    IsLocalRing (CompletedTensor (O := O) R (RingHom.ker (algebraMap O k))
      (fun i ↦ RingHom.ker (res i))) :=
  completedTensor_isLocalRing R res hO (Ideal.fg_of_isNoetherianRing _)
    (fun _i ↦ Ideal.fg_of_isNoetherianRing _)

/-- Proper quotients retain locality. The zero-ring case remains covered by the ring equivalence. -/
```

```
theorem completed_tensor_quotient_isLocalRing [IsNoetherianRing O]
    [∀ i, IsNoetherianRing (R i)] (hO : Surjective (algebraMap O k))
    (J : ∀ i, Ideal (R i))
    [Nontrivial (CompletedTensor (O := O) R (RingHom.ker (algebraMap O k))
      (fun i ↦ RingHom.ker (res i)) / closedConditions R (RingHom.ker (algebraMap O k))
        (fun i ↦ RingHom.ker (res i)) J)] :
    IsLocalRing (CompletedTensor (O := O) R (RingHom.ker (algebraMap O k))
      (fun i ↦ RingHom.ker (res i)) / closedConditions R (RingHom.ker (algebraMap O k))
        (fun i ↦ RingHom.ker (res i)) J) := by
  have := completedTensor_isLocalRing_noetherian R res hO
  exact IsLocalRing.of_surjective' (Ideal.Quotient.mk _) Ideal.Quotient.mk_surjective

/-- The completion topology is the topology of its actual extended defining ideal. -/
theorem completedTensor_isAdic (mO : Ideal O) (m : ∀ i, Ideal (R i))
    (fgO : mO.FG) (fg : ∀ i, (m i).FG) :
    IsAdic ((definingIdeal (O := O) R mO m).map (tensorEmbed R mO m).toRingHom) :=
  AdicModel.completed_isAdic _ (definingIdeal_fg R mO m fgO fg)
```

The completed universal map also has explicit topological control. Its target P is a complete Hausdorff uniform topological ring with O-algebra structure, m_P-adic topology, and factor maps carrying both the scalar and factor defining ideals into m_P. The raw family map is proved continuous, extended to the actual completion, and shown unique by density and finite tensor extensionality. The final existence-and-uniqueness theorem is an output of that construction.

Source: `PaperXII/CompletedTensorUniversal.lean`*, lines 15–23.*

```
  [IsTopologicalRing P] [CompleteSpace P] [T2Space P]
variable (mP : Ideal P) (hP : IsAdic mP) (f : ∀ i, R i →ₐ[O] P)
  (hO : mO.map (algebraMap O P) ≤ mP) (hf : ∀ i, (m i).map (f i).toRingHom ≤ mP)

omit [UniformSpace P] [IsUniformAddGroup P] [IsTopologicalRing P] [CompleteSpace P] [T2Space P] in
include hO hf in
/-- Local ideal control extends from the factors to the actual algebraic tensor. -/
theorem liftFamily_definingIdeal : (definingIdeal R mO m).map (liftFamily R f).toRingHom ≤ mP := by
  rw [definingIdeal, Ideal.map_sup, sup_le_iff, Ideal.map_map]
```

Source: `PaperXII/CompletedTensorUniversal.lean`*, lines 83–99.*

```
  exact DFunLike.congr_fun hr (AdicModel.rawEquiv _ _ r)

include hP hO hf in
/-- Existence and uniqueness of the continuous algebra map specified by the local factors. -/
theorem exists_unique_completedLift :
    ∃! g : CompletedTensor R mO m →ₐ[O] P,
      Continuous g ∧ ∀ i, g.comp (completedInclude R mO m i) = f i := by
  refine ⟨completedLift R mO m mP hP f hO hf, ⟨completedLift_continuous .., ?_⟩, ?_⟩
  · intro i; ext x; exact completedLift_factor R mO m mP hP f hO hf i x
  · intro g hg
    apply completedTensor_hom_ext R mO m hg.1 (completedLift_continuous ..)
    intro i
    rw [hg.2 i]
    ext x
    exact (completedLift_factor R mO m mP hP f hO hf i x).symm

end PaperXII.FiniteTensor
```

The empty family retains m_O's adic topology. Zero conditions give the identity comparison. A unit condition produces the zero ring and still satisfies the ring comparison. Locality of a quotient is claimed

only with an explicit nontriviality premise; the zero ring is not assigned the manuscript's nonzero local ring and residue field. Source boundary checks cover these cases separately.

7 Claim 3.3: completed global base change and its points

Let A be the constructed completed local tensor, let J be its closed condition ideal, and let C be the independently completed tensor of R_i/I_i. The A-algebra structure on C is built from the proved Claim 3.1 equivalence $A/J \cong C$. `quotientIdentification_ideal` proves that the transported defining ideal is the one defining the actual condition tensor topology.

Given the supplied global ring B and its structural A-algebra map, the global condition ring is the actual completion of $B \otimes_A C$. The transport code first constructs the tensor with A/J, then transports along the proved quotient identification. It is not an assumed pushout with a supplied point-factorization field. For complete B, the actual ring is bicontinuously identified with $B/\overline{JB}$, and its canonical global map is surjective.

The reusable ring-point theorem assumes finite generation of the scalar and factor defining ideals, B's declared adic topology, and an ideal bound for the structural map. Its target is a complete Hausdorff uniform topological ring. It proves that a unique continuous point factor exists precisely when each original local restriction kills I_i.

Source: `PaperXII/CompletedLocalConditions.lean`, *lines 17–38.*

```
17 variable (R : ι → Type*) [∀ i, CommRing (R i)] [∀ i, Algebra O (R i)]
18 variable (mO : Ideal O) (m : ∀ i, Ideal (R i)) (I : ∀ i, Ideal (R i))
19
20 abbrev LocalTensor := FiniteTensor.CompletedTensor R mO m
21 abbrev Conditions := FiniteTensor.CompletedTensor (fun i ↦ R i / I i) mO
22   (fun i ↦ (m i).map (Ideal.Quotient.mk (I i)))
23 abbrev closedIdeal := FiniteTensor.closedConditions R mO m I
24 abbrev localIdeal := (FiniteTensor.definingIdeal R mO m).map
25   (FiniteTensor.tensorEmbed R mO m).toRingHom
26
27 /-- The actual condition tensor receives the canonical local quotient action. -/
28 instance conditionAlgebra : Algebra (LocalTensor R mO m) (Conditions R mO m I) :=
29   ((FiniteTensor.completedQuotientEquiv R mO m I).toRingHom.comp
30     (Ideal.Quotient.mk (closedIdeal R mO m I))).toAlgebra
31
32 /-- Claim 3.1 upgraded to the local-tensor coefficient algebra. -/
33 def quotientIdentification : (LocalTensor R mO m / closedIdeal R mO m I) ≃ₐ[LocalTensor R mO m]
34     Conditions R mO m I where
35   __ := FiniteTensor.completedQuotientEquiv R mO m I
36   commutes' _ := rfl
37
38 /-- The transported ideal is exactly the one defining the actual condition tensor topology. -/
```

Source: `PaperXII/CompletedLocalConditions.lean`, *lines 66–100.*

```
66
67 /-- The defining construction provides genuine completeness and separation. -/
68 theorem globalRing_complete : CompleteSpace (GlobalRing R mO m I mB) ∧
69     T2Space (GlobalRing R mO m I mB) := ⟨inferInstance, inferInstance⟩
70
71 variable [UniformSpace B] [IsUniformAddGroup B] [IsTopologicalRing B]
72 variable {P : Type*} [CommRing P] [UniformSpace P] [IsUniformAddGroup P]
73   [IsTopologicalRing P] [CompleteSpace P] [T2Space P]
```

```
74
75 omit [IsUniformAddGroup B] in
76 /-- Full Claim 3.3: the named completed base change imposes exactly all local conditions. -/
77 theorem global_base_change (fgO : mO.FG) (fg : ∀ i, (m i).FG) (hB : IsAdic mB)
78     (h : (localIdeal R mO m).map (algebraMap (LocalTensor R mO m) B) ≤ mB)
79     (p : B →+* P) (hp : Continuous p) :
80     (∃! g : GlobalRing R mO m I mB →+* P,
81       Continuous g ∧ g.comp (globalMap R mO m I mB) = p) ↔
82         ∀ i, I i ≤ RingHom.ker ((p.comp (algebraMap (LocalTensor R mO m) B)).comp
83           (FiniteTensor.completedInclude R mO m i).toRingHom) := by
84   have hp' : Continuous (p.comp (AdicModel.rawEquiv B mB).toRingHom) :=
85     hp.comp (AdicModel.continuous_of_map_le (AdicModel.carrier_isAdic B mB) hB
86       (AdicModel.rawEquiv B mB).toRingHom (by
87         change mB.map (RingHom.id B) ≤ mB
88         simp))
89   refine (CompletedBaseChange.transported_exists_unique_factor_iff (localIdeal R mO m) mB
90     (closedIdeal R mO m I) (quotientIdentification R mO m I) h p hp').trans ?_
91   change FiniteTensor.closedConditions R mO m I ≤ RingHom.ker
92     (p.comp (algebraMap (LocalTensor R mO m) B)) ↔ _
93   rw [FiniteTensor.closedConditions_eq]
94   apply LocalQuotients.closed_conditions_iff R
95     (fun i ↦ (FiniteTensor.completedInclude R mO m i).toRingHom) I
96   exact hp.comp (AdicModel.continuous_of_map_le
97     (FiniteTensor.completedTensor_isAdic R mO m fgO fg) hB
98       (algebraMap (LocalTensor R mO m) B) h)
99
100 end PaperXII.CompletedLocalConditions
```

The manuscript capstone uses Noetherianity and common residue maps to supply finite generation. It requires B local and the structural algebra map local; the resulting maximal ideal bound is proved internally. Locality is an essential premise here. A bare continuous ring map is not the full printed capstone's input. The final traditional manuscript exposes these inherited conventions directly in Claim 3.3 and spells out the point category rather than relying on conventions elsewhere.

Source: `PaperXII/CompletedPaperCapstone.lean`*, lines 17–69.*

```
17 variable (R : ι → Type*) [∀ i, CommRing (R i)] [∀ i, Algebra O (R i)]
18 variable (res : ∀ i, R i →ₐ[O] k) (hO : Function.Surjective (algebraMap O k))
19 variable [IsNoetherianRing O] [∀ i, IsNoetherianRing (R i)]
20
21 include hO in
22 /-- The ideal defining the actual local tensor topology is its maximal ideal. -/
23 theorem localIdeal_eq_maximalIdeal :
24     haveI := FiniteTensor.completedTensor_isLocalRing_noetherian R res hO
25     localIdeal R (RingHom.ker (algebraMap O k)) (fun i ↦ RingHom.ker (res i)) =
26       IsLocalRing.maximalIdeal (LocalTensor R (RingHom.ker (algebraMap O k))
27         (fun i ↦ RingHom.ker (res i))) := by
28   have hm := FiniteTensor.residueIdeal_isMaximal R res hO
29   exact (AdicModel.completed_maximalIdeal _ (FiniteTensor.definingIdeal_fg R _ _
30     (Ideal.fg_of_isNoetherianRing _) (fun _i ↦ Ideal.fg_of_isNoetherianRing _))).symm
31
32 variable (I : ∀ i, Ideal (R i))
33 variable {B : Type*} [CommRing B]
34   [Algebra (LocalTensor R (RingHom.ker (algebraMap O k)) (fun i ↦ RingHom.ker (res i))) B]
35   [IsLocalRing B]
36   [IsLocalHom (algebraMap (LocalTensor R (RingHom.ker (algebraMap O k))
37     (fun i ↦ RingHom.ker (res i))) B)]
38
39 include hO in
40 /-- A local global coefficient map automatically respects the required adic ideals. -/
41 theorem local_map_ideal_bound :
42     (localIdeal R (RingHom.ker (algebraMap O k)) (fun i ↦ RingHom.ker (res i))).map
43       (algebraMap (LocalTensor R (RingHom.ker (algebraMap O k))
```

```
        (fun i ↦ RingHom.ker (res i))) B) ≤ IsLocalRing.maximalIdeal B := by
  have := FiniteTensor.completedTensor_isLocalRing_noetherian R res hO
  rw [localIdeal_eq_maximalIdeal R res hO]
  exact IsLocalRing.map_maximalIdeal_le _

variable [UniformSpace B] [IsUniformAddGroup B] [IsTopologicalRing B]
variable {P : Type*} [CommRing P] [UniformSpace P] [IsUniformAddGroup P]
  [IsTopologicalRing P] [CompleteSpace P] [T2Space P]

omit [IsUniformAddGroup B] in
include hO in
/-- Claim 3.3 at the manuscript's Noetherian and local-map boundary. -/
theorem global_base_change_noetherian (hB : IsAdic (IsLocalRing.maximalIdeal B))
    (p : B →+* P) (hp : Continuous p) :
    (∃! g : GlobalRing R (RingHom.ker (algebraMap O k))
      (fun i ↦ RingHom.ker (res i)) I (IsLocalRing.maximalIdeal B) →+* P,
      Continuous g ∧ g.comp (globalMap R (RingHom.ker (algebraMap O k))
        (fun i ↦ RingHom.ker (res i)) I (IsLocalRing.maximalIdeal B)) = p) ↔
      ∀ i, I i ≤ RingHom.ker ((p.comp (algebraMap (LocalTensor R
        (RingHom.ker (algebraMap O k)) (fun i ↦ RingHom.ker (res i))) B)).comp
          (FiniteTensor.completedInclude R (RingHom.ker (algebraMap O k))
            (fun i ↦ RingHom.ker (res i)) i).toRingHom) :=
  global_base_change R _ _ I _ (Ideal.fg_of_isNoetherianRing _)
    (fun _i ↦ Ideal.fg_of_isNoetherianRing _) hB (local_map_ideal_bound R res hO) p hp

end PaperXII.CompletedLocalConditions
```

For continuous local O-algebra points, the production point theorem additionally has B complete, actual O-algebra structures on B and P, and an original point which is an O-algebra map reflecting units. The completed global coefficient action is the actual composite through B. Surjectivity of the global map proves that the unique ring factor reflects units; equality on coefficients proves it is an O-algebra map. Naturality follows from equality after the surjective global map.

Source: `PaperXII/CompletedPointCategory.lean`, *lines 49–92.*

```
  [IsUniformAddGroup P] [IsTopologicalRing P] [CompleteSpace P] [T2Space P]

/-- Claim 3.3 as an equivalence of continuous local algebra points. -/
theorem local_algebra_point_iff (fgO : mO.FG) (fg : ∀ i, (m i).FG)
    (hB : IsAdic mB)
    (h : (localIdeal R mO m).map (algebraMap (LocalTensor R mO m) B) ≤ mB)
    (p : B →ₐ[O] P) [IsLocalHom p.toRingHom] (hp : Continuous p) :
    (∃! g : GlobalRing R mO m I mB →ₐ[O] P,
      Continuous g ∧ IsLocalHom g.toRingHom ∧ g.comp (globalAlgMap R mO m I mB) = p) ↔
        ∀ i, I i ≤ RingHom.ker ((p.toRingHom.comp (algebraMap (LocalTensor R mO m) B)).comp
          (FiniteTensor.completedInclude R mO m i).toRingHom) := by
  constructor
  · rintro ⟨g, ⟨hg, _, hgp⟩, _⟩
    apply (global_base_change R mO m I mB fgO fg hB h p.toRingHom hp).mp
    refine ⟨g.toRingHom, ⟨hg, ?_⟩, ?_⟩
    · exact congrArg AlgHom.toRingHom hgp
    · intro g' hg'
      ext x
      obtain ⟨b, rfl⟩ := globalMap_surjective R mO m I mB hB h x
      exact (DFunLike.congr_fun hg'.2 b).trans
        (DFunLike.congr_fun (congrArg AlgHom.toRingHom hgp) b).symm
  · intro hI
    obtain ⟨g, ⟨hg, hgp⟩, hu⟩ :=
      (global_base_change R mO m I mB fgO fg hB h p.toRingHom hp).mpr hI
    let ga : GlobalRing R mO m I mB →ₐ[O] P :=
      { __ := g
        commutes' := fun a ↦ (DFunLike.congr_fun hgp (algebraMap O B a)).trans (p.commutes a) }
    refine ⟨ga, ⟨hg, point_factor_isLocal R mO m I mB hB h p.toRingHom g hgp, ?_⟩, ?_⟩
```

```
    · exact AlgHom.ext (fun b ↦ DFunLike.congr_fun hgp b)
    · intro g' hg'
      apply AlgHom.ext
      intro x
      exact DFunLike.congr_fun (hu g'.toRingHom ⟨hg'.1, congrArg AlgHom.toRingHom hg'.2.2⟩) x

omit [UniformSpace P] [IsUniformAddGroup P] [IsTopologicalRing P] [CompleteSpace P] [T2Space P] in
/-- The point factorization is natural under coefficient-preserving target maps. -/
theorem algebra_factor_natural {Q : Type*} [CommRing Q] [Algebra O Q]
    (hB : IsAdic mB)
    (h : (localIdeal R mO m).map (algebraMap (LocalTensor R mO m) B) ≤ mB)
    (p : B →ₐ[O] P) (u : P →ₐ[O] Q)
    (g : GlobalRing R mO m I mB →ₐ[O] P)
    (g' : GlobalRing R mO m I mB →ₐ[O] Q)
    (hg : g.comp (globalAlgMap R mO m I mB) = p)
    (hg' : g'.comp (globalAlgMap R mO m I mB) = u.comp p) :
```

The ring-point result, local algebra-point result and Noetherian/local specialization are distinct declarations and must be read together. A local deformation test point is distinct from a characteristic-zero field specialization, where a coefficient uniformizer can be a unit; continuity of the specialized representation is separately required by realization. The supplementary `claim33` consumes the local point statements using manuscript-level hypotheses, without an extra finite-generation or ideal-bound hypothesis. It does not prove arithmetic representability of a deformation functor. Representability, universal arithmetic deformation representations, and existence of the local quotient rings remain supplied inputs.

8 Ordinary support, patching and intrinsic Fitting ideals

For a finite module, production support is Mathlib's actual localization support on the prime spectrum. Lean proves

$$\operatorname{Supp}_R M = V(\operatorname{Ann}_R M).$$

Under the Noetherian hypotheses, near faithfulness is equivalent to full support. Quotient support is obtained on corresponding primes using actual quotient modules and localization/Nakayama arguments. Ordinary patching then uses supplied coefficient and module control isomorphisms to descend full support from the patched finite module; finite generation and finite-level full support are outputs.

An arithmetic patching theorem supplying the patched ring, module, control maps and full patched support remains external. The finite-level R-action on the same carrier must be restriction of its Hecke action through the supplied comparison, as expressed by the ambient scalar-tower hypotheses; derived carriers require the corresponding transport of the actual complex. Two unrelated actions on a carrier do not supply this compatibility. Freeness over an auxiliary algebra is insufficient for full deformation support; the power-series counterexample demonstrates the missing implication. A ring isomorphism $R \cong \mathbb{T}$ transports existing support. A surjection alone can erase a component, as in the actual node $k[[x, y]]/(xy)$ followed by quotienting by (x).

The ten-module Fitting block constructs the determinantal ideals of a finite presentation and proves their invariance under changes of presentation, generator and relation counts. Its intrinsic definition chooses a presentation using classical choice and then proves equality with the minors of every actual finite presentation. No presentation-independence theorem is supplied as a hypothesis.

Source: `PaperXII/IntrinsicFitting.lean`, lines 8–39.

```
variable (R : Type*) [CommRing R] (M : Type*) [AddCommGroup M] [Module R M]
```

```
/-- Actual finite free relations, generators, and a surjective quotient map. -/
structure FiniteMatrixPresentation where
  generators : ℕ
  relations : ℕ
  matrix : Matrix (Fin generators) (Fin relations) R
  quotient : (Fin generators → R) →ₗ[R] M
  presents : Presents matrix quotient

instance finiteMatrixPresentation_nonempty [Module.FinitePresentation R M] : Nonempty (FiniteMatrixPresentation R M) := by
  classical
  obtain ⟨n,m,f,g,hf,hgf⟩ := Module.FinitePresentation.exists_fin' R M
  refine ⟨⟨n,m,LinearMap.toMatrix' g,f,⟨hf,?_⟩⟩⟩
  simpa only [← Matrix.toLin'_apply',Matrix.toLin'_toMatrix'] using hgf.linearMap_ker_eq.symm

/-- A chosen finite presentation. Its choice is removed by `fittingIdeal_eq_presentation`. -/
def chosenPresentation [Module.FinitePresentation R M] : FiniteMatrixPresentation R M :=
  Classical.choice inferInstance

/-- The intrinsic j-th Fitting ideal of a finitely presented module. -/
def fittingIdeal [Module.FinitePresentation R M] (j : ℕ) : Ideal R :=
  presentationIdeal (chosenPresentation R M).matrix j

variable {R M}

/-- The intrinsic ideal equals the minors computed from any actual finite presentation. -/
theorem fittingIdeal_eq_presentation [Module.FinitePresentation R M]
    {m n : Type*} [Fintype m] [Fintype n]
    (A : Matrix m n R) (π : (m → R) →ₗ[R] M) (h : Presents A π) (j : ℕ) :
    fittingIdeal R M j = presentationIdeal A j :=
  presentationIdeal_independent _ _ _ _ (chosenPresentation R M).presents h j
```

The arbitrary base-change theorem transports each minor under the ring map. The support criterion and all higher strata are stated for finitely presented modules; Noetherianity plus finite generation in the manuscript supplies that premise. For a presentation with n generators,

$$\dim_{\kappa(\mathfrak{p})}(M \otimes \kappa(\mathfrak{p})) = n - \operatorname{rank} A(\mathfrak{p}).$$

For $1 \le r \le n$, the inequality $\dim \ge r$ is detected by minors of size $n - r + 1$. If $r > n$, the fiber locus is empty and $\mathrm{Fitt}_{r-1}(M) = R$. The final ordinary proof explicitly separates this boundary; the formal block already includes upper-index normalization and empty/zero-minor checks. Fitting strata count minimal fiber generators, not automorphic multiplicities.

9 Derived support uses actual perfect models

`PaperXII.PerfectDerivedSupport` does not encode a complex by a bare list of supports. Its `PerfectComplex` contains an actual Mathlib derived-category object, a bounded finite-projective cochain model, and an actual derived isomorphism from that model. Base change is computed degree by degree on the projective model. The code proves compatibility with actual derived isomorphisms, so support does not depend on the chosen model.

Source: `PaperXII/PerfectDerivedSupport.lean`, lines 23–53.

```
open PerfectModels

/-- An arbitrary perfect derived object with a bounded finite-projective model. -/
```

```
structure PerfectComplex (R : Type u) [CommRing R] where
  object : DerivedCategory (ModuleCat.{u} R)
  model : PerfectModel R
  modelIso : model.derivedObject ≅ object

variable {R S T : Type u} [CommRing R] [CommRing S] [CommRing T]

/-- A bounded projective model determines a perfect derived object. -/
def ofModel (P : PerfectModel R) : PerfectComplex R :=
  (P.derivedObject, P, Iso.refl _)

/-- Derived scalar extension, computed on the finite-projective model. -/
def PerfectComplex.baseChange (P : PerfectComplex R) (f : R →+* S) : PerfectComplex S :=
  ofModel (P.model.baseChange f)

/-- Every actual derived isomorphism induces one after arbitrary scalar extension. -/
def baseChangeIso (f : R →+* S) {P Q : PerfectComplex R}
    (e : P.object ≅ Q.object) : (P.baseChange f).object ≅ (Q.baseChange f).object :=
  baseChange_derivedIso f P.model Q.model (P.modelIso <<>> e <<>> Q.modelIso.symm)

/-- Successive derived scalar extensions agree with extension by the composite. -/
def baseChangeCompIso (P : PerfectComplex R) (f : R →+* S) (g : S →+* T) :
    ((P.baseChange f).baseChange g).object ≅ (P.baseChange (g.comp f)).object :=
  ComplexSupport.fiberCompIso f g P.model.complex

/-- Identity scalar extension recovers the arbitrary represented derived object. -/
def baseChangeIdIso (P : PerfectComplex R) :
```

Derived support is the set of primes with nonzero residue-field derived fiber. Scalar extension has spectral inverse-image support; the quotient identity is expressed as an image in the original spectrum. It includes the unit-ideal quotient. Field-valued points are handled at their actual kernel primes and then by faithful flatness of the field extension, without imposing a closed-point restriction.

Source: `PaperXII/PerfectDerivedSupport.lean`, *lines 56–96.*

```

/-- The residue-field definition of derived support in the original manuscript. -/
def support (P : PerfectComplex R) : Set (PrimeSpectrum R) :=
  ComplexSupport.support P.model.complex

/-- Full support of a perfect derived object. -/
def FullSupport (P : PerfectComplex R) : Prop := support P = Set.univ

/-- Derived support is independent of both model and representative. -/
theorem support_eq_of_iso {P Q : PerfectComplex R} (e : P.object ≅ Q.object) :
    support P = support Q := by
  ext p
  exact not_congr (baseChangeIso (algebraMap R p.asIdeal.ResidueField) e).isZero_iff

/-- The field-valued criterion uses the exact prime kernel of the point. -/
theorem nonzero_fiber_iff_support {K : Type u} [Field K] (P : PerfectComplex R)
    (x : R →+* K) :
    ¬IsZero (P.baseChange x).object ↔ FieldFibers.pointPrime x ∈ support P :=
  ComplexSupport.nonzero_fiber_iff_support x P.model.complex

/-- Derived support under every scalar map is the actual spectral inverse image. -/
theorem support_baseChange (P : PerfectComplex R) (f : R →+* S) :
    support (P.baseChange f) = PrimeSpectrum.comap f ⁻¹' support P :=
  ComplexSupport.support_baseChange f P.model.complex

/-- Original Theorem 6.16, in literal ambient-spectrum form for the actual quotient. -/
theorem derived_quotient_support (P : PerfectComplex R) (I : Ideal R) :
    PrimeSpectrum.comap (Ideal.Quotient.mk I) '' support (P.baseChange (Ideal.Quotient.mk I)) =
```

```
      support P ∩ PrimeSpectrum.zeroLocus I :=
  ComplexSupport.image_support_quotient I P.model.complex

/-- Support transports along a ring isomorphism by its actual map on spectra. -/
theorem support_ringEquiv (P : PerfectComplex R) (e : R ≃+* S) :
    PrimeSpectrum.comap e.toRingHom '' support (P.baseChange e.toRingHom) = support P := by
  rw [support_baseChange, Set.image_preimage_eq_inter_range]
  have hsurj : Function.Surjective (PrimeSpectrum.comap e.toRingHom) :=
    (PrimeSpectrum.isHomeomorph_comap_of_bijective e.bijective).surjective
  rw [Set.range_eq_univ.mpr hsurj, Set.inter_univ]

/-- Full support survives arbitrary derived scalar extension. -/
theorem fullSupport_baseChange (P : PerfectComplex R) (f : R →+* S)
```

A nonzero field fiber yields some nonzero actual derived cohomology object in an integer degree. The bounded projective model proves vanishing outside its bounds. The induced Hecke action comes from multiplication on the original cochain model, through scalar extension and passage to derived cohomology, and equals the specialized scalar character.

These are formal algebraic conclusions. They neither choose a classical weight nor establish classicality. Revised Claim 9.6 requires the classicality range to contain the model interval; revised 12.4 permits applicability at the selected eigensystem itself. The original unrestricted versions remain partial in their original ledger. The cited external cohomological description of derived support is not needed by this production residue-field support proof.

10 Artin packets, residual augmentation and finite adequacy

Hecke packet construction starts with a supplied canonical ring decomposition into actual localizations. The input is `ArtinData.DecompositionR`, an R-algebra equivalence whose coordinates are proved to be the canonical localization maps. It supplies no module direct-sum decomposition and no module localization equivalence; those are constructed subsequently.

Source: `PaperXII/ArtinData.lean`*, lines 17–41.*

```

/-- The canonical ring decomposition supplied by external Fact 8.1. -/
abbrev Decomposition (R : Type*) [CommRing R] :=
  R ≃ₐ[R] (∀ m : MaximalSpectrum R, Localization.AtPrime m.asIdeal)

/-- The actual central idempotent indexed by a maximal ideal. -/
noncomputable def idempotent (D : Decomposition R) (m : MaximalSpectrum R) : R := by
  classical
  exact D.symm (Pi.single m 1)

/-- Every coordinate is the canonical localization map, by R-algebra linearity. -/
theorem coordinate (D : Decomposition R) (r : R) (m : MaximalSpectrum R) :
    D r m = algebraMap R (Localization.AtPrime m.asIdeal) r := by
  have h := D.commutes r
  exact congrFun h m

/-- The canonical coordinate maps determine the decomposition uniquely. -/
theorem decomposition_unique (D₁ D₂ : Decomposition R) : D₁ = D₂ := by
  ext r m
  rw [coordinate, coordinate]

/-- The coordinate-vector identity determines the idempotent uniquely. -/
theorem map_idempotent [DecidableEq (MaximalSpectrum R)]
    (D : Decomposition R) (m : MaximalSpectrum R) :
```

```
41     D (idempotent D m) = Pi.single m 1 := by
```

From this input, Lean constructs orthogonal idempotents, their packet submodules, the direct sum, and the packet/localized-module equivalence. It verifies the map $v \mapsto v/1$. The selected local factor's nilpotent maximal ideal and an invertible generator on every other factor identify the packet with the simultaneous generalized eigenspace for one positive exponent. This does not assume semisimplicity. In characteristic zero, the projector trace is the dimension of its image cast into the field, and nonzero trace detects occupancy. Positive-characteristic trace cancellation remains a real boundary.

Auxiliary group-algebra transfer is constructed on the actual group basis and commutes with actual augmentation. Coinvariant recovery proves a quotient by all balancing relations and identifies it with the ordinary tensor product when the appropriate algebra is commutative. The right-module treatment handles arbitrary groups, including nonabelian two-groups; a supplied free basis descends to a coefficient basis with the same index set, including rank zero.

For a product of cyclic p-groups in characteristic p, the full multivariable truncated-polynomial algebra equivalence is constructed and its augmentation generators identified. The no-nontrivial-idempotent result covers every finite p-group, including nonabelian ones, by an orbit-span/fixed-vector argument over the prime field. The dyadic code proves that rational sign projectors are not integral and constructs $\mathbb{F}_2[C_2] \cong \mathbb{F}_2[X]/(X^2)$. None of this supplies dyadic arithmetic image, local geometry or classicality hypotheses.

The S_3 example over $\mathbb{F}_7$ is a Lean certificate for one finite representation. The modules construct the standard representation from the permutation module, prove absolute irreducibility, identify its trace-zero adjoint with sign plus standard, and connect averaging to actual first group cohomology. Canonical generalized-eigenspace projectors detect every nonzero adjoint constituent. Independent finite computations can check the printed matrices and trace pairings; their scope remains separate from the formal certificate and from arithmetic adequacy applications. The adequacy convention remains a cited input.

11 Occupancy, stable cores and adic completion of classes

In one finite-dimensional carrier, nested idempotent ranges stabilize; nonzero ranges at every level yield a nonzero intersection. In changing carriers, the inverse limit is controlled by the actual intersections of transition images, not merely by nonzero individual levels. `PaperXII.StableCores` uses a `LinearSystem` with actual linear maps and identity/composition laws. It proves that transition images are antitone, that finite-dimensionality makes each stable core an eventual image, and that the core transition is surjective. Recursive lifting then produces a compatible nonzero vector above a chosen nonzero core coordinate.

Source: `PaperXII/StableCores.lean`, *lines 6–38.*

```
6  /-- A countable inverse system, with the ordinary identity and composition laws. -/
7  structure LinearSystem (K : Type*) [Field K] (V : ℕ → Type*)
8      [∀ n, AddCommGroup (V n)] [∀ n, Module K (V n)] where
9    map : ∀ n m, m ≤ n → V n →ₗ[K] V m
10   map_self : ∀ n, map n n le_rfl = LinearMap.id
11   map_comp : ∀ (a b c) (hcb : c ≤ b) (hba : b ≤ a),
12     (map b c hcb).comp (map a b hba) = map a c (hcb.trans hba)
13
14 variable {K : Type*} [Field K] {V : ℕ → Type*}
15     [∀ n, AddCommGroup (V n)] [∀ n, Module K (V n)]
16
17 /-- Padded image family: indices before the target level contribute the whole space. -/
```

```
def imageAt (D : LinearSystem K V) (m n : ℕ) : Submodule K (V m) :=
  iInf h : m ≤ n, LinearMap.range (D.map n m h)

/-- The actual intersection of the images from every later level. -/
def stableCore (D : LinearSystem K V) (m : ℕ) : Submodule K (V m) :=
  iInf n, imageAt D m n

/-- Membership is precisely membership in every later transition image. -/
theorem mem_stableCore_iff (D : LinearSystem K V) (m : ℕ) (v : V m) :
    v ∈ stableCore D m ↔ ∀ n (h : m ≤ n), ∃ w, D.map n m h w = v := by
  simp only [stableCore, imageAt, Submodule.mem_iInf, LinearMap.mem_range]

/-- A valid later index has its literal transition range as padded image. -/
theorem imageAt_eq_range (D : LinearSystem K V) {m n : ℕ} (h : m ≤ n) :
    imageAt D m n = LinearMap.range (D.map n m h) := by
  simp [imageAt, h]

/-- Composition makes the transition images descend. -/
theorem imageAt_antitone (D : LinearSystem K V) (m : ℕ) : Antitone (imageAt D m) := by
  intro a b hab
  apply le_iInf
```

Finite companion forests give branches when every finite level is nonempty. An empty level obstructs complete realizations whose actual finite reductions belong to those declared levels; it does not rule out representations excluded by an arbitrary admissibility restriction. The production `empty_level_no_realization` explicitly takes reduction maps from the supplied realization type into every forest level. Forests do not alone produce continuous integral representations with every arithmetic condition. Revised Claim 10.5 makes the missing effectivity inputs explicit: complete separated coefficient rings for finitely generated ideals, finite residue rings, finite occupied classes of actual continuous quotient representations, reduction compatibility, continuous coefficient maps, integral Frobenius polynomials with matching scalar changes, and declared polynomial ideal conditions valid in every admissible frame.

Finite frames turn finite admissible conjugacy classes into finite sets of actual representatives. Compactness selects strictly compatible representatives without assuming transition surjectivity or lifting a prescribed conjugating matrix. Entrywise reconstruction, including inverse entries, constructs the actual matrix-group inverse limit. Continuity is derived from finite quotient continuity and the adic topology. Separatedness upgrades compatible polynomial and subgroup-triviality congruences to exact integral equalities. Evaluation then kills each declared polynomial ideal and factors through its actual quotient.

Source: `PaperXII/ConditionalCompletion.lean`*, lines 8–54.*

```

open ContinuousMonoidHom Matrix
open AdicMatrixGroups AdicCompanions ContinuousClasses ClassCompanions AdicSpines
open AdicLocalConditions LocalQuotients PaperX.ClassInvariants

variable {Γ ι : Type} [Group Γ] [TopologicalSpace Γ] [Fintype ι] [DecidableEq ι]

/-- Positive finite class forests realize the prescribed common spine and every
declared polynomial quotient condition on the actual representation entries. -/
theorem simultaneous_companion_completion
    {Λ Ω L V : Type} [Field L] (O K : Λ → Type)
    [∀ l, CommRing (O l)] [∀ l, Field (K l)] [∀ l, TopologicalSpace (K l)]
    (J : ∀ l, Ideal (O l)) [∀ l, IsAdicComplete (J l) (O l)]
    (hJ : ∀ l, (J l).FG) [∀ l, Finite (O l ⧸ J l)]
    (C : ∀ l n, LevelClass (Γ := Γ) (ι := ι) (J l) n → Prop)
    (hC : ∀ l n c, C l (n + 1) c → C l n (classStep (J l) n c))
    (finiteC : ∀ l n, 0 < n → Finite {c // C l n c})
    (hc : ∀ l n, 0 < n → ∀ c, C l n c → ClassContinuous c)
```

```
    (hne : ∀ l n, 0 < n → ∃ c, C l n c)
    (root : ∀ l, LevelClass (Γ := Γ) (ι := ι) (J l) 1)
    (hroot : ∀ l c, C l 1 c → c = root l)
    (inc : ∀ l, O l →+* K l)
    (hinc : ∀ l, @Continuous (O l) (K l) (J l).adicTopology inferInstance (inc l))
    (emb : ∀ l, L →+* K l) (frob : Ω → Γ) (good : Λ → Ω → Prop)
    (P : Ω → Polynomial L) (localP : ∀ l, Ω → Polynomial (O l))
    (hmatch : ∀ l v, good l v → (localP l v).map (inc l) = (P v).map (emb l))
    (hspec : ∀ l n, 0 < n → ∀ c, C l n c → ∀ v, good l v →
      classCharpoly c (frob v) = (localP l v).map (quotientMap (J l) n))
    (H : Λ → Subgroup Γ)
    (hH : ∀ l n, 0 < n → ∀ c, C l n c → ClassTrivial (H l) c)
    (relations : ∀ l, V → Ideal (MvPolynomial (Coordinates (Γ := Γ) (ι := ι)) (O l)))
    (hrelations : ∀ l v n, 0 < n → ∀ σ, Framed (J l) (C l) n σ →
      relations l v ≤ RingHom.ker (finitePoint (J l) n σ)) :
    ∃ ρ : ∀ l, AdicRealization (J l) (Framed (J l) (C l)),
      (∀ l, classOfRep (reduceRep (J l) 1 (ρ l).val) = root l) ∧
      (∀ l v, good l v →
        (fieldRep (J l) (inc l) (hinc l) (ρ l).val (frob v)).val.charpoly =
          (P v).map (emb l)) ∧
      (∀ l g, g ∈ H l → fieldRep (J l) (inc l) (hinc l) (ρ l).val g = 1) ∧
      ∀ l v, Factors (relations l v) (point (J l) (ρ l).val) := by
  obtain ⟨ρ, hrootρ, hpolyρ, hHρ⟩ := ClassSpines.simultaneous_class_spine
    O K J hJ C hC finiteC hc hne root hroot inc hinc emb frob good P localP
    hmatch hspec H hH
  exact ⟨ρ, hrootρ, hpolyρ, hHρ, fun l v ↦
    class_condition_preserved (J l) (C l) (relations l v) (ρ l) (hrelations l v)⟩

end PaperXII.ConditionalCompletion
```

The conclusion retains a residual conjugacy class, not a prescribed residual matrix frame; uniqueness up to integral conjugacy is not asserted. No arbitrary Hodge or Weil–Deligne condition is identified with a polynomial ideal without additional arithmetic work. Integral Frobenius lifts and occupancy at every precision are hypotheses, not outcomes of abstract compactness. A missing coefficient-place member is not supplied by propagation inside an existing compatible family.

12 Conditional realization and the five separate gates

The formal realization theorem compares actual matrix representations via an invertible linear intertwiner. It assumes a Hausdorff characteristic-zero topological field with continuous ring operations, continuous compared representations, a supplied semisimplification preserving characteristic polynomials, arithmetic density of the declared Frobenius conjugacy saturation, and a general algebraic Brauer–Nesbitt identification theorem over that coefficient field.

`PaperXII.DenseFrobenius` proves continuity of characteristic-polynomial coefficients and extends equality from the dense Frobenius conjugacy set to all group elements. The supplied Brauer–Nesbitt input then identifies the semisimple representations. Neither density nor the general Brauer–Nesbitt theorem is a newly introduced global Lean axiom: each is an explicit ordinary theorem argument, and the production result is conditional on it.

Source: `PaperXII/GaloisRealization.lean`, lines 13–53.

```
variable {E Γ Ω ι : Type*} [Field E] [CharZero E] [Group Γ]
    [TopologicalSpace E] [IsTopologicalRing E] [T2Space E]
    [TopologicalSpace Γ] [Fintype ι] [DecidableEq ι]

/-- The semisimplification of a continuous representation is identified with a
```

```
semisimple attached representation using only the general standard input. -/
theorem pointwise_semisimple_identification (bn : BrauerNesbitt E Γ ι)
    (ρ σ : Γ →ₜ* GL ι E) (s : Semisimplification ρ.toMonoidHom)
    (hσ : Representation.IsSemisimpleRepresentation (representation σ.toMonoidHom))
    (frob : Ω → Γ) (good : Ω → Prop)
    (hdense : Dense (conjugacySaturation frob good))
    (hpoly : ∀ v, good v → (ρ (frob v)).val.charpoly = (σ (frob v)).val.charpoly) :
    Isomorphic s.rep σ.toMonoidHom := by
  have hall := charpoly_eq_of_dense_frobenius ρ σ frob good hdense hpoly
  exact bn s.rep σ.toMonoidHom s.semisimple hσ
    (fun g ↦ (s.charpoly_eq g).trans (hall g))

/-- The irreducible case gives an isomorphism of the original representations. -/
theorem pointwise_irreducible_identification (bn : BrauerNesbitt E Γ ι)
    (ρ σ : Γ →ₜ* GL ι E)
    (hρ : Representation.IsIrreducible (representation ρ.toMonoidHom))
    (hσ : Representation.IsSemisimpleRepresentation (representation σ.toMonoidHom))
    (frob : Ω → Γ) (good : Ω → Prop)
    (hdense : Dense (conjugacySaturation frob good))
    (hpoly : ∀ v, good v → (ρ (frob v)).val.charpoly = (σ (frob v)).val.charpoly) :
    Isomorphic ρ.toMonoidHom σ.toMonoidHom :=
  bn ρ.toMonoidHom σ.toMonoidHom (irreducible_semisimple _ hρ) hσ
    (charpoly_eq_of_dense_frobenius ρ σ frob good hdense hpoly)

/-- Both printed conclusions for an arbitrary mapped Hecke polynomial spine. -/
theorem common_spine_identification {B : Type*} [CommRing B]
    (bn : BrauerNesbitt E Γ ι) (ρ σ : Γ →ₜ* GL ι E)
    (s : Semisimplification ρ.toMonoidHom)
    (hσ : Representation.IsSemisimpleRepresentation (representation σ.toMonoidHom))
    (frob : Ω → Γ) (good : Ω → Prop)
    (hdense : Dense (conjugacySaturation frob good)) (x : B →+* E)
    (P : Ω → Polynomial B)
    (hρP : ∀ v, good v → (ρ (frob v)).val.charpoly = (P v).map x)
    (hσP : ∀ v, good v → (σ (frob v)).val.charpoly = (P v).map x) :
    Isomorphic s.rep σ.toMonoidHom ∧
      (Representation.IsIrreducible (representation ρ.toMonoidHom) →
```

For a supplied universal representation, `specialize` is literally coefficient change into the chosen field. Its continuity is explicit. The two representations must have the same specialized Hecke polynomials on the same dense good set. The ordinary completion theorem has the universal specialization polynomial identity as a separate explicit premise; its classicality input supplies the attached representation's matching polynomial identity. The final traditional definition 12.1 spells out these inputs; a nonzero fiber and an attached representation alone would not identify the universal specialization.

Source: `PaperXII/GaloisRealization.lean`, lines 73–97.

```

/-- Continuity of the specialization under consideration is explicit. -/
def continuousSpecialization (ρR : Γ →* GL ι R) (x : R →+* E)
    (hx : Continuous (specialize ρR x)) : Γ →ₜ* GL ι E :=
  { __ := specialize ρR x, continuous_toFun := hx }

/-- Source-shaped pointwise realization: the compared representation is literally
x composed with the supplied universal representation. -/
theorem universal_pointwise_realization {T : Type*} [CommRing T]
    (bn : BrauerNesbitt E Γ ι) (ρR : Γ →* GL ι R) (x : R →+* E)
    (hx : Continuous (specialize ρR x))
    (s : Semisimplification (specialize ρR x))
    (σ : Γ →ₜ* GL ι E)
    (hσ : Representation.IsSemisimpleRepresentation (representation σ.toMonoidHom))
    (frob : Ω → Γ) (good : Ω → Prop)
    (hdense : Dense (conjugacySaturation frob good))
```

```
    (y : T →+* E) (P : Ω → Polynomial T)
    (hρP : ∀ v, good v → (specialize ρR x (frob v)).val.charpoly = (P v).map y)
    (hσP : ∀ v, good v → (σ (frob v)).val.charpoly = (P v).map y) :
    Isomorphic s.rep σ.toMonoidHom ∧
      (Representation.IsIrreducible (representation (specialize ρR x)) →
        Isomorphic (specialize ρR x) σ.toMonoidHom) :=
  common_spine_identification bn (continuousSpecialization ρR x hx) σ s hσ
    frob good hdense y P hρP hσP

```

Source: `PaperXII/OrdinaryCompletion.lean`, *lines 171–213.*

```
  exact supported_pointwise_realization eT x
    ((fullSupport_iff_localizations R H).mp hH _) bn ρR (hx x hxU) (s x hxU)
    frob good hdense P (hρP x hxU) occurs attached (classicality x hxU)

variable {A Pinf : Type*} [CommRing A] [IsNoetherianRing A] [Algebra A R]
    [AddCommGroup Pinf] [Module A Pinf] [Module.Finite A Pinf]
    [Module A H] [IsScalarTower A R H]

/-- Revised Theorem 12.2, including its actual nonzero tensor fiber, the supplied
classical object's occurrence, semisimplified identification, and irreducible
identification. Finite-level support and finite generation are derived from
the printed patched quotient datum; classicality is used only after this. -/
theorem ordinary_conditional_completion (I : Ideal A)
    (eR : (A / I) ≃ₐ[A] R)
    (eM : (Pinf / (I • ⊤ : Submodule A Pinf)) ≃ₗ[A] H)
    (eT : R ≃ₐ[R] T) (hP : FullSupport A Pinf) (x : R →+* E)
    (bn : BrauerNesbitt E Γ ι) (ρR : Γ →* GL ι R)
    (hx : Continuous (specialize ρR x)) (s : Semisimplification (specialize ρR x))
    (frob : Ω → Γ) (good : Ω → Prop)
    (hdense : Dense (conjugacySaturation frob good)) (P : Ω → Polynomial T)
    (hρP : ∀ v, good v → (specialize ρR x (frob v)).val.charpoly =
      (P v).map (transportedPoint eT.toRingEquiv x))
    (occurs : (R →+* E) → Auto → Prop) (attached : Auto → Γ →ₜ* GL ι E)
    (classicality :
      letI : Algebra T E := (transportedPoint eT.toRingEquiv x).toAlgebra
      Nontrivial (E ⊗[T] H) → ∃ π : Auto, occurs x π ∧
        Representation.IsSemisimpleRepresentation (representation (attached π).toMonoidHom) ∧
        ∀ v, good v → (attached π (frob v)).val.charpoly =
          (P v).map (transportedPoint eT.toRingEquiv x)) :
    letI : Algebra T E := (transportedPoint eT.toRingEquiv x).toAlgebra
    Nontrivial (H ⊗[T] E) ∧ ∃ π : Auto, occurs x π ∧
      Isomorphic s.rep (attached π).toMonoidHom ∧
      (Representation.IsIrreducible (representation (specialize ρR x)) →
        Isomorphic (specialize ρR x) (attached π).toMonoidHom) := by
  let : Module.Finite R H := finite_level_finite (R := R) I eM
  let : Module.Finite T H := Module.Finite.of_restrictScalars_finite R T H
  exact supported_pointwise_realization eT x
    ((fullSupport_iff_localizations R H).mp (finite_level_fullSupport I eR eM hP) _)
    bn ρR hx s frob good hdense P hρP occurs attached classicality

end Realization

end PaperXII.OrdinaryCompletion
```

The five-gate modules derive finite-level support from the actual patched quotient/control data. A general comparison can use an actual inclusion of support subsets under a ring map, with ordinary or derived support as appropriate. The compatible Hecke point y satisfying $y \circ f = x$ is separately supplied. Support preservation alone does not provide such a point over the prescribed coefficient field: $\mathbb{Q} \to \mathbb{Q}(i)$ preserves full support but admits no extension of the identity point to $\mathbb{Q}$.

The chosen nonzero tensor vector or cohomology vector carries all prescribed Hecke eigenvalues. Classicality is applied to this actual eigensystem, and supplies an occurring automorphic object plus its attached representation. The representation comparison follows under the common-polynomial, continuity, density and semisimplification hypotheses. Propagation uses two already existing compatible families, the family attached to the same automorphic object, distinguished member identities, and their common algebraic spine.

Gate	Input and formal consequence
Residual	Arithmetic residual occupancy and representability supply the deformation problem, its ring and universal representation.
Patching	An applicable arithmetic theorem supplies patched carriers, control maps and full patched support. Lean descends support to the actual finite-level carrier.
Comparison	A support-preserving comparison and compatible Hecke point are supplied. Lean transports support and constructs a nonzero specialized eigenvector.
Fiber	Ordinary tensor or derived cohomology is constructed. The actual Hecke action becomes the specialized scalar character.
Realization	Applicability of classicality to the selected eigensystem, occurrence, attachment, common polynomials, density, semisimplification and Brauer–Nesbitt are supplied. Lean deduces the intertwiner.

13 External arithmetic inputs and editorial boundaries

The formal code begins after the relevant arithmetic objects have been supplied. Retained external inputs include representability and local quotient existence; universal arithmetic deformation representations; residual automorphic occupancy; applicability of a named patching package; classicality at the selected eigensystem; an occurring automorphic object and continuous semisimple attached representation; matching Frobenius normalizations; already existing attached compatible families; arithmetic Frobenius density; construction of semisimplification; and general original-field Brauer–Nesbitt. The canonical Artin localization decomposition and the adequacy convention are also supplied inputs.

An application must state a finite ramification set and coherent choices of Frobenius convention and polynomial normalization. The literal abstract Lean group is Γ; its arithmetic realization as $G_{F,S}$, and density of the corresponding permitted conjugacy classes, are provided by the application. Finite-quotient Chebotarev followed by continuity is the relevant density mechanism for an arbitrary declared Hausdorff coefficient topology. Claims of historical novelty or citation support are not certified by the kernel.

The final traditional manuscript clarifies the statement context without changing production Lean: Claim 3.3 exposes the local structural map and the complete Hausdorff local coefficient-algebra point category; the proof of 6.4 treats $r > n$ explicitly; and Definition 12.1 exposes the same Frobenius polynomials and all realization hypotheses needed by 9.4. Compatible carrier actions and the declared-level scope of companion obstruction are also explicit, as are the finite ramification set and Frobenius convention. These are final editorial clarifications, tracked separately from the eight inherited amendments. They are not fresh kernel certification of a newly edited proposition.

The strongest global conclusion is conditional automorphic realization inside an explicitly declared supported and realized class, followed by propagation inside already existing compatible systems. Universal

residual companions, arbitrary compatible-family existence, unrestricted reciprocity and an arithmetic theorem at coefficient prime two are not conclusions of these sources.

14 Replay prerequisites, pins and audit procedure

The production proof replay is Linux/WSL oriented. Native Windows behavior is not claimed. Obtain the separate portable runtime whose root contains `toolchain/bin/lean` and `workspace/`. It must include the matching cached dependency objects; the guide does not redistribute Lean or the Mathlib dependency cache. Python 3, Bash, Git and ordinary POSIX process tools are needed by the inherited verifier.

Component	Required pin
Lean version	`4.34.0-rc2`, x86_64 Linux Release
Lean commit	`6a10ac8c22beadecabdbb0919c2b50214762f91d`
Compiler SHA-256	`e8baaa71855a616dc351028f3ad2200051b0671f423a1696a100e809302d5550`
Mathlib commit	`aeb72f1e9981b6d35b426b16dca7759be3e7ba06`
Options	One thread; warnings as errors; per-process timeout 180 seconds by default

Before replay, verify the compiler's exact hash/commit and every dependency HEAD against the runtime's dependency-pin record, not just the version substring and Mathlib HEAD. The inherited verifier checks a version substring, Mathlib HEAD/manifest, production preservation hashes, exact source manifest, object hashes during resume and all axiom targets. It sets `MALLOC_ARENA_MAX=2` and `-j1`. It does not itself impose an address-space resource limit. To reproduce the reported four-GiB limit on Linux/WSL, enforce it in the caller:

```
cd formal_release
sha256sum /path/to/Komorebi_Portable_Lean_4.34.0-rc2/toolchain/bin/lean
ulimit -v 4194304
bash Run-Reverified.sh /path/to/Komorebi_Portable_Lean_4.34.0-rc2
```

The address-space limit is in KiB in Bash on Linux. Run in a writable extracted copy: replay cleans and rewrites `build/`, `logs/`, `verification.json`, the current axiom consumer and the supplemental consumer report. Preserve the inherited records separately before executing. The Bash wrapper uses bounded four-module batches and resumes only while status is `RUNNING`. It rejects a final result other than `PASS`.

The cumulative verifier builds every registered production module from source, then imports each exact object and runs `#print axioms` for every public named declaration selected by its inventory. Only `propext`, `Classical.choice` and `Quot.sound` are allowed in the transitive audit. It finally tests three invalid inputs: a false proof, a placeholder and an incorrect empty-family normalization. All three must be rejected with the expected diagnostics; a timeout is not a successful negative control.

`check_final_claims.py` then verifies source and object hashes against the successful cumulative report and compiles the three direct consumer theorems. Core consumers are `PaperXIIFinalConsumer.exact_completed_kernel`, `claim31` and `claim33`. The native v0.17.1 archive additionally retains a distinct three-theorem `PaperXII.ManuscriptContractReplay` consumer and its `CONTRACT_REPLAY.json`. Both

are supplementary interface evidence, not six additional production theorems and not new numbered claims. The native and core production source bytes agree.

The core inherited cumulative run is recorded at 03:48:31–04:00:23 UTC on September 12, 2026. The native cumulative run is separately recorded at 04:04:33–04:15:36 UTC, with native contracts at 04:16:12–04:16:15 UTC. Matching sources, archived objects and log/report hashes corroborate provenance; they do not transform those historical runs into a new October run.

15 Package contents, Markdown files and rebuilding this document

Complete production source is in `formal_release/lean/`; it includes all reused dependencies. The directory also supplies the registered manifest, replay scripts, consumers, invalid controls, preserved compiler/axiom logs, inherited verification records, preservation ledgers and required predecessor inventory records. The original raw production sources are unchanged. The native contract consumers and their separate provenance are in `native_contract_evidence/`.

The `data/` folder supplies JSON and CSV forms of the complete source catalog, declaration census, direct import audit and revised 49-claim crosswalk. Every catalog record includes its full source hash. `SOURCE_SHA256SUMS.txt` hashes all production sources. `excerpts/` contains 24 full exact core modules, their documented typography copies, and the excerpt index with source hashes. These exact files allow every excerpt's ambient hypotheses and proof text to be inspected without relying on the PDF.

Markdown (.md) files are included. `README.md` explains the guide package, evidence scope and build commands; `SCOPE_AND_AUDIT.md` explains counts, assumptions and final self-review. Preserved release Markdown notes describe the September replay, final two claims, historical failures and mathematical self-review. Markdown is plain text and can be read without TeX or Lean.

From the release root, build this guide together with the other readers using Python 3.10 or later, TeX Live with XeLaTeX, `fontspec`, `unicode-math`, `fvextra`, `microtype`, `booktabs`, `longtable`, `fancyhdr` and `hyperref`, and Poppler's `pdfinfo`, `pdftotext`, `pdffonts`. Install P052, Nimbus Sans, DejaVu Sans Mono, and Latin Modern Math as local build dependencies. Standalone font files are not distributed. The builder stages all style and include files and does not require a bundled fonts directory.

```
python3 tools/build_readers.py --output /absolute/path/to/fresh-output
```

The builder stages its relative inputs in the fresh output directory, fixes `SOURCE_DATE_EPOCH` and the document date, suppresses optional timestamp/trailer metadata, and runs three TeX passes. It checks missing glyphs, unresolved references, overfull boxes, required text anchors and embedded fonts. Its report records PDF, extracted-text and source hashes plus page count. Independent fresh-directory rebuilds must agree byte-for-byte on the PDF, text and pages under the documented same tool versions. PDF determinism is distinct from proof replay determinism.

A Full source catalog

The complete 96-module production catalog follows in registered dependency order. "Thm." counts each module's public theorem declarations, including reused declarations where present. SHA prefixes aid visual comparison; the exact 64-character hashes and imports are supplied in `data/source_catalog.json` and `data/source_catalog.csv`. Source paths are determined by replacing dots in the module name with slashes and adding `.lean`, relative to `formal_release/lean/`.

Module	Lines	Thm.	Source SHA-256 (prefix)
PaperX.Reconstruction	189	12	d8de70c175635ea47d8eaed4
PaperXII.ArtinData	97	9	953b639b0309f883d326d0f8
PaperXII.AuxiliaryGroupAlgebra	72	7	ef811535231880a3b7aeb984
PaperXII.CompletionQuotient	181	19	c7b60b3b5a59cf73ce1891b7
PaperXII.ComplexBaseChange	86	3	6d91bbc8a4849ac4a5bae4f9
PaperXII.DenseFrobenius	113	8	941465b7bae2df984810d17a
PaperXII.DeterminantalIdeals	131	11	1091c57b8583977120bbe9d2
PaperXII.DyadicProjectors	195	17	a2d01c1ddf2152e2b6bd0269
PaperXII.FiniteCarrier	67	7	ccc9770f4c6cc72b83a94f0a
PaperXII.FiniteGroupCohomology	107	9	abab8fb127de1bbc804296e2
PaperXII.FiniteTensorQuotient	172	15	1ede310ac1f69aeeab66b851
PaperXII.LocalIdempotents	18	1	e34c194a98e51680b23542d6
PaperXII.LocalQuotients	126	11	738ec11826cea7e7aabccb35
PaperXII.ModuleSupport	63	8	b724c27d7ea2d3f2a5941ffe
PaperXII.PacketTrace	74	8	ad4c7192632b5075026bb73c
PaperXII.RepresentationData	86	6	6f938f551619e48f1f5a94ae
PaperXII.ReuseCompactness	175	12	0bd9169cb2b0efc4938e70d7
PaperXII.ReuseFiniteSeparation	97	6	1975b5fb629823eb7f80c17c
PaperXII.S3Standard	221	22	c483d46bda60362ef0650d78
PaperX.RepresentationLimits	193	15	81fb37746e7470557deced36
PaperXII.AdicModel	73	6	61c8a69111fc54671eb20b99
PaperXII.ArtinModules	206	15	78d81338b8f8f57638051245
PaperXII.AugmentationCoinvariants	120	8	9bbc93c64a5beccbe8fbc440
PaperXII.CompanionForests	84	10	db2d885e73099916d65bcaf0
PaperXII.ComplexFieldFibers	108	9	cf9b3255fcadbb9d1807f957
PaperXII.ComponentSupport	21	1	d6a2a42f8834ba233d112609
PaperXII.DeterminantalStabilization	130	10	7ee60aa621c1a1b775918634
PaperXII.ExistingCompatibleSystems	81	3	ed9748c8de4dc8693af79eb9
PaperXII.FieldFibers	73	6	36d276ab57e7cc632e15571f
PaperXII.FrobeniusSeparation	76	7	009a953c292806afb770c631
PaperXII.GaloisRealization	98	5	19e69eedbf482775e718e525
PaperXII.GeneralizedPackets	137	7	4a67970524a245b0414d0b7c
PaperXII.NonabelianResidual	142	8	923412037be8c47c1017e742
PaperXII.PerfectModels	122	2	fa7dd21c8776d96b8ab917cc
PaperXII.PowerSeriesComponent	246	38	cd0599c3aee9d2397d505928
PaperXII.QuotientSupport	60	5	733d40c9b25bbee513955de0
PaperXII.ResidualAugmentation	78	6	8ceac287f565bf81c0267dae
PaperXII.S3Adjoint	240	16	c5cee6cdd374926c9897e0a6
PaperXII.S3Detection	316	26	365b36245a4c4aa11c4522a4
PaperXII.S3Spectrum	86	8	ca1cdfc3a4797bc71ee146fa
PaperXII.StableCores	92	6	e266cbe63a32d7c4d3b719a3
PaperX.RepresentationClasses	132	16	a35e3a56c160992d4883ced0
PaperX.RepresentationTopology	124	8	9952178e10c02b66f27949cd
PaperXII.AdicQuotientTopology	105	10	c3ffb8523760a84befb64bb0
PaperXII.BalancedAugmentation	128	9	dd128eb806a101cebe777f9a
PaperXII.BaseChangeSupport	47	4	0da87b182a1e79db32ef36c5
PaperXII.ComplexSupport	103	7	186a3d1126dfa9296836b763
PaperXII.DeterminantalBlockStabilization	83	8	f6657438b1c2b5fd31012d09
PaperXII.DyadicProjectorsResidual	124	13	757fa5f1c294698c32648da8
PaperXII.FiniteAdequacy	109	6	aa7fba126405fa95fcb12944

Module	Lines	Thm.	Source SHA-256 (prefix)
PaperXII.InverseLimitRepresentations	169	9	ba425975d3baba88d65b5316
PaperXII.OrdinaryPatching	98	5	5724b6d1587cb4c657d25380
PaperXII.SupportTransport	59	4	717228feacde958e8dcd7638
PaperXII.TruncatedGroupAlgebra	297	25	676626c8a833d93839711162
PaperXII.VectorInverseLimit	122	10	ae0a5eff607a14a7e93fd523
PaperX.ClassInvariants	103	12	0506bcb396c1d8cfce4dfbed
PaperXII.AdicCompletionComparison	210	14	2dd1070d880040a9990f96b3
PaperXII.AdicMatrixGroups	163	8	21d738f44b4c35e4778528e0
PaperXII.CoinvariantTensorComparison	108	4	e920619c2a26f8b003123021
PaperXII.CompletedFiniteTensor	134	7	9c4e04f241b588d50db1d905
PaperXII.CompletedGlobalBaseChange	263	18	11fc86e509581906275b565e
PaperXII.CorrectedOccupancy	24	1	efbbb72e4f864a073569846e
PaperXII.HeckeFibers	90	7	3a0e3363a03013bd87c2f504
PaperXII.PerfectDerivedSupport	119	9	f0e29718343af1845da6b39b
PaperXII.PresentationComparison	122	4	f64299848f002c18396ce9bd
PaperXII.RightAugmentation	91	3	5fdf4abbf1666a8e03ab4cd9
PaperXII.AdicCompanions	134	7	b1f7e850bd067860825671d1
PaperXII.Coinvariants	47	2	7500fa7b2277651c91f93365
PaperXII.CompletedBaseChangeTransport	113	7	332e73f093c5fefc16ad077d
PaperXII.DerivedHeckeAction	116	7	b1cb976fcc30917465dca404
PaperXII.DerivedPatching	92	8	853d09528413ec978e2ea106
PaperXII.FiniteTensorLocality	149	11	be33c445a4329462b7039912
PaperXII.OrdinaryCompletion	213	9	07087a7da30fd656baed57b3
PaperXII.PerfectCohomology	108	10	2c560e3c14cc3082ffeb9ef4
PaperXII.PresentationIndependence	63	2	88a4c15e4ba774f041eeb364
PaperXII.AdicSpines	112	6	f90d8f75e4106eacda34b776
PaperXII.CompletedGlobalQuotient	173	17	71eede618e5c6aa481d5ec6b
PaperXII.CompletedTensorUniversal	99	7	92a58d870df81ff11bdc3f97
PaperXII.DerivedRealization	177	7	0316cfac39230ec302b3ae8b
PaperXII.HeckeCarriers	237	16	dd3bdac5d81c73a6d28bb62b
PaperXII.IntrinsicFitting	69	4	33c122a7c407120e6bda7a36
PaperXII.CompletedLocalConditions	100	3	7f7f43315592150c1f9b0e48
PaperXII.ContinuousClasses	112	10	45111ecf0dc7bd82f36a468d
PaperXII.FittingBaseChange	60	3	5d693d829097c7810a8c3ef6
PaperXII.PatchingCarriers	109	5	3c8506f8babeddb508dbeb92
PaperXII.ClassCompanions	197	15	9b177812cf70d0fc68998853
PaperXII.CompletedPaperCapstone	69	3	2489bf5ea48d25a39a90ea28
PaperXII.FittingTransport	60	3	23d49ac97ae28abf066ebd10
PaperXII.FiveGateCompletion	168	5	f107470686ed79b1cf37952c
PaperXII.ClassSpines	116	4	00aed3a592da86f63b02a21a
PaperXII.CompletedPointCategory	101	5	1e1fe94ece8611b4b6496b82
PaperXII.FittingBoundaryChecks	66	6	8516575908e9f9351e9f4666
PaperXII.AdicLocalConditions	116	8	09cce645d5f4f86c36db909a
PaperXII.CompletionBoundaryChecks	45	4	ed6c4701f20500afe72ba1c8
PaperXII.FittingStrata	64	5	a5313b7c1341dbf6bd573598
PaperXII.ConditionalCompletion	54	1	cc5234bb8a3172352e4b8e70

B All 49 revised internal claims

The crosswalk lists every revised internal claim, its manuscript label, amendment status, original status and number of mapped targets. Every target name and source module is supplied in `data/claim_crosswalk.json`; the condensed module crosswalk is `data/claim_crosswalk.csv`. "Amended" means complete for the revised statement while the original statement retains its own status. Ordinary editorial clarifications described in Section 13 are separately tracked.

Claim	Manuscript label	Revised	Original	Targets
3.1	`thm:tensor_quotient`	complete	complete	17
3.2	`cor:simultaneous_local`	complete	complete	3
3.3	`prop:global_local_basechange`	complete	complete	13
3.4	`prop:local_idempotent`	complete	complete	1
4.2	`thm:s3_adequacy`	complete	complete	20
5.1	`prop:group_algebra_transfer`	complete	complete	5
5.2	`thm:residual_augmentation_algebra`	complete	complete	7
5.3	`cor:no_residual_character_splitting`	complete	complete	7
5.4	`thm:coinvariant_recovery`	complete	complete	6
6.2	`thm:finite_module_support`	complete	complete	1
6.3	`thm:fitting_support`	complete	complete	2
6.4	`thm:higher_fitting_strata`	complete	complete	2
6.5	`prop:fitting_base_change`	complete	complete	2
6.7	`thm:near_faithful_full_support`	complete	complete	4
6.8	`cor:every_component_support`	complete	complete	2
6.9	`thm:quotient_support`	complete	complete	3
6.11	`thm:ordinary_support_descent`	complete	complete	4
6.12	`cor:componentwise_occupancy`	amended	unchecked	1
6.16	`thm:derived_quotient_support`	complete	complete	20
6.18	`thm:derived_support_descent`	complete	complete	15
6.19	`cor:derived_degree_survives`	complete	complete	7
7.1	`thm:rt_support_transport`	complete	complete	2
7.2	`cor:rt_full_support`	complete	complete	1
7.3	`prop:quotient_erases_component`	complete	complete	10
7.4	`thm:field_fiber_criterion`	complete	complete	2
7.5	`cor:supported_hecke_fiber`	complete	complete	1
8.2	`thm:canonical_hecke_packets`	complete	complete	11
8.3	`cor:simultaneous_generalized_eigenspaces`	complete	complete	6
8.4	`cor:rank_trace_detection`	complete	complete	6
8.7	`thm:nested_finite_carrier`	complete	complete	1
8.8	`cor:finite_witness_incompatibility`	complete	complete	1
8.10	`lem:stable_core_surjectivity`	complete	complete	1
8.11	`thm:nonzero_stable_core`	complete	complete	2
8.12	`cor:surjective_inverse_system`	complete	complete	1
9.3	`thm:support_gives_hecke_system`	complete	complete	2
9.4	`thm:pointwise_galois_realization`	amended	partial	7
9.5	`cor:full_support_pointwise_automorphy`	amended	partial	1
9.6	`thm:derived_pointwise_realization`	amended	partial	21
10.3	`thm:companion_branch_criterion`	complete	complete	2
10.4	`cor:empty_companion_obstruction`	complete	complete	2

Claim	Manuscript label	Revised	Original	Targets
10.5	thm:simultaneous_companion_completion	amended	partial	1
10.7	thm:one_member_propagation	complete	complete	2
10.9	prop:finite_frobenius_separator	complete	complete	2
11.1	prop:dyadic_sign_projectors	complete	complete	14
11.2	cor:dyadic_coinvariant_recovery	complete	complete	5
12.2	thm:ordinary_conditional_completion	amended	partial	7
12.4	thm:derived_conditional_completion	amended	partial	14
12.5	thm:system_completion	complete	complete	1
12.6	thm:five_gate_completion	amended	partial	36

C A practical reading and inspection sequence

A reader examining a mathematical claim should first locate its revised number in the crosswalk, then read the exact production declarations, their ambient variables and the relevant proof. Follow internal imports to any structure or construction the theorem consumes. Determine which fields are supplied data and which declarations construct the desired maps, objects or identities. Match the point category, coefficient maps, topologies, completeness, finite-generation and locality assumptions with the intended application. Consult the original ledger when comparing against earlier wording.

For Claims 3.1 and 3.3, the central source chain is the raw tensor quotient, completion/closed quotient comparison, adic quotient topology, completed finite tensor, algebraic adic comparison/locality, global tensor transport, local condition factorization, Noetherian/local capstone and coefficient/local point category. Checking only ordinary tensor quotients or an assumed universal-property interface would miss the constructed completed objects and their actual topologies.

For the arithmetic conclusion, trace the nonzero fiber or cohomology vector through the actual carrier and point. Verify classicality for that selected eigensystem, then check attachment, matching Frobenius polynomials, density, semisimplification and original-field representation comparison. Finally ensure any propagated family already exists and belongs to the same attached automorphic object. This sequence explains exactly what the formalization establishes and what an arithmetic application must still provide.