OPERATION KOMOREBI

REALIZATION CALCULUS

MATHEMATICAL MONOGRAPH

Constructive Repair

in Realization Calculus

Obstructions, admissible corrections,

and exact rational certificates

A correction is legitimate only when it removes the defect in the original mathematical object. Quotients identify what is missing. Admissibility determines which repairs are allowed.

Komorebi P.

Release candidate 1.0.0rc3 / 19 October 2026

51 unchanged Lean modules / fresh release audit

Orientation

This monograph develops a restricted theory of constructive repair. An original correction map is fixed, a family of defects is supplied, and additional correction directions may be introduced. The central question is not whether one can erase the description of a defect. It is whether the permitted changes produce an exact solution of the original correction equations.

The answer has three layers. Quotient obstructions decide feasibility. Linear sections attain the smallest possible added dimension over a field. A separate finite search chooses a minimum cost subset from a declared menu. The original finite-domain solver is retained, while the rational certificate extension checks exact coefficients without enumerating an infinite coefficient space. Faithful transport and conservation laws specify when a proposed simplification retains the information needed for those conclusions.

> **The central result**
>
> Let $D : U \to V$ be linear over a field and let
>
> $$q : V \to V/\operatorname{im} D, \qquad O = \operatorname{span}\{q(r_i) : i \in I\}.$$
>
> If V is finite dimensional, the least added dimension that repairs every r_i is $\dim O$. If an allowed subspace $A \leq V$ satisfies $O \subseteq q(A)$, the minimum can be attained with all added directions in A. Selecting entries from a fixed menu is a different optimization problem.

The mathematical development is self contained at the level of basic module theory, finite dimensional linear algebra, elementary arithmetic, and differentiation of polynomials. It refines the correction and obstruction viewpoint of Paper III rather than replacing the foundations of Realization Calculus. Its relation to the ambient realization program of Komorebi Space is discussed in section 7.3.

Formal scope. This edition retains the original 49 theorems in ten modules and the later rational certificate and costed-menu layers. The structural tree certificates and branch producer are retained unchanged. The final appendix consolidates the installed interface, current evidence bindings, compact source provenance, and precisely limited support statement. Fresh audits remain separate from historical receipts.

Reading route. Chapters 1–3 cover correction and minimum dimension, Chapters 4–5 preservation and finite search, and Chapters 6–7 arithmetic and nonlinear boundaries. Chapters 8–10 develop rational certificates, explicit attainment, and costed menus. Chapter 11 gives the structural tree checker. The appendices record formal correspondence and verification. The final appendix describes the current release contract without duplicating chronological engineering reports. Declaration counts describe coverage, not novelty.

Contents

Orientation **2**

1 Correction and quotient obstructions **7**

1.1 The original equation 7

1.2 What the quotient remembers 7

1.3 Repair as an image condition 8

1.4 The obstruction span of a family 9

2 Minimum repair dimension **10**

2.1 A lower bound that every repair must satisfy 10

2.2 An attaining construction 10

2.3 Directions restricted to a subspace 11

2.4 Matrix coordinates and a boundary over rings 12

3 Finite menus and conservation **13**

3.1 A menu is an actual coefficient map 13

3.2 One obstruction, two selected entries 14

3.3 Preserved linear invariants 15

4 Faithful transport of obstructions **16**

4.1 The induced quotient map 16

4.2 Faithfulness on precisely the required data 17

4.3 A transport that conceals failure 17

5 Certified finite search **19**

5.1 The finite contract 19

5.2 Enumerating menus and selecting an optimum 19

5.3 A fully evaluated example over two elements 21

5.4 What the algorithm does not promise 21

6 Arithmetic digit repair **22**

6.1 An exact correction at one scale . . . 22

6.2 The quadratic lifting theorem . . . 23

6.3 An exact seven-adic trace . . . 23

6.4 Why a singular derivative is inconclusive . . . 24

7 Nonlinear limits and native realizations **25**

7.1 A linearized problem is a different equation . . . 25

7.2 Three conclusions that must stay distinct . . . 26

7.3 The relation to Realization Calculus . . . 26

7.4 Established reach and remaining questions . . . 26

8 Exact rational certificates **28**

8.1 Let the search propose and the kernel check . . . 28

8.2 Separating obstructions . . . 28

8.3 Dimension certificates without a reported rank . . . 29

8.4 Exact rank and menu cost . . . 30

9 Constructive matrix attainment **32**

9.1 A complete detector . . . 32

9.2 The attaining construction . . . 32

9.3 How the Python producer obtains the matrices . . . 33

9.4 A full intermediate trace . . . 33

9.5 Serialization and input identity . . . 34

9.6 Legality and coefficient rings . . . 35

10 Costed rational menus and rejection covers **36**

10.1 The original menu contract . . . 36

10.2 Separating blocks . . . 37

10.3 Complete coverage of cheaper alternatives . . . 38

10.4 Constructing the evidence . . . 38

10.5 Two entries for one missing dimension . . . 39

10.6 Cost, dimension, and evidence size . . . 40

10.7 Fractional witnesses and independent original data . . . 40

10.8 Bounded executable use . . . 41

11 Proof carrying branch search **42**

11.1 Intervals of candidate selections . . . 42

11.2 Three terminal certificates . . . 43

11.3 Soundness of the whole tree . . . 44

11.4 How the producer constructs evidence . . . 45

11.5 A twenty-four-candidate construction . . . 45

11.6 Performance and semantic limits . . . 46

A Formal correspondence **47**

A.1 How the prose matches the source . . . 47

A.2 Constructive existence and executable construction . . . 47

A.3 Stable source locators . . . 48

B Original declaration register **49**

B.1 Basic . . . 49

B.2 Minimal . . . 49

B.3 ConstrainedMinimal . . . 50

B.4 Menu . . . 50

B.5 Conservation . . . 50

B.6 Transport . . . 51

B.7 Search . . . 51

B.8 Arithmetic . . . 51

B.9 Nonlinear . . . 52

B.10 Examples . . . 52

C Verification and provenance **53**

C.1 The evidence attached to the formal release . . . 53

C.2 Negative controls and their meaning . . . 54

C.3 Experimental evidence is a different layer . . . 54

C.4 The monograph edition . . . 54

D Rational bridge declaration register **55**

D.1 RationalCertificates . . . 55

D.2 ConstructiveMatrix . . . 56

D.3 Wire . . . 56

D.4 Concrete exported problems . . . 56

D.5 Menu and corruption examples . . . 57

E Rational bridge verification and use **58**

E.1 The v0.3.0 checkpoint 58

E.2 The single-command example 59

E.3 Failures and their corrections 59

E.4 Remaining boundaries 59

F Menu certificates: sources and fresh verification **61**

F.1 General declarations 61

F.2 Exported original problems 62

F.3 Fresh verification results 63

F.4 Rejection controls and command execution 63

F.5 Development record and remaining scope 64

G Branch certificate verification **65**

G.1 General declaration correspondence 65

G.2 Exported case register 65

G.3 Fresh combined verification 66

G.4 Controls, command execution, and provenance 67

G.5 Trust and remaining scope 68

H Release contract and reproducibility **69**

H.1 A frozen mathematical component 69

H.2 Using the installed interface 69

H.3 A gate for the exact distributed artifact 70

H.4 Compact provenance without recursive archives 71

H.5 Acceptance and supported environment 71

H.6 Remaining decisions and limits 72

References **73**

Correction and quotient obstructions

1.1 The original equation

A residual describes a failure relative to a stated equation. A correction map describes the changes that are allowed to address that failure. These are separate pieces of data. Their separation prevents the phrase "the obstruction" from concealing an unstated choice of permissible operations.

Let $\Bbbk$ be a commutative ring. All modules are unital $\Bbbk$-modules and all maps in this chapter are $\Bbbk$-linear. Fix

$$D : U \longrightarrow V,$$

and let $r : I \to V$ be a family of residuals. The index set I need not be finite for the algebraic statements. When finite enumeration is used later, finiteness will be imposed explicitly.

Definition 1.1 (Correction and repair)**.** A residual $r \in V$ is *correctable* by D when there is $u \in U$ such that

$$Du = -r. \tag{1.1}$$

An added correction map $E : W \to V$ *repairs* r when there are $u \in U$ and $w \in W$ with

$$Du + Ew = -r. \tag{1.2}$$

It repairs the family when this condition holds for every r_i. The witnesses may depend on i.

The sign convention makes the repaired residual $r + Du + Ew$ equal to zero. In particular, the target of the equation remains V. A calculation in another space can establish (1.2) only if a valid route back to V is supplied.

The family condition does not ask for one common pair (u, w) that corrects every residual at once. It asks for one common added map E and separate coefficients for each residual. Nor does it require those witnesses to vary continuously, measurably, or linearly with the index. Such conditions would define different problems.

1.2 What the quotient remembers

Write

$$Q_D = \operatorname{coker} D = V/\operatorname{im} D, \qquad q_D : V \longrightarrow Q_D.$$

The class $q_D(r)$ records the part of r that cannot be changed by the original correction map. Since $q_D(Du) = 0$, all residuals that differ by an original correction have the same class.

Theorem 1.2 (Exact obstruction criterion)**.** For every $r \in V$,

$$\exists u \in U \ (Du = -r) \quad \Longleftrightarrow \quad q_D(r) = 0.$$

Proof. Applying q_D to $Du = -r$ gives $0 = -q_D(r)$, hence $q_D(r) = 0$. Conversely, $q_D(r) = 0$ means $r \in \operatorname{im} D$. Choose $v \in U$ with $Dv = r$. Then $D(-v) = -r$. □

Formal counterpart: `correctable_iff_obstruction_zero`.

This is an exact criterion for the declared linear correction equation. It is not a universal test for a nonlinear equation from which D might have been obtained by differentiation. Chapter 7 gives counterexamples in both directions.

The quotient also has a universal property. If $L : V \to Z$ annihilates $\operatorname{im} D$, it descends to a unique linear map $\overline{L} : Q_D \to Z$ with $L = \overline{L} q_D$. Thus every linear detector that ignores correctable directions factors through the same quotient. This is the classical cokernel construction used in Paper III [7]. Canonicality is relative to D, not to an unspecified mathematical problem.

Example 1.3 (A visible residual)**.** Let $D : \mathbb{Q} \to \mathbb{Q}^2$ be $D(u) = (u, 0)$ and let $r = (2, 3)$. The quotient is identified with the second coordinate. The correction $u = -2$ removes the first coordinate but leaves $(0, 3)$. The functional $L(x, y) = y$ satisfies $LD = 0$ and $L(r) = 3$, so no original correction eliminates r.

1.3 Repair as an image condition

An added map $E : W \to V$ contributes precisely the classes in $\operatorname{im}(q_D E)$. This leads to the basic feasibility criterion.

Theorem 1.4 (A single residual)**.** For every $E : W \to V$ and $r \in V$,

$$\exists u, w \ (Du + Ew = -r) \quad \Longleftrightarrow \quad q_D(r) \in \operatorname{im}(q_D E). \tag{1.3}$$

Proof. Suppose $Du + Ew = -r$. Applying the quotient gives $q_D(Ew) = -q_D(r)$, so $q_D(r) = q_D(E(-w))$ lies in the image.

Conversely, choose w with $q_D(Ew) = q_D(r)$. Then $q_D(r - Ew) = 0$, so there is $v \in U$ with $Dv = r - Ew$. Taking $u = -v$ and $w' = -w$ gives

$$Du + Ew' = -(r - Ew) - Ew = -r.$$

The witnesses solve the equation in V, not just in its quotient. □

Formal counterpart: `repairable_iff_obstruction_mem_range`.

Corollary 1.5 (Reconstruction from a quotient witness)**.** If $q_D(Ew) = q_D(r)$, there are $u \in U$ and $w' \in W$ such that

$$Du + Ew' + r = 0.$$

Proof. Use the witnesses constructed in the reverse direction of theorem 1.4 and move r to the left. □

Formal counterpart: `reconstruct_cancellation`.

1.4 The obstruction span of a family

Definition 1.6 (Required obstruction span)**.** For a residual family $r : I \to V$, put

$$O_D(r) = \operatorname{span}_{\Bbbk}\{q_D(r_i) : i \in I\} \leq Q_D.$$

We write O when the family and correction map are understood.

Every $q_D(r_i)$ lies in O. The use of a span matters because an added linear correction map supplies a submodule of quotient classes, not merely a list of unrelated vectors.

Theorem 1.7 (Family feasibility)**.** The map $E : W \to V$ repairs the family r if and only if

$$O_D(r) \subseteq \operatorname{im}(q_D E). \tag{1.4}$$

Proof. If every residual is repairable, theorem 1.4 places every generator $q_D(r_i)$ in $\operatorname{im}(q_D E)$. The image is a submodule, so it contains their span. Conversely, (1.4) places each generator in that image, and theorem 1.4 supplies its correction witnesses. □

Formal counterpart: `family_repairable_iff_span_le`.

For an empty family, $O = 0$ and the family condition is vacuous. For a nonempty family whose residuals are already correctable, O is still zero. Both cases should permit a repair of added dimension zero. The next chapter includes them without a special exception.

> **What has been established**
>
> The quotient determines feasibility only because the correction map and the original equation have been declared. A positive quotient witness can be lifted to actual cancellation in V. A disappearance caused by forgetting part of V is a different operation and requires a preservation theorem.

CHAPTER 2

Minimum repair dimension

2.1 A lower bound that every repair must satisfy

In this chapter $\mathbb{k}$ is a field. The added space W is finite dimensional when its numerical dimension is compared. The attainment result itself uses a linear section and does not require all ambient spaces to be finite dimensional.

Theorem 2.1 (Dimension lower bound)**.** If $E : W \to V$ repairs r and W is finite dimensional, then $O = O_D(r)$ is finite dimensional and

$$\dim_{\mathbb{k}} O \leq \dim_{\mathbb{k}} W.$$

Proof. By theorem 1.7, $O \subseteq \operatorname{im}(q_D E)$. The image of a linear map from a finite dimensional space is finite dimensional and has dimension at most that of its domain. Hence

$$\dim O \leq \dim \operatorname{im}(q_D E) \leq \dim W.$$ □

Formal counterpart: `repair_dimension_lower_bound`.

The bound measures missing independent directions, not the number of residuals. A large family can have a one dimensional obstruction span. Conversely, a single residual never requires more than one freely chosen direction, since its own negative may be made available. These statements concern unconstrained linear corrections and say nothing about the cost of producing a permitted direction in a domain application.

2.2 An attaining construction

Lemma 2.2 (A section of the quotient)**.** There is a linear map $s : Q_D \to V$ with

$$q_D s = \mathrm{id}_{Q_D}.$$

Proof. Choose a basis of the vector space Q_D. Since q_D is surjective, choose a representative in V of each basis vector. Extend this assignment linearly. Its composite with q_D fixes the basis and therefore fixes all of Q_D. □

Formal counterpart: `exists_obstruction_section`.

This choice of section is generally nonunique. Changing a representative by a vector in $\operatorname{im} D$ produces another valid section. The construction is an existence proof using classical choice, not an algorithm for selecting a basis.

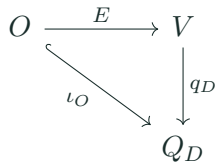
Theorem 2.3 (Attainment on the obstruction span)**.** Let $\iota_O : O \hookrightarrow Q_D$ be inclusion. There exists a linear map $E : O \to V$ such that

$$q_D E = \iota_O \tag{2.1}$$

and E repairs the family.

Proof. Choose the section s from theorem 2.2 and set $E = s\iota_O$. Then $q_D E = q_D s \iota_O = \iota_O$. Every $z \in O$ is therefore in $\operatorname{im}(q_D E)$. Family feasibility follows from theorem 1.7. □

Formal counterpart: `exists_attaining_repair`.



The triangle records exactly what is preserved: a required quotient class is lifted to a representative of that same class. The lift does not manufacture an unrelated object with a similar invariant.

Theorem 2.4 (The attained free minimum)**.** If V is finite dimensional, the minimum possible dimension of a finite dimensional added space that repairs the family is

$$\min\{\dim W : E : W \to V \text{ repairs } r\} = \dim O_D(r). \tag{2.2}$$

Proof. Every admissible term on the left is at least $\dim O$ by theorem 2.1. Because V is finite dimensional, its quotient and the subspace O are finite dimensional. The map in theorem 2.3 is a repair with domain O, so the bound is attained. □

Formal counterpart: `free_minimal_repair_rank`.

No desired existence statement occurs among the inputs. The theorem proves nonemptiness of the optimization class as well as its lower bound.

2.3 Directions restricted to a subspace

An application usually prohibits some corrections. A first useful constraint is a subspace $A \leq V$ containing every permitted added direction. Its quotient image is

$$q_D(A) = \operatorname{im}(q_D|_A : A \to Q_D).$$

Theorem 2.5 (Attainment inside a legal subspace)**.** If $O \subseteq q_D(A)$, there exists a linear map $E : O \to V$ satisfying

$$\operatorname{im} E \subseteq A, \qquad q_D E = \iota_O,$$

and repairing the entire family. If V is finite dimensional, its added dimension is optimal even among all unrestricted repairs.

Proof. Consider the surjection

$$f : A \longrightarrow q_D(A), \qquad a \longmapsto q_D(a).$$

Choose a linear section $s_A : q_D(A) \to A$. Let $j : O \hookrightarrow q_D(A)$ and $\iota_A : A \hookrightarrow V$ be inclusions. Define

$$E = \iota_A s_A j.$$

Its image is in A by construction. Moreover, $q_D \iota_A s_A j = j$ as a map into Q_D, which is precisely ι_O. By theorem 1.7, E repairs the family. The lower bound of theorem 2.1 proves optimality when dimensions are finite. □

Formal counterpart: `admissible_subspace_attainment`.

Theorem 2.6 (Subspace impossibility)**.** If $O \not\subseteq q_D(A)$, no added map $E : W \to V$ with image contained in A repairs the family.

Proof. Such a map satisfies $\operatorname{im}(q_D E) \subseteq q_D(A)$. If it repaired the family, theorem 1.7 would imply $O \subseteq \operatorname{im}(q_D E) \subseteq q_D(A)$, contrary to the hypothesis. □

Formal counterpart: `admissible_subspace_no_go`.

The word "legal" means membership in a specified linear subspace. It does not mean that the paper has constructed an arithmetic host, an equivariant representation, or a topology on that space. Those interpretations require separate witnesses.

2.4 Matrix coordinates and a boundary over rings

For finite matrices, the abstract minimum has a familiar coordinate form.

Corollary 2.7 (Rank difference in matrix coordinates)**.** Let the columns of a matrix R be a finite residual family and let D represent D. Then

$$\dim O_D(r) = \operatorname{rank}[\mathsf{D} \mid \mathsf{R}] - \operatorname{rank} \mathsf{D}.$$

Proof. Set $M = \operatorname{im} D + \operatorname{span}\{r_i\}$. The restriction $q_D|_M$ has image O and kernel $\operatorname{im} D$. Rank and nullity give $\dim M = \dim \operatorname{im} D + \dim O$. The columns of $[\mathsf{D} \mid \mathsf{R}]$ span M. □

This coordinate corollary is an ordinary derivation included for the computational interpretation. The v0.2.0 Lean release proves the abstract attained dimension theorem, but does not yet identify this matrix formula with the Python rank implementation.

The field hypothesis in the section argument is substantive. Over $\mathbb{Z}$, take $D : \mathbb{Z} \to \mathbb{Z}$ to be multiplication by two. The quotient map $q : \mathbb{Z} \to \mathbb{Z}/2\mathbb{Z}$ has no $\mathbb{Z}$-linear section. A section would send the nonzero class to an integer n with $2n = 0$, forcing $n = 0$ and contradicting $q(n) = 1$. This elementary example explains why the module feasibility theorem does not automatically imply a splitting theorem over arbitrary rings.

CHAPTER 3

Finite menus and conservation

3.1 A menu is an actual coefficient map

Return to a commutative ring $\Bbbk$ and modules U, V. Let $e : J \to V$ be a family of candidate directions and let $S \subseteq J$ be finite. A coefficient vector is a function $a : S \to \Bbbk$. Define

$$E_S : \Bbbk^S \longrightarrow V, \qquad E_S(a) = \sum_{j \in S} a_j e_j. \tag{3.1}$$

Finite addition and scalar multiplication show that E_S is linear. The coefficient domain is the selected set itself, so no unselected direction appears in the equation.

Write

$$O_S = \operatorname{span}_{\Bbbk}\{q_D(e_j) : j \in S\}.$$

Lemma 3.1 (Quotient image of a menu)**.** One has $\operatorname{im}(q_D E_S) = O_S$.

Proof. For every coefficient vector a,

$$q_D(E_S a) = \sum_{j \in S} a_j q_D(e_j),$$

which lies in O_S. Conversely, every generator $q_D(e_j)$ is the image of the coefficient vector that equals one at j and zero elsewhere. Since the image is a submodule, it contains their span. □

Formal counterpart: `menu_quotient_range`.

Theorem 3.2 (Exact finite menu feasibility)**.** The selected menu S repairs the family exactly when

$$O_D(r) \subseteq O_S. \tag{3.2}$$

Equivalently,

$$\forall i \in I \ \exists u_i \in U \ \exists a_i \in \Bbbk^S, \qquad D u_i + \sum_{j \in S} a_{ij} e_j = -r_i.$$

Proof. Apply theorem 1.7 to the actual map E_S and use theorem 3.1. Its reverse direction supplies the original correction witnesses and the selected coefficients, rather than only an abstract statement about dimensions. □

Formal counterpart: `finite_menu_repair_iff_span`.

Proposition 3.3 (Monotonicity of feasibility)**.** If $S \subseteq T$, then $O_S \subseteq O_T$. Every family repairable with S is therefore repairable with T.

Proof. Each generator defining O_S is also a generator defining O_T. Apply theorem 3.2. Equivalently, extend each coefficient vector by zero on $T \setminus S$. □

Formal counterpart: `repairable_menu_mono`.

An external admissibility condition need not be monotone. A rule allowing at most one selected direction can reject an enlarged menu even though algebraic feasibility only improves. Cost and admissibility must not be confused with span inclusion.

3.2 One obstruction, two selected entries

Example 3.4 (The difference between the two minima)**.** Over $\mathbb{Q}$, let

$$D(u) = (u, 0, 0), \qquad r = (0, 1, 0),$$

and make only the following directions available:

$$e_0 = (0, 1, 1), \qquad e_1 = (0, 0, 1).$$

The quotient by $\operatorname{im} D$ is the plane of the second and third coordinates. There $q_D(r) = (1, 0)$, $q_D(e_0) = (1, 1)$, and $q_D(e_1) = (0, 1)$.

Using only e_0, the repair equation reads

$$(u, a, a) = (0, -1, 0).$$

The second coordinate forces $a = -1$ and the third forces $a = 0$, a contradiction. Using only e_1, the second coordinate stays zero and cannot become -1. Both directions together give the exact witness

$$D(0) - e_0 + e_1 = (0, -1, 0) = -r. \tag{3.3}$$

The empty menu also fails. Its correction range is the first coordinate axis.

Selected set	Quotient directions	Repairs r?
$\varnothing$	0	No
$\{0\}$	$\operatorname{span}\{(1, 1)\}$	No
$\{1\}$	$\operatorname{span}\{(0, 1)\}$	No
$\{0, 1\}$	$\mathbb{Q}^2$	Yes

The obstruction span has dimension one. A freely chosen direction r repairs the residual with coefficient -1. The prescribed menu nevertheless requires two selected entries. Both statements are formalized in the examples module. The difference is not a contradiction: one cost counts independent added dimensions, the other counts named entries that must be purchased or enabled.

A weighting makes the distinction sharper. Assigning positive costs c_0, c_1 gives optimum $c_0 + c_1$ for this menu. A subspace containing $e_0 - e_1$ does not charge those two entries separately unless its cost model explicitly does so.

3.3 Preserved linear invariants

Conservation laws supply impossibility certificates without enumerating every correction.

Proposition 3.5 (Invariant preservation)**.** Let $L : V \to Z$ be linear. If $LD = 0$ and $LE = 0$, then

$$L(v + Du + Ew) = L(v)$$

for all v, u, w.

Proof. Linearity gives $L(v + Du + Ew) = L(v) + LD(u) + LE(w)$. The last two terms vanish. □

Formal counterpart: `invariant_preserved`.

Theorem 3.6 (Conservation obstruction)**.** If $LD = 0$, $LE = 0$, and $L(r) \neq 0$, then E does not repair r.

Proof. A repaired residual would be $r + Du + Ew = 0$. Applying theorem 3.5 would give $L(r) = L(0) = 0$, contrary to the hypothesis. □

Formal counterpart: `conservation_no_go`.

Corollary 3.7 (Conservation for a selected menu)**.** Suppose $LD = 0$ and $L(e_j) = 0$ for every $j \in S$. If $L(r) \neq 0$, no coefficients from S repair r.

Proof. For every $a : S \to \mathbb{k}$,

$$L(E_S a) = \sum_{j \in S} a_j L(e_j) = 0.$$

Thus $LE_S = 0$, and theorem 3.6 applies. □

Formal counterpart: `conservation_menu_no_go`.

For instance, let $D(u) = (u, 0, 0)$ and permit additional directions only on the third coordinate axis. The invariant $L(x, y, z) = y$ annihilates every legal correction. It detects $r = (0, 1, 0)$, so no legal repair exists. Enlarging the number of copies of the third coordinate direction cannot help.

The theorem is only as meaningful as the invariant it uses. In a domain application one must establish that L is the intended native invariant and that every permitted transformation really preserves it. An abstract functional with a suggestive name is not such a proof.

CHAPTER 4

Faithful transport of obstructions

4.1 The induced quotient map

Suppose two correction systems are related by a commuting square

$$\begin{array}{ccc} U & \xrightarrow{D} & V \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle b} \\ U' & \xrightarrow[D']{} & V'. \end{array} \qquad bD = D'a. \tag{4.1}$$

All four spaces are modules over the same commutative ring. Denote the quotient maps by q_D and $q_{D'}$.

Lemma 4.1 (The correction image descends)**.** The map b sends $\operatorname{im} D$ into $\operatorname{im} D'$.

Proof. If $v = Du$, then $b(v) = bD(u) = D'a(u)$, which lies in $\operatorname{im} D'$. □

Formal counterpart: `correction_range_maps`.

It follows that there is a well defined linear map

$$B : Q_D \longrightarrow Q_{D'}, \qquad B(q_D(v)) = q_{D'}(b(v)).$$

Indeed, changing v by an element of $\operatorname{im} D$ changes $b(v)$ by an element of $\operatorname{im} D'$. This is the quotient map constructed in the formal source, not an assumed function on unidentified obstruction spaces.

Proposition 4.2 (Naturality)**.** For every residual $r \in V$,

$$B(q_D(r)) = q_{D'}(b(r)). \tag{4.2}$$

Proof. This is the defining equation of the induced quotient map. □

Formal counterpart: `transport_obstruction_naturality`.

Naturality transports a correctable residual to a correctable one. The converse can fail because B may annihilate a nonzero class. A second condition is needed to use target vanishing as a source certificate.

4.2 Faithfulness on precisely the required data

Definition 4.3 (Faithfulness on a submodule)**.** For $O \leq Q_D$, the map B is *faithful on* O when

$$\forall z \in O, \qquad Bz = 0 \Longrightarrow z = 0.$$

This condition is equivalent to $\ker B \cap O = 0$. It does not require injectivity on all of Q_D when only a smaller family of obstruction classes matters.

Proposition 4.4 (Equivalent injectivity criterion)**.** The map B is faithful on O if and only if its restriction $B|_O : O \to Q_{D'}$ is injective.

Proof. If B is faithful and $Bx = By$ for $x, y \in O$, then $x - y \in O$ and $B(x - y) = 0$. Faithfulness gives $x = y$. Conversely, if $B|_O$ is injective and $Bz = 0 = B0$ for $z \in O$, injectivity gives $z = 0$. □

Formal counterpart: `faithfulOn_iff_injective`.

Theorem 4.5 (Reflection of correctability)**.** Let $O = O_D(r)$. If B is faithful on O, then for every $i \in I$,

$$b(r_i) \text{ is correctable by } D' \quad \Longleftrightarrow \quad r_i \text{ is correctable by } D.$$

Proof. By theorem 1.2, target correctability means $q_{D'}(b(r_i)) = 0$. Naturality identifies this class with $B(q_D(r_i))$. Because $q_D(r_i) \in O$, faithfulness then yields $q_D(r_i) = 0$, which is source correctability. Conversely, if $q_D(r_i) = 0$, naturality and linearity give $q_{D'}(b(r_i)) = B0 = 0$. □

Formal counterpart: `transport_faithfulness_on_obstruction_span`.

Proposition 4.6 (A witness to information loss)**.** If B is not faithful on O, there is $z \in O$ such that

$$z \neq 0, \qquad Bz = 0.$$

Proof. Negating the quantified implication in theorem 4.3 gives an element of O whose image is zero and which is itself nonzero. □

Formal counterpart: `nonfaithful_witness`.

The witness may be a linear combination of the supplied obstruction classes rather than one particular generator. This is why injectivity on the entire required span is a stronger preservation test than checking that a short list of nonzero classes individually remains nonzero.

4.3 A transport that conceals failure

Example 4.7 (Correct in the target, false in the source)**.** Let

$$D : \mathbb{Q} \to \mathbb{Q}^2, \quad D(u) = (u, 0), \qquad D' = \mathrm{id}_{\mathbb{Q}}, \quad a = \mathrm{id}_{\mathbb{Q}}, \quad b(x, y) = x.$$

Then $bD = D'a$. For $r = (0, 1)$ there is no u with $(u, 0) = (0, -1)$, so the source residual is not correctable. Its image $b(r) = 0$ is correctable by D'.

Here Q_D is one dimensional and $Q_{D'} = 0$. The induced map B annihilates every source obstruction. It is therefore not faithful on the span of $q_D(r)$. The formal example proves both the false positive produced by projection and the failure of the faithfulness hypothesis.

A second elementary calculation clarifies why faithfulness is imposed on a span. The map $B : \mathbb{Q}^2 \to \mathbb{Q}$, $B(x, y) = x + y$, sends each standard basis vector to the nonzero number one. Nevertheless, it kills their difference $(1, -1)$. Survival of generators is not injectivity on their span. This calculation is explanatory and has no separate declaration in the present formal source.

> **Transport is not repair**
>
> A repair changes the allowed correction image while keeping the original equation meaningful. A transport changes the representation of that equation. The two may be composed, but vanishing in the transported space certifies source correctability only when the stated preservation hypotheses hold.

CHAPTER 5

Certified finite search

5.1 The finite contract

The algebraic existence theorems need not be executable. A linear section of a quotient may be chosen by classical choice, and a feasibility statement over an arbitrary infinite field need not come with a terminating decision procedure. The finite reference solver uses a different contract.

Assume $\Bbbk$, U, I, and J have specified finite enumerations. The ring $\Bbbk$ is commutative, U, V are $\Bbbk$-modules, and equality in V is decidable. The target module V need not itself have a finite enumeration. Fix a linear order on J so that the candidate universe can be enumerated in a definite order.

For each finite subset $S \subseteq J$, define the Boolean test

$$\mathrm{check}(S) = \mathrm{decide}\left(\forall i \in I\ \exists u \in U\ \exists a \in \Bbbk^S, \quad Du + E_S a = -r_i\right). \tag{5.1}$$

All quantified domains are finite. Decidable equality in V determines each candidate equation.

Proposition 5.1 (The Boolean test has the intended meaning)**.** The equality $\mathrm{check}(S) = \mathrm{true}$ holds if and only if E_S repairs the residual family.

Proof. The Boolean is defined by deciding precisely the quantified equations in theorem 1.1 for the map E_S. No intermediate rank guess or unproved feasibility oracle replaces those equations. □

Formal counterpart: `repairMenuCheck_spec`.

Additional admissibility is a supplied function $\mathrm{allowed} : \mathcal{P}(J) \to \{\mathrm{false}, \mathrm{true}\}$. The cost is any supplied function $c : \mathcal{P}(J) \to \mathbb{N}$. The formal theorem does not require c to be additive in the selected entries.

5.2 Enumerating menus and selecting an optimum

Enumerate J as a sorted list. Take every sublist and convert it to a finite set. The resulting list, denoted $\mathcal{L}_J$, contains every subset of J.

Lemma 5.2 (Coverage of the menu universe)**.** Every finite subset $S \subseteq J$ occurs in $\mathcal{L}_J$.

Proof. Filter the sorted full list by membership in S. The result is a sublist of the full list and therefore appears among its sublists. Converting this filtered list to a finite set gives exactly S. □

Formal counterpart: `mem_allMenus`.

For a finite list $\mathcal{L}$ of candidates, a Boolean predicate h, and a natural cost c, define $\mathrm{bestByCheck}(h, c, \mathcal{L})$ to filter by h and then return an argument of minimum cost. It returns `none` exactly when the filtered list is empty. Ties need not define a mathematically unique optimum.

Lemma 5.3 (Finite selection)**.** If this selector returns x, then $x \in \mathcal{L}$, $h(x) = \mathrm{true}$, and $c(x) \le c(y)$ for every $y \in \mathcal{L}$ with $h(y) = \mathrm{true}$. It returns `none` if and only if no listed candidate passes h.

Proof. Every element of the filtered list is an original list member satisfying h. A minimum cost element of that finite list has cost no larger than that of any other member. The selector returns no element exactly when the filtered list is empty, which is exactly the stated rejection condition. □

Formal counterpart: `bestByCheck_spec`.

The repair optimizer is

$$\mathrm{bestRepairMenu} = \mathrm{bestByCheck}\big(S \mapsto \mathrm{allowed}(S) \wedge \mathrm{check}(S),\ c,\ \mathcal{L}_J\big).$$

Theorem 5.4 (Soundness and minimum cost)**.** If the optimizer returns `some S`, then S is allowed, E_S repairs the family, and

$$c(S) \le c(T)$$

for every allowed subset $T \subseteq J$ whose menu map repairs the family.

Proof. Apply theorem 5.3. Passing the conjunction gives both admissibility and $\mathrm{check}(S) = \mathrm{true}$. Proposition 5.1 converts the latter into actual repair feasibility. Every competing T occurs in $\mathcal{L}_J$ by theorem 5.2, so the same finite selection lemma compares its cost to $c(S)$. □

Formal counterpart: `menu_optimality_certificate`.

Theorem 5.5 (Complete rejection within the declared universe)**.** The optimizer returns `none` if and only if no allowed subset of J repairs the family.

Proof. If it returns `none`, theorem 5.3 excludes every passing member of $\mathcal{L}_J$. Any allowed feasible subset would occur there by theorem 5.2 and pass the Boolean test by theorem 5.1, a contradiction. Conversely, if no allowed subset repairs the family, no listed subset satisfies the filter, so the selector returns `none`. □

Formal counterpart: `menu_search_none_iff`.

The output is a selected subset or a rejection, not an extracted table of correction coefficients. The correctness theorem establishes existence of the coefficients. A separately implemented witness extraction or serialized certificate checker would provide those coefficients as a returned data object.

5.3 A fully evaluated example over two elements

Take the data of theorem 3.4 over $\mathbb{F}_2$. Negation is now the identity. The equation is

$$(u, a, a + b) = (0, 1, 0).$$

Thus the full menu succeeds with $u = 0$, $a = b = 1$. The two singleton menus still fail.

Menu	Cardinality	All menus allowed	At most one entry
$\varnothing$	0	Infeasible	Infeasible
$\{0\}$	1	Infeasible	Infeasible
$\{1\}$	1	Infeasible	Infeasible
$\{0, 1\}$	2	Feasible	Not allowed

With cardinality as the cost, the two evaluations are

```
All subsets permitted:                  some {0, 1}
Only subsets with cardinality at most 1: none
```

These equalities are the statements of `executable_menu_returns_both` and `executable_restricted_menu_rejects`. Their source proofs use kernel reduction. The generic optimizer theorem then proves that every feasible subset has cardinality at least two, recorded as `executable_menu_is_globally_optimal`. Printed evaluation output is kept distinct from the proof of that output.

5.4 What the algorithm does not promise

The reference solver is deliberately finite and exhaustive. With $t = |J|$, there are 2^t candidate subsets. For a subset of size s, each residual has at most $|U|\,|\mathbb{k}|^s$ candidate coefficient and correction assignments to examine. Summing this upper bound over all subsets gives

$$|I|\,|U| \sum_{s=0}^{t} \binom{t}{s} |\mathbb{k}|^s = |I|\,|U|(1 + |\mathbb{k}|)^t$$

possible equation tests before accounting for short circuiting and inadmissible subsets. This is an elementary enumeration bound, not a formal runtime theorem about the compiled Lean implementation.

The Python prototype uses exact Gaussian elimination for its linear equations. The finite Lean solver instead decides those equations by finite enumeration. They share a mathematical specification in the relevant finite cases, but the present release does not contain a proof identifying their implementations. In particular, the Lean optimizer is not an executable search over arbitrary rational coefficient assignments.

CHAPTER 6

Arithmetic digit repair

6.1 An exact correction at one scale

Arithmetic repair adds a second kind of constraint: the new correction must improve divisibility while retaining the previous approximation. A quadratic equation makes the mechanism visible without appealing to a general existence theorem.

The results in this chapter are first stated over $\mathbb{Z}$. An integer p need not be prime for the finite divisibility implications. The interpretation as a p-adic lift uses a positive prime p.

Lemma 6.1 (A linear digit from Bézout data)**.** Let $p, A, c \in \mathbb{Z}$. If there are $s, b \in \mathbb{Z}$ with $sA+bp = 1$, then there is $t \in \mathbb{Z}$ such that

$$p \mid c + At.$$

Proof. Set $t = -sc$. Then

$$c + At = c(1 - As) = pbc.$$ □

Formal counterpart: `Arithmetic.linear_digit_exists`.

Lemma 6.2 (Quadratic correction identity)**.** Let $p, m, x, a, c, t \in \mathbb{Z}$ satisfy

$$x^2 - a = mc, \qquad p \mid c + 2xt, \qquad p \mid m.$$

Then

$$mp \mid (x + mt)^2 - a.$$

Proof. Write $c + 2xt = pd$ and $m = ps$. Direct expansion gives

$$\begin{aligned}(x + mt)^2 - a &= (x^2 - a) + 2mxt + m^2t^2 \\ &= m(c + 2xt) + m^2t^2 \\ &= mp(d + st^2).\end{aligned}$$ □

The last expression displays the required quotient explicitly.

Formal counterpart: `Arithmetic.quadratic_digit_correction`.

The term m^2t^2 is not discarded as an approximation. Its divisibility follows from $p \mid m$. This is what makes a linear digit equation sufficient for the finite quadratic step.

6.2 The quadratic lifting theorem

Theorem 6.3 (One higher precision)**.** Let $p, x, a \in \mathbb{Z}$ and $k \in \mathbb{N}$ with $k \geq 1$. Assume

$$p^k \mid x^2 - a, \qquad \gcd(2x, p) = 1.$$

Then there exists $t \in \mathbb{Z}$ such that

$$p^{k+1} \mid (x + p^k t)^2 - a. \tag{6.1}$$

Proof. Choose $c \in \mathbb{Z}$ with $x^2 - a = p^k c$. Coprimality supplies Bézout data for $2x$ and p, so theorem 6.1 gives t with $p \mid c + 2xt$. Since $k \geq 1$, one has $p \mid p^k$. Apply theorem 6.2 with $m = p^k$. □

Formal counterpart: `Arithmetic.quadratic_hensel_step`.

The formal hypothesis is `IsCoprime (2 * x) p`, expressed by Bézout data. The theorem constructs an integer representative of a correction class. It does not state that this representative is already the unique digit between zero and $p - 1$, nor that an infinite succession of such choices has been constructed in a complete field.

6.3 An exact seven-adic trace

For $x^2 = 2$ at $p = 7$, start with $x_1 = 3$. At each step, let

$$c_k = \frac{x_k^2 - 2}{7^k}, \qquad c_k + 2x_k t_k \equiv 0 \pmod 7, \qquad x_{k+1} = x_k + 7^k t_k.$$

The following table records normalized choices for this example. Every displayed digit equation is an exact integer calculation.

k	x_k	$c_k \bmod 7$	t_k	x_{k+1}
1	3	1	1	10
2	10	2	2	108
3	108	6	6	2,166
4	2,166	1	1	4,567
5	4,567	2	2	38,181
6	38,181	1	1	155,830

All representatives are congruent to three modulo seven, so $2x_k \equiv 6 \pmod 7$ remains a unit. The formal statement `Arithmetic.hensel_seven_trace` verifies the seven congruences

$$7^k \mid x_k^2 - 2 \qquad (1 \leq k \leq 7)$$

for the displayed representatives. It is a finite conjunction of exact divisibility facts. The general one step theorem is proved separately.

For example,

$$10^2 - 2 = 98 = 7^2 \cdot 2,$$

and the next digit satisfies $2 + 2 \cdot 10 \cdot 2 = 42$. Hence

$$108 = 10 + 7^2 \cdot 2, \qquad 108^2 - 2 = 7^3 \cdot 34.$$

The new digit corrects the residual quotient at precisely the next required modulus.

6.4 Why a singular derivative is inconclusive

Proposition 6.4 (A modular solution that does not lift)**.** The equation $x^2 = 2$ has no solution in $\mathbb{Z}/4\mathbb{Z}$.

Proof. The four residue classes have squares $0, 1, 0, 1$, respectively. None has square two. □

Formal counterpart: `Arithmetic.singular_mod_four_impossible`.

Nevertheless, $x = 0$ solves the equation modulo two. Its two possible lifts modulo four are zero and two, and both fail. Thus existence at one precision is not itself an existence theorem at the next.

Proposition 6.5 (Singularity does not prohibit lifting)**.** For every $k \in \mathbb{N}$,

$$2^k \mid 1^2 - 1.$$

Proof. The residual is zero, which is divisible by every integer. □

Formal counterpart: `Arithmetic.singular_derivative_can_still_lift`.

For the equation $x^2 - 1 = 0$, the derivative at $x = 1$ is two and is not a unit modulo two. The exact root still persists at every dyadic precision. A unit derivative is a sufficient hypothesis for theorem 6.3, not a necessary condition for every possible lift.

CHAPTER 7

Nonlinear limits and native realizations

7.1 A linearized problem is a different equation

Suppose a nonlinear map F is differentiable at an approximation x_0. Its linearized correction equation is

$$DF(x_0)u = -F(x_0). \tag{7.1}$$

The cokernel criterion completely decides (7.1) once the derivative is fixed. It does not, by itself, decide whether there is x with $F(x) = 0$. Higher order terms are absent from the linearized equation, and they may either create a solution or prevent the linear prediction from reaching one.

The next two propositions are not warnings based only on examples computed numerically. The formal source includes actual derivative statements and exact proofs of the contrary conclusions.

Proposition 7.1 (A nonlinear root with a nonzero first order obstruction)**.** For $a \in \mathbb{R} \setminus \{0\}$, define $F_a(x) = x^2 - a^2$. At zero,

$$F_a'(0) = 0,$$

and the correction equation $0 \cdot u = a^2$ has no solution. Nevertheless, $F_a(a) = 0$.

Proof. Differentiation gives $F_a'(x) = 2x$, so the derivative at zero is zero. Since $a \neq 0$, its square is nonzero, while $0 \cdot u = 0$ for every u. Finally, $F_a(a) = a^2 - a^2 = 0$. □

Formal counterpart: `Nonlinear.nonlinear_root_despite_first_order_obstruction`.

The root can lie arbitrarily close to zero by choosing a small nonzero a. This produces a family of nearby counterexamples. It does not assert that one fixed equation with a nonzero residual at zero has roots converging to zero.

Proposition 7.2 (A solvable linearization without a real root)**.** Let $G(x) = x^2 + x + 1$. Then

$$G'(0) = 1, \qquad \exists u \in \mathbb{R}\ (u = -1),$$

so the linearized correction at zero is solvable. However, G has no real root.

Proof. The derivative is $2x + 1$, which equals one at zero, and $G(0) = 1$. Thus $u = -1$ solves the linearized equation. But

$$G(x) = \left(x + \frac{1}{2}\right)^2 + \frac{3}{4} > 0$$

for every real x. □

Formal counterpart: `Nonlinear.linearized_solution_without_real_root`.

7.2 Three conclusions that must stay distinct

The results support three different kinds of certificate.

An exact linear repair provides witnesses for $Du + Ew = -r$ in the original target. Its validity is governed by quotient image membership.

A finite arithmetic lift provides a correction at one modulus and proves the exact divisibility gain. Its validity also uses arithmetic conditions such as Bézout coprimality and a scale divisibility relation.

A nonlinear existence result proves that a solution of the original nonlinear equation exists. It requires an additional theorem controlling the nonlinear terms, compatible successive approximations, or a different existence mechanism.

Paper III develops correction, nonlinear analysis, and filtered lifting under separate hypotheses [7]. The present formal component does not import those complete releases. It proves a restricted linear theory, a finite arithmetic step, and counterexamples that prevent unjustified upgrades between these conclusions.

7.3 The relation to Realization Calculus

The conceptual setting is the established separation between a mathematical object, its chosen realization, and the operations a realization preserves. Paper I's semantic realization criterion requires an explicit construction, preservation of the operations used in the proof, verification of additional domain hypotheses, and a conclusion stated in native objects [6].

Constructive repair fits that criterion in a direct way. The native equation supplies V and its residuals. The allowed transformations supply D and the permissible class of added maps. The quotient organizes which residual classes those maps can remove. A transport that is used to decide the problem must preserve the necessary obstruction data.

Komorebi Space is the proposed ambient setting for organizing realizations, compatibility data, residuals, and obstructions. This monograph supplies a bounded repair construction that can be placed in that setting. It does not prove that every possible realization in Komorebi Space has a canonical finite obstruction space or an optimal repair.

A successful application must still identify the formal data with the actual objects of its domain. For example, an arithmetic representation with prescribed character and phase information cannot be obtained merely by naming an abstract linear space an arithmetic carrier. The Paper XI research companion distinguishes finite comparison from arithmetic existence and develops specific enlarged carrier constructions under stated hypotheses [8]. Those constructions motivate the repair viewpoint, but are not reproduced or newly formalized here.

7.4 Established reach and remaining questions

The finite theory has a clear end point. It proves exactly when an added linear map repairs a family, the least possible added dimension over a field, optimal attainment inside a feasible legal subspace,

complete finite menu feasibility, and a minimum cost search over finite domains. Its transport theorem excludes certificates that work only by collapsing the relevant quotient information.

The rational matrix bridge in Chapters 8 and 9 now connects an exact Python construction to a small formal verifier. Further questions remain separate. Equivariant, filtered, topological, and arithmetic repairs need admissibility classes more structured than a subspace. Nonlinear minimum repair needs conditions that do not confuse a derivative obstruction with global nonexistence. Infinite lifting needs compatible choices and a convergence theorem.

These are extensions of a precise contract rather than reasons to enlarge its present claims. The elementary linear algebra and arithmetic mechanisms are classical. The contribution of this component is their explicit repair interface, formal proofs, certified finite search, and preservation boundaries. No claim of a universally complete discovery procedure or historical priority follows from the formal theorem count.

> **A usable stopping point**
>
> A certificate must state which equation it solves, which changes are allowed, and which information is preserved. Within the declared finite linear setting, the theorems provide both successful repairs and complete rejection of the supplied menu. Outside that setting, the missing domain hypotheses remain mathematical obligations.

CHAPTER 8

Exact rational certificates

8.1 Let the search propose and the kernel check

The finite optimizer in Chapter 5 enumerates finite correction domains. Rational coefficients do not form such a domain. The appropriate interface is therefore not an enumeration of all rational assignments. A solver proposes finite rational matrices, and a verifier checks their identities in the original target.

Fix matrices

$$D \in \mathbb{Q}^{m\times n}, \qquad R \in \mathbb{Q}^{m\times s}, \qquad E \in \mathbb{Q}^{m\times q}.$$

The residuals are the columns of R. For a selected menu S, let $\widetilde{E}_S$ retain the original candidate indexing and replace unselected columns by zero. The supplied coefficient matrices have shapes $U \in \mathbb{Q}^{n\times s}$ and $W \in \mathbb{Q}^{q\times s}$. Their correction equation is

$$DU + \widetilde{E}_S W = -R. \tag{8.1}$$

Keeping the original indexing avoids any ambiguity about a permutation of candidate columns. Coefficients on a zeroed column have no effect. The admissibility predicate and cost function belong to the original problem, not to the certificate producer.

Proposition 8.1 (Successful certificate)**.** If an admissible S and rational matrices U, W satisfy (8.1), they repair every original residual.

Proof. Take the ith column of the displayed identity. It is the original equation $Du_i + \sum_{j\in S} w_{ji} e_j = -r_i$. □

Formal counterpart: `RationalCertificates.checkRepair_sound`.

> **A rejected certificate is not a rejected problem**
>
> Failure to parse data, a false identity, a missing witness, or a resource limit is an unresolved verification outcome. It does not establish mathematical infeasibility. A rejection of the problem requires a valid obstruction certificate with an explicit scope.

8.2 Separating obstructions

Definition 8.2 (Separating row)**.** A separating certificate for S is a row $Y \in \mathbb{Q}^{1\times m}$ satisfying

$$YD = 0, \qquad Y\widetilde{E}_S = 0, \qquad YR \neq 0. \tag{8.2}$$

The last condition asks that at least one original residual is detected. No residual index needs to be selected when the family is empty.

Theorem 8.3 (Sound separation)**.** A separating certificate rules out a repair by the selected menu.

Proof. Multiplying a supposed correction equation on the left by Y gives $0 = -YR$, contradicting (8.2). □

Formal counterpart: `RationalCertificates.annihilator_kills_residual`.

Formal counterpart: `RationalCertificates.checkSeparator_sound`.

Proposition 8.4 (Linear alternative)**.** For a fixed rational menu, either the correction matrices exist or a separating row exists, but not both.

Proof. Exclusivity follows from the preceding theorem. If no correction matrices exist, some residual r_i lies outside the column space of $[D\ \widetilde{E}_S]$. Choose a basis for that space, adjoin r_i, and extend to a basis of $\mathbb{Q}^m$. The functional taking value one on r_i and zero on all other chosen basis vectors is a rational separating row. □

This completeness argument is an ordinary proof. The new Lean module proves certificate soundness, not a general algorithm or formal theorem producing a separator for every infeasible rational menu.

8.3 Dimension certificates without a reported rank

Suppose an augmentation with t columns succeeds. Its optimality can be certified by two additional matrices

$$H \in \mathbb{Q}^{t\times m}, \qquad Q \in \mathbb{Q}^{s\times t}$$

with

$$HD = 0, \qquad HRQ = I_t. \tag{8.3}$$

These identities make a lower bound on the required dimension explicit.

Theorem 8.5 (Dimension lower certificate)**.** Assume (8.3). If an arbitrary competing augmentation $E' \in \mathbb{Q}^{m\times w}$ has coefficients satisfying $DU' + E'W' = -R$, then $t \le w$.

Proof. Multiplication by H and Q gives $HE'W'Q = -I_t$, hence

$$(HE')(-W'Q) = I_t.$$

Thus $HE' : \mathbb{Q}^w \to \mathbb{Q}^t$ has a right inverse. It is surjective, and consequently $t \le w$. □

Formal counterpart: `RationalCertificates.dimension_certificate_right_inverse`.

Formal counterpart: `RationalCertificates.dimension_certificate_lower_bound`.

A successful augmentation with t columns attains this bound. The verifier checks the original repair identity and (8.3). It does not trust a numerical rank supplied by the search program.

Formal counterpart: `RationalCertificates.checkDimension_sound`.

Proposition 8.6 (Availability of lower certificates)**.** Let O be the span of the residual classes in $\mathbb{Q}^m / \operatorname{im} D$, and let $t = \dim O$. Then matrices H, Q satisfying (8.3) exist.

Proof. Choose coordinates $O \cong \mathbb{Q}^t$, extend that map to the whole quotient, and compose with the quotient map to obtain H. Then $HD = 0$. The columns of HR span $\mathbb{Q}^t$, so HR has a linear right inverse Q. □

The availability argument is an ordinary finite-dimensional proof. The concrete producer in the next chapter finds and exports such data. Its search implementation is not itself proved correct in Lean.

8.4 Exact rank and menu cost

For $A \in \mathbb{Q}^{a \times b}$, a certificate of rank d consists of appropriately shaped rational matrices with

$$A = FG, \qquad PAQ = I_d, \tag{8.4}$$

where the inner dimension of FG is d.

Proposition 8.7 (Rank certificate)**.** The identities (8.4) imply $\operatorname{rank} A = d$.

Proof. The factorization gives $\operatorname{rank} A \le d$. The identity factorization through A gives $d \le \operatorname{rank} A$. □

Formal counterpart: `RationalCertificates.checkRank_sound`.

For comparison with the original quotient statement, ordinary rank-nullity gives

$$\dim O = \operatorname{rank}[D \; R] - \operatorname{rank} D. \tag{8.5}$$

Indeed, the quotient map restricted to $\operatorname{im} D + \operatorname{im} R$ has kernel $\operatorname{im} D$ and image O. The current matrix verifier does not depend on a formal identification of this formula with the earlier abstract Lean quotient API. It obtains the lower bound directly from a right inverse.

Minimum menu cost is a separate optimization problem. To certify a successful menu S, the verifier requires a separating row for every admissible menu T with strictly smaller declared cost. It quantifies over all subsets of the original finite candidate universe. It does not trust an incomplete producer-supplied list.

Theorem 8.8 (Certified finite-menu optimum)**.** An admissible successful menu, together with separating rows for every cheaper admissible menu, has minimum cost among all admissible feasible menus.

Proof. A cheaper feasible menu would contradict its supplied separating row by Theorem 8.3. □

Formal counterpart: `RationalCertificates.checkOptimal_sound`.

A certificate rejecting every admissible subset proves that no menu in the declared universe succeeds.

Formal counterpart: `RationalCertificates.checkNoRepair_sound`.

The rational example $D(u) = (u, 0, 0)$, $r = (0, 1, 0)$, $e_0 = (0, 1, 1)$, $e_1 = (0, 0, 1)$ is checked in the new matrix layer. Both entries repair the residual with coefficients $(-1, 1)$. Rows $(0, 1, 0)$, $(0, 1, -1)$, and $(0, 1, 0)$ separate the empty menu, the first singleton, and the second singleton. Thus the minimum cardinality is two. Restricting admissibility to at most one entry gives a checked complete rejection.

Formal counterpart: `RationalMenus.menu_certificate`.

Formal counterpart: `RationalMenus.two_entries_minimal`.

Formal counterpart: `RationalMenus.restricted_certificate`.

Formal counterpart: `RationalMenus.no_small_menu`.

The subset search may be exponential in menu size. The free augmentation exporter developed next does not implement a scalable menu optimizer, and the existence of a free minimum says nothing about which directions occur in a prescribed menu.

CHAPTER 9

Constructive matrix attainment

9.1 A complete detector

Fix $D \in \mathbb{Q}^{m\times n}$ and $R \in \mathbb{Q}^{m\times s}$. A split correction certificate consists of

$$P \in \mathbb{Q}^{n\times m}, \quad K \in \mathbb{Q}^{m\times d}, \quad L \in \mathbb{Q}^{d\times m}$$

with

$$LD = 0, \qquad DP + KL = I_m. \tag{9.1}$$

The first identity makes L a detector that ignores original corrections. The second proves that no further information has been discarded: every target vector has the decomposition $v = DPv + KLv$. There is no assumption that $LK = I_d$, and redundant detector coordinates are permitted.

Theorem 9.1 (Exact detection)**.** Under (9.1),

$$(\exists X,\ DX = -R) \quad \Longleftrightarrow \quad LR = 0.$$

When $LR = 0$, the explicit witness is $X = -PR$.

Proof. A correction gives $0 = LDX = -LR$. Conversely, $LR = 0$ gives $R = (DP + KL)R = DPR$, so $D(-PR) = -R$. □

Formal counterpart: `RationalCertificates.split_correctable_iff`.

At the quotient level, L induces an injection $\mathbb{Q}^m/\operatorname{im} D \to \mathbb{Q}^d$. If $Lv = 0$, the split identity gives $v = DPv$, proving that its class is zero. The image is $\operatorname{im} L$. Hence the required residual obstruction span is isomorphic to the column space of $C = LR$. This quotient interpretation is an ordinary consequence of the displayed identities.

9.2 The attaining construction

Suppose the detected residual matrix $C = LR$ has certificate data

$$F \in \mathbb{Q}^{d\times t}, \quad G \in \mathbb{Q}^{t\times s}, \quad J \in \mathbb{Q}^{t\times d}, \quad Q \in \mathbb{Q}^{s\times t}$$

satisfying

$$LR = FG, \qquad JLRQ = I_t. \tag{9.2}$$

The new augmentation and its correction coefficients are not unspecified witnesses. They are

$$\boxed{E = KF, \qquad U = -PR, \qquad W = -G.} \tag{9.3}$$

Theorem 9.2 (Constructive minimum augmentation)**.** Assume (9.1) and (9.2). The matrices in (9.3) satisfy $DU + EW = -R$. Every competing rational augmentation repairing the same residual family has at least t added dimensions.

Proof. The constructed matrices satisfy

$$DU + EW = -DPR - KFG = -DPR - KLR = -(DP + KL)R = -R.$$

For the lower bound set $H = JL$. Then $HD = 0$ and $HRQ = I_t$. Apply Theorem 8.5 to every competing repair. □

Formal counterpart: `RationalCertificates.split_constructs_repair`.

Formal counterpart: `RationalCertificates.checkOptimalConstruction_sound`.

This proves attainment and minimality after checking four finite identities. The generalized inverse and the choice of a residual basis belong to the producer. They are not assumed trustworthy. A valid set of identities is sufficient, regardless of how the producer found it.

9.3 How the Python producer obtains the matrices

The executable producer uses exact rational row reduction. It selects independent columns of D and extends them by standard basis vectors to a basis B of $\mathbb{Q}^m$. The inverse B^{-1} supplies the image coordinates and the complementary detector coordinates. The selected image coordinates are placed in the original domain to obtain P, while the complementary basis vectors form K and the corresponding rows of B^{-1} form L.

For $C = LR$, the producer selects independent residual columns to form F. A selection matrix Q records those columns. A left inverse J of F gives $G = JC$. The producer then forms E, U, W using (9.3), and checks every identity again with exact rational arithmetic before export.

All of these data can be obtained over a field. The search implementation may nevertheless contain a bug, encounter a resource limit, or reject malformed input. Such events do not become mathematical no-go results. The Lean certificates verify successful outputs, not the total correctness or complexity of this search program.

Zero-dimensional targets, original domains, obstruction spaces, and residual families are retained. For an already correctable residual family the construction has $t = 0$, so it adds no directions. The input limit of the supplied prototype is an implementation resource bound, not a restriction on Theorem 9.2.

9.4 A full intermediate trace

Take

$$D = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \qquad R = \begin{pmatrix} 2 & 4 \\ 1 & 2 \\ 1 & 2 \end{pmatrix}.$$

The producer selects the first column of D and the complementary second and third coordinate axes. It obtains

$$P = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}, \quad K = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad L = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Thus $LD = 0$, $DP = \operatorname{diag}(1, 0, 0)$, and $KL = \operatorname{diag}(0, 1, 1)$. The residual coordinates are

$$C = LR = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}.$$

A rank-one certificate is

$$F = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad G = \begin{pmatrix} 1 & 2 \end{pmatrix}, \quad J = \begin{pmatrix} 1 & 0 \end{pmatrix}, \quad Q = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

The result is

$$E = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \qquad U = \begin{pmatrix} -2 & -4 \end{pmatrix}, \qquad W = \begin{pmatrix} -1 & -2 \end{pmatrix}.$$

The original equations become

$$DU + EW = \begin{pmatrix} -2 & -4 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ -1 & -2 \\ -1 & -2 \end{pmatrix} = -R.$$

The lower witness is $H = JL = (0, 1, 0)$, with $HD = 0$ and $HRQ = 1$. The ambient cokernel has dimension two, but the residual classes span only one direction. The construction therefore adds one direction and proves that zero added directions are insufficient.

Formal counterpart: `Case000.certificate_checked`.

Formal counterpart: `Case000.original_equations_checked`.

Formal counterpart: `Case000.optimal`.

The fractional case uses $D = (2/3, 4/3)^{\mathsf{T}}$ and $R = (1/2, 5/6)^{\mathsf{T}}$. It exercises nonintegral splitting and correction coefficients while retaining exact arithmetic. Empty and rank-deficient cases are included in the same exported corpus. Every case has its own original input file, certificate file, and formal optimality statement.

9.5 Serialization and input identity

The wire format represents a rational entry by a signed decimal numerator and a positive decimal denominator. The Python decoder rejects floats, nonfinite values, duplicate object keys, zero or negative denominators, unreduced pairs, malformed dimensions, and extraneous certificate fields. Resource limits produce an invalid or unresolved outcome, not an infeasibility theorem.

The generated Lean data use typed pairs in $\mathbb{Z} \times \mathbb{N}$. Their decoder returns the intended rational when the denominator is nonzero and explicitly rejects a zero denominator at the cell level. Each generated input and certificate additionally proves denominator validity for all its cells.

Formal counterpart: `RationalCertificates.decodeCell_zero`.

Formal counterpart: `RationalCertificates.decodeCell_nonzero`.

Formal counterpart: `RationalCertificates.decodeCell_sound`.

Formal counterpart: `RationalCertificates.checkWireMatrix_sound`.

Formal counterpart: `RationalCertificates.decodeWireMatrix_cell`.

The original matrices D, R are emitted separately from certificate data. The certificate module imports them and cannot replace them through a certificate field. A canonical problem hash binds the JSON records. A diagnostic Lean echo of the decoded input matrices is compared with the original JSON values for every exported case. That echo is an additional execution check, never a theorem premise.

The precise trust boundary

Lean checks the matrix identities, the original equations, and the minimum-dimension conclusion for the literal decoded problem in the generated input module. The numeric cell-decoding lemmas are formalized. The whole JSON parser, file system, hashing routine, and source emitter are not proved correct in Lean. Their input binding is checked by hashes, strict decoding, and a separate Lean echo. A user must still identify the intended original problem correctly.

No theorem is accepted through `native_decide`. Concrete certificate propositions are proved with `decide +kernel`, and every exported theorem is included in the transitive axiom audit.

9.6 Legality and coefficient rings

Free minimum dimension is not minimum menu cost. The constructed direction KF need not occur among supplied menu entries. When legal directions form $\operatorname{im} A$, an additional identity $KF = AZ$ certifies that the augmentation lies in that legal subspace. This identity also rewrites every added correction as $A(ZW)$.

Formal counterpart: `RationalCertificates.constructed_repair_in_legal_image`.

The availability of splitting also depends on the coefficient ring. For $D = (2)$ over $\mathbb{Z}$, the condition $LD = 0$ forces $L = 0$. The split identity would then require $2P = 1$, which is impossible over $\mathbb{Z}$. Over $\mathbb{Q}$, $P = 1/2$ is permitted. The current producer and checker are rational, not a certificate of integral realizability.

Finally, exact rational correction is not nonlinear convergence or arithmetic representation-theoretic existence. The existing boundaries of Paper III and Conjecture 66 are unchanged. The contribution of this extension is a usable and checked witness interface inside the established calculus, not a replacement for its foundations.

CHAPTER 10

Costed rational menus and rejection covers

The free augmentation problem asks how many new dimensions are necessary. A prescribed menu asks a different question: which available directions should be purchased, enabled, or admitted? Two menu entries can be needed for a one-dimensional obstruction, and the cheapest solution need not use the fewest entries. This chapter makes that second problem operational over the rationals.

The implementation uses exact elimination to propose an answer. A separate certificate checker verifies the answer against the original matrices and the complete admissibility rule. The search procedure is outside the proof trust boundary. The returned mathematical witnesses are not.

10.1 The original menu contract

Fix rational matrices

$$D \in \mathbb{Q}^{m\times n}, \qquad E \in \mathbb{Q}^{m\times q}, \qquad R \in \mathbb{Q}^{m\times s}.$$

The columns e_j of E are the available new directions. The finite family

$$\mathcal{A} \subseteq \mathcal{P}(\{0,\dots,q-1\})$$

contains exactly the admissible selections. It need not be downward closed. Nonnegative integer weights c_j define the cost

$$c(S) = \sum_{j\in S} c_j.$$

These matrices, selections, and costs are fixed original input. A certificate cannot replace them with another problem.

Let $\widetilde{E}_S$ retain each column of E selected by S and replace every other column by zero. This convention preserves the original candidate indexing. A coefficient table $W \in \mathbb{Q}^{q\times s}$ may have arbitrary values in unselected rows, but those rows contribute nothing. A proposed repair consists of matrices U, W with

$$DU + \widetilde{E}_S W = -R. \tag{10.1}$$

Every selected direction is charged, even if its coefficient row happens to be zero. Thus cost measures the selected menu, not merely the support of a particular coefficient table.

10.2 Separating blocks

An infeasible menu admits a rational separating row, by the exact linear alternative in chapter 8. The row annihilates all available corrections while detecting at least one residual. Instead of providing a separate row for every rejected subset, we can reuse one row on all subsets of a larger rejected set.

Definition 10.1 (Separating block)**.** A separating block is a pair (B, Y), where $B \subseteq \{0, \dots, q-1\}$ and $Y \in \mathbb{Q}^{1\times m}$ satisfy

$$YD = 0, \qquad Y\widetilde{E}_B = 0, \qquad YR \neq 0. \tag{10.2}$$

The set B need not itself belong to $\mathcal{A}$.

The last condition is a matrix inequality: at least one column is detected. For an empty residual family it cannot hold, which correctly prevents a false separating witness for a vacuously solved family.

Lemma 10.2 (Column-mask identity)**.** For every row Y and column j,

$$(Y\widetilde{E}_S)_{0j} = \begin{cases} (YE)_{0j}, & j \in S, \\ 0, & j \notin S. \end{cases}$$

Proof. The mask either retains the entire jth column or replaces it by zero. Expanding the finite matrix product gives the two cases. □

Formal counterpart: `KomorebiRepair.RationalCertificates.mul_masked_entry`.

Theorem 10.3 (Downward validity of a separator)**.** If (B, Y) is a separating block and $T \subseteq B$, then Y also separates T. In particular, T cannot satisfy (10.1).

Proof. For a selected column $j \in T$, one also has $j \in B$, so $(YE)_{0j} = 0$ by the block identity. Every unselected column is zero after masking. Thus $Y\widetilde{E}_T = 0$. The other two equations in (10.2) are unchanged.

If a repair for T existed, multiplication by Y would give

$$0 = YDU + Y\widetilde{E}_T W = -YR,$$

contradicting the detection condition. □

Formal counterpart: `KomorebiRepair.RationalCertificates.separates_subset`.

The restriction to an admissible family does not invalidate this argument. A block can be a forbidden superset while still supplying valid mathematical evidence against its permitted subsets. What matters is that the block's own linear impossibility has been checked.

10.3 Complete coverage of cheaper alternatives

Let $\mathcal{B}$ be a finite list of separating blocks. Say that $\mathcal{B}$ covers T when some $(B, Y) \in \mathcal{B}$ satisfies $T \subseteq B$. Validity of the block list and coverage of the relevant selections are separate obligations.

Theorem 10.4 (Certified minimum menu cost)**.** Suppose the following data have been checked:

1. $S \in \mathcal{A}$ and the original repair equation holds for S, U, W.
2. Every member of $\mathcal{B}$ is a valid separating block.
3. Every $T \in \mathcal{A}$ with $c(T) < c(S)$ is covered by $\mathcal{B}$.

Then S has minimum cost among all admissible feasible selections.

Proof. The first condition supplies an admissible feasible selection. If a cheaper admissible feasible T existed, the third condition would place it inside a separating block. Downward validity would make T infeasible, a contradiction. □

Formal counterpart: `KomorebiRepair.RationalCertificates.checkCoveredOptimal_sound`.

The checker quantifies over the whole finite candidate-subset universe when it verifies coverage. It does not accept a producer's claim that its own list of alternatives is complete. Equal-cost alternatives need not be rejected, and the certificate does not claim that the optimum is unique.

Theorem 10.5 (Certified rejection of the admissible family)**.** If every block is valid and every member of $\mathcal{A}$ is covered, then no admissible menu repairs the residual family.

Proof. Every admitted selection lies inside a valid separating block and is therefore infeasible. □

Formal counterpart: `KomorebiRepair.RationalCertificates.checkCoveredNoRepair_sound`.

When $\mathcal{A}$ is empty, this conclusion is vacuous. The implementation reports that fact explicitly. It does not manufacture a nonzero obstruction or claim that the original equations have no solution without the admissibility restriction.

> **The scope of rejection**
>
> An accepted rejection certificate rules out the declared admissible menus. It does not rule out a new direction outside the menu, a different coefficient ring, a changed admissibility rule, or a different arithmetic host. A failed check or exhausted time budget establishes none of these impossibility claims.

10.4 Constructing the evidence

For each candidate subset, the producer performs exact rational row reduction on the selected correction columns. It keeps the row transformation matrix as part of the computation. If the transformed residual has a nonzero entry in a zero coefficient row, that row transformation supplies

a separating functional. Otherwise the pivot equations give a coefficient table for every residual. Each proposed identity is rechecked using exact arithmetic.

The producer evaluates all 2^q selections in this bounded implementation. It chooses a feasible admitted selection of minimum declared cost, with a deterministic tie break used only for reproducibility. Alternatively, it records that none of the admitted selections passed the exact linear alternative.

For rejection evidence, the producer finds inclusion-maximal infeasible subsets of the full candidate universe. Every infeasible selection is contained in such a maximal subset because the universe is finite. A separating row for each selected maximal subset therefore provides a complete supply of potential rejection blocks. The producer then greedily retains blocks that cover the required alternatives.

The greedy compression is not claimed to minimize certificate size. Nor does a smaller rejection cover imply a faster search algorithm: the present producer still examines every subset. The verified conclusion concerns menu cost and the original equations, not optimal compression or polynomial-time complexity.

10.5 Two entries for one missing dimension

Take

$$D = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \qquad E = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}, \qquad R = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix},$$

with both candidate weights equal to one and every subset admitted. The chosen menu is $S = \{0, 1\}$, with

$$U = (0), \qquad W = \begin{pmatrix} -1 \\ 1 \end{pmatrix}.$$

Its exact repair is

$$DU + EW = \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix} = -R.$$

Two separating blocks suffice:

$$B_0 = \{0\}, \quad Y_0 = (0, 1, -1), \qquad B_1 = \{1\}, \quad Y_1 = (0, 1, 0).$$

Both satisfy $Y_i D = 0$, annihilate their masked candidate matrices, and give $Y_i R = 1$.

Cheaper menu	Cost	Covering block
$\varnothing$	0	B_0 or B_1
$\{0\}$	1	B_0
$\{1\}$	1	B_1

Thus cost two is minimal. Restricting admissibility to the three cheaper menus turns the same two blocks into a complete no-admissible-repair certificate. The unrestricted one-dimensional repair direction R still exists, but it is not one of the two listed candidate directions.

Executed examples: Menu000 and Menu001. Each has separately emitted original input, exact rational witnesses, a Lean certificate proof, an axiom audit, and an input echo.

10.6 Cost, dimension, and evidence size

A third candidate can make the cost distinction sharper. Add the direct direction $e_0 = (0, 1, 0)$ at cost five, and price $e_1 = (0, 1, 1)$ and $e_2 = (0, 0, 1)$ at one each. One menu entry can now repair the residual, but the cheapest repair selects two entries, $\{1, 2\}$, at total cost two. Minimum free dimension, minimum menu cardinality, and minimum menu cost are three different quantities.

A separate example shows why rejection covers help. Let

$$D = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad E = \begin{pmatrix} 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \qquad R = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

The row $Y = (0, 1)$ separates the full four-entry menu. One block therefore excludes all sixteen subsets. Adding $(0, 1)$ as a new candidate would change the problem and allow a repair. The certified rejection of the original menu would remain correct.

Executed examples: Menu002 distinguishes cost from cardinality. Menu004 excludes all sixteen subsets with one separating block. Menu005 uses one block to cover eight cheaper selections before accepting a positive-cost direction.

10.7 Fractional witnesses and independent original data

The solver is not restricted to coefficients in a finite field. For

$$D = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad E = \begin{pmatrix} 0 \\ 2/3 \end{pmatrix}, \quad R = \begin{pmatrix} 1/2 \\ 5/6 \end{pmatrix},$$

the selected direction at cost seven has witnesses

$$U = -1/2, \qquad W = -5/4.$$

The checker verifies the exact rational equation, with no tolerance or floating-point rank test.

The generated input module contains D, E, R, the candidate weights, and the literal admissibility table. The certificate module imports those inputs and supplies only coefficient matrices and separating blocks. Typed rational decoding and nonzero denominator proofs are retained from the preceding bridge.

An independent diagnostic echo evaluates the matrices in Lean and evaluates the admissibility and cost of every candidate subset. Python compares those evaluated results with the original JSON input. This comparison catches mismatched original constraints as well as mismatched matrices. It is an executed statement-binding check, not a formal proof of the entire external JSON parser or source emitter.

10.8 Bounded executable use

The command `scripts/certify_menu.py` creates a new local directory, constructs or reads a certificate, emits the fixed input and witness modules, compiles their proofs, audits their axioms, replays the modules, and checks the original-data echo. Existing output directories are refused.

```
export KOMOREBI_LEAN_KIT=/path/to/Komorebi_Portable_Lean_4.34.0-rc2
python scripts/certify_menu.py examples/menu_corpus/Menu003_input.json --output
↪  /tmp/repair-menu
```

The successful mathematical outcomes are `KERNEL_CHECKED_OPTIMAL_MENU` and `KERNEL_CHECKED_NO_ADMISSIBLE_MENU`. Invalid data, unsupported sizes, compiler failure, and timeouts produce `UNRESOLVED`. A Python-valid certificate alone is not a kernel-checked result.

This version limits the menu to eight candidates. Other input bounds include thirty-two rows or columns, canonical rational cells with bounded decimal length, and a four-megabyte JSON file limit. These are implementation safeguards, not hypotheses of the finite mathematical theorems. Exhaustive search is still exponential in menu size, and the time limit can be reached within the admitted size bounds.

The module is a constructive extension of the correction framework already developed in Realization Calculus. It gives Komorebi Space a more explicit candidate interface for admissible finite linear repairs, but supplies no universal semantic-realization theorem. Integration into the authoritative Paper III tree and the established Komorebi Python library remains separate from this isolated, reproducible implementation.

CHAPTER 11

Proof carrying branch search

The preceding menu verifier checks every required selection against a family of separating blocks. That is a sound endpoint, but its coverage test still ranges over the full subset universe. A large menu may therefore be expensive to verify even when one short argument excludes almost every alternative.

This chapter replaces the powerset coverage test with a finite search tree. Each node describes an interval of selections. A terminal node either proves a cost lower bound, supplies one separator for the entire interval, or checks that the original admissibility rule excludes that interval. A branching node must retain both possible decisions about one unresolved candidate.

The producer may search in any order and use any heuristic. The verifier checks the submitted tree. It does not trust the producer's account of which alternatives were examined.

> **The new verified boundary**
>
> The mathematical result is soundness of a finite rational certificate tree. It certifies minimum cost within the declared menu or infeasibility of the declared admissible family. It is not a proof of correctness or termination of the entire Python producer, a polynomial complexity result, or a native arithmetic realization theorem.
> All 37 earlier proof modules are retained unchanged. The new module reuses their exact rational repair and separator theorems.

11.1 Intervals of candidate selections

Let $J = \{0, \dots, q-1\}$ be the original candidate index set. The matrices D, E, R and the original correction equation remain fixed:

$$DU + \widetilde{E}_S W = -R.$$

Here $\widetilde{E}_S$ zeros unselected columns of E while retaining its original indexing. Costs are nonnegative integers attached to those same columns:

$$c(S) = \sum_{j \in S} w_j, \qquad w_j \in \mathbb{N}.$$

The admissibility rule either permits every subset or supplies a fixed finite table of permitted subsets. An empty table permits none. It is different from the all-subsets rule.

Definition 11.1 (Selection interval)**.** For $A \subseteq B \subseteq J$, define

$$\mathcal{I}(A, B) = \{T \subseteq J : A \subseteq T \subseteq B\}.$$

The set A contains candidates already required, and B contains all candidates still permitted by the branch decisions. The unresolved indices form $B \setminus A$.

Lemma 11.2 (Exact branch partition)**.** If $j \in B \setminus A$, then

$$\mathcal{I}(A, B) = \mathcal{I}(A, B \setminus \{j\}) \sqcup \mathcal{I}(A \cup \{j\}, B).$$

Proof. Every $T \in \mathcal{I}(A, B)$ either excludes j or includes j. In the first case it belongs to the left interval, and in the second it belongs to the right interval. Conversely, each child interval is contained in the parent. Their membership conditions on j are incompatible, so the union is disjoint. □

This identity is the completeness mechanism of the verifier. A certificate cannot simply omit an inconvenient branch. Its split constructor contains both children, and the checker verifies both. The proof of `checkBranch_sound` applies the two cases directly.

11.2 Three terminal certificates

Cost lower bound

Nonnegative additive costs are monotone under inclusion:

$$A \subseteq T \quad \Longrightarrow \quad c(A) \leq c(T).$$

Therefore, when an incumbent repair has cost c_*, the finite check

$$c_* \leq c(A)$$

excludes every strictly cheaper repair in $\mathcal{I}(A, B)$. It does not say the interval is infeasible. It only says that it cannot improve the incumbent.

Formal counterpart: `KomorebiRepair.RationalCertificates.additiveCost_mono`.

Proof. The sum for $c(T)$ contains all terms in the sum for $c(A)$ and only additional nonnegative terms. Apply transitivity to $c_* \leq c(A) \leq c(T)$. □

A cost leaf is forbidden in an impossibility certificate. Even a very expensive feasible repair refutes a claim that no repair exists. The implementation expresses this distinction using an optional threshold: a supplied natural number requests a lower bound, while no threshold requests infeasibility.

Separator for the upper selection

A row Y is checked against the largest selection B in the interval:

$$Y D = 0, \qquad Y \widetilde{E}_B = 0, \qquad Y R \neq 0.$$

Every $T \subseteq B$ uses only columns annihilated by Y. Consequently, the same row proves that no $T \in \mathcal{I}(A, B)$ repairs the family. This reuses the downward separator theorem from the previous chapter.

The residual matrix in the last inequality is the original R. Projecting it to zero in some unrelated representation is not accepted as a repair.

Exclusion by the original admissibility rule

If admissibility is specified by a list $\mathcal{A}$, a terminal check establishes

$$\nexists T \in \mathcal{A} : \quad A \subseteq T \subseteq B.$$

It examines the original list, not the powerset of J. For the all-subsets rule this terminal certificate is unavailable.

Formal counterpart: `KomorebiRepair.RationalCertificates.intervalExcluded_sound`.

It is not enough that A itself be inadmissible. A larger selection in the interval might be permitted. For example, a rule that permits only the pair $\{0, 1\}$ excludes the empty selection but not the interval from the empty selection to the full pair. The verifier retains this distinction and does not assume that admissibility is downward closed.

11.3 Soundness of the whole tree

Definition 11.3 (Accepted branch tree)**.** A tree for $\mathcal{I}(A, B)$ consists of one of the three terminal certificates above, or a split at $j \in B \setminus A$ with accepted trees for both child intervals. In lower-bound mode a cost threshold is fixed throughout the tree. In infeasibility mode cost leaves are rejected.

Theorem 11.4 (Structural certificate soundness)**.** An accepted lower-bound tree with threshold c_* proves

$$T \in \mathcal{I}(A, B), \quad T \text{ admissible and feasible} \quad \Longrightarrow \quad c_* \leq c(T).$$

An accepted infeasibility tree proves that no admissible selection in $\mathcal{I}(A, B)$ is feasible.

Formal counterpart: `KomorebiRepair.RationalCertificates.checkBranch_sound`.

Proof. Induct on the supplied tree. For a cost leaf, monotonicity gives the lower bound. For a separator leaf, the exact linear alternative rules out feasibility of every subset of the upper selection. For an admissibility leaf, the original table rules out the premise.

At a split, the exact branch partition puts any proposed selection in one child interval. Apply that child's induction hypothesis. Both children are present in the data and checked by the verifier. This proves the assertion without enumerating every selection in the parent interval. □

Corollary 11.5 (Minimum cost certificate)**.** Suppose an admissible selection S has supplied coefficients U, W satisfying the original repair equation. An accepted lower-bound tree of threshold $c(S)$ on $\mathcal{I}(\varnothing, J)$ proves that S has minimum cost among all admissible feasible selections.

Formal counterpart: `KomorebiRepair.RationalCertificates.checkTreeOptimal_sound`.

Proof. The supplied coefficient equation proves feasibility of S. Every selection lies in $\mathcal{I}(\varnothing, J)$. Structural soundness therefore gives $c(S) \leq c(T)$ for every admissible feasible competitor T. □

Corollary 11.6 (Scoped complete rejection)**.** An accepted infeasibility tree on $\mathcal{I}(\varnothing, J)$ proves that the declared admissible family contains no repair.

Formal counterpart: `KomorebiRepair.RationalCertificates.checkTreeNoRepair_sound`.

There is no claim that directions outside J cannot repair the residual. Nor does a rejected certificate establish infeasibility. A missing child, a bad separator, or an exhausted resource budget leaves the outcome unresolved.

11.4 How the producer constructs evidence

The Python producer traverses intervals and retains exact row-reduction witnesses. It first checks whether the original admissibility table excludes the interval. It then uses the incumbent cost to test a bound leaf. If necessary, it solves the rational linear problem on the upper selection B. An infeasible upper selection supplies a separator that ends the whole branch.

If the lower selection A is admitted and feasible, its coefficient matrix supplies a new incumbent and the interval ends at a cost leaf. Otherwise, the producer selects an unresolved index and explores both decisions. The current heuristic visits low-cost candidates first. Correctness of this heuristic is not an assumption of any theorem.

The incumbent may improve while the search proceeds. Earlier cost leaves remain sound: a later incumbent has no greater cost, so any earlier lower bound also excludes improvement over the final incumbent. The finished checker uses only the final threshold and rechecks every leaf against it. It need not trust the producer's history of incumbents.

Exact linear solves are cached by selection. This avoids repeating an identical solve when several intervals have the same upper or lower selection. Caching is an implementation choice outside the trusted mathematical layer.

11.5 A twenty-four-candidate construction

Let b_1, b_2, b_3 be the standard basis of $\mathbb{Q}^3$. Take $D(u) = ub_1$ and residual $r = b_2 + b_3$. There are 24 candidate entries:

$$a_0 = b_2, \qquad a_1 = b_3, \qquad a_j = b_1 \quad (2 \le j < 24).$$

Their costs are two, three, and ten for each remaining entry. All subsets are admissible.

The selected repair is $S = \{0, 1\}$, with cost five, original coefficient zero, and coefficient -1 on each selected menu direction. Its full coefficient matrix satisfies the original equation exactly.

The proof tree has five nodes. At the root it splits on candidate zero. The excluded child lacks the only second-coordinate correction and has separator $(0, 1, 0)$. The included child splits on candidate one. Excluding candidate one has separator $(0, 0, 1)$. Including both reaches lower selection $\{0, 1\}$ with cost five and terminates at a bound leaf.

Branch	Certified conclusion
Exclude candidate 0	The second coordinate of the residual cannot be corrected.
Include 0, exclude 1	The third coordinate of the residual cannot be corrected.
Include 0 and 1	Every superset costs at least five.

There are $2^{24} = 16{,}777{,}216$ possible selections, but the producer used six distinct linear solves and submitted five tree nodes. Lean checks that tree and proves the minimum-cost conclusion for every selection. This is not an experiment that visited sixteen million cases. It is one structured problem with a short exhaustive mathematical argument.

A second 24-candidate example puts every available correction on the first coordinate axis of $\mathbb{Q}^2$ and sets the residual to $(0, 1)$. The row $(0, 1)$ supplies a one-node certificate rejecting all selections. This illustrates compressed reasoning, not a claim that arbitrary difficult search problems have short certificates.

11.6 Performance and semantic limits

The earlier eight-candidate exhaustive producer remains available as a reference. The new producer accepts up to 24 candidates, with explicit limits on nodes and linear solves. These are implementation resource guards, not mathematical restrictions on the tree theorem.

In a favorable instance, separators or cost bounds eliminate large intervals. In the worst case the tree can still approach the full binary decision tree. Neither polynomial-time search nor a polynomial-size certificate is established. With an explicit admissibility table, checking an exclusion leaf also costs work proportional to that table.

The nonnegative cost requirement is essential to the implemented bound rule. If an optional candidate had negative cost, adding it could lower the cost below the lower selection's cost. The input schema rejects negative weights instead of silently applying an invalid monotonicity argument.

The theorem operates on the original rational matrices and declared subset policy. It does not prove that these finite data capture a native arithmetic, analytic, or representation-theoretic construction. That remains the semantic realization obligation of the domain application. In Komorebi Space, the admissible family must retain its declared meaning. A compressed certificate cannot replace that meaning with a more convenient problem.

APPENDIX A

Formal correspondence

A.1 How the prose matches the source

The original formal source is the supplied *Constructive Repair Lean v0.2.0* release [4]. The namespace is `KomorebiRepair`. The notation Q_D, q_D, $O_D(r)$, and E_S corresponds to `ObstructionSpace`, `obstructionMap`, `obstructionSpan`, and `menuMap`, respectively.

The mathematical statements here use $\exists u, w$ with the same sign convention as the source. Families quantify $\forall i\, \exists u_i, w_i$. They do not assume one shared correction vector. The algebraic base is a commutative ring. The dimension and section results specialize to a field, and finiteness is stated when numerical dimensions or terminating enumeration are used.

Mathematical topic	Principal formal module	Discussion
Correction and repair	`Basic`	Chapter 1
Free minimum	`Minimal`	Chapter 2
Legal subspace	`ConstrainedMinimal`	Chapter 2
Finite menu span	`Menu`	Chapter 3
Invariant obstruction	`Conservation`	Chapter 3
Faithful transport	`Transport`	Chapter 4
Executable search	`Search`	Chapter 5
Quadratic digit step	`Arithmetic`	Chapter 6
Nonlinear boundaries	`Nonlinear`	Chapter 7
Concrete witnesses	`Examples`	Chapters 3, 4, 5

A correspondence records the mathematical content of the formal declaration. It does not make every explanatory sentence or supplementary example a separately compiled theorem. In particular, theorem 2.7, the nonfield section counterexample, the span faithfulness illustration, and the enumeration bound are ordinary derivations given in the text. The source map distinguishes these from named Lean declarations.

A.2 Constructive existence and executable construction

The free minimum uses a linear section of a quotient. Its formal proof invokes a standard vector space splitting theorem and classical choice. The proof establishes an actual mathematical map with the required properties, but does not extract a canonical or efficiently computable representative system.

The finite optimizer is executable for the finite types in its contract. It returns a selected set, not a basis of the obstruction space or a coefficient table. Its logical correctness proves that suitable

correction witnesses exist. The two evaluated finite examples are proved by kernel reduction, while their separate printed outputs serve only as execution traces.

These uses of the word “constructive” should not be merged. The first constructs existence within classical mathematics. The second defines and certifies a terminating finite algorithm. Chapters 8 and 9 now supply a separate rational certificate checker and an explicit matrix construction. This is not a proof that the earlier Python optimizer and finite-domain Lean solver implement the same search algorithm.

A.3 Stable source locators

Appendix B gives every named theorem in the source registry, its source line, and its role in this monograph. The line numbers refer to the unchanged files in the accompanying original archive. Mathematical numbering in this document is independent of source declaration order.

The register was compared with the actual source files and with the theorem axiom entries in the supplied verification record. All 49 entries are represented. The declaration count includes helper facts and concrete examples. It is not a claim that the document establishes 49 independently novel mathematical discoveries.

APPENDIX B

Original declaration register

Every source path below is relative to the root of the formal release. Prefix theorem names with `KomorebiRepair`. For the `Arithmetic`, `Nonlinear`, and `Examples` modules, retain that additional namespace component. The register is generated from the supplied JSON declaration list and linked to the corresponding mathematical discussion.

B.1 Basic

Source: `lean/KomorebiRepair/Basic.lean`.

`obstructionMap_correction` line 31

Original corrections map to zero in the quotient. See theorem 1.2.

`correctable_iff_obstruction_zero` line 35

Exact correction is equivalent to vanishing of the quotient class. See theorem 1.2.

`repairable_iff_obstruction_mem_range` line 46

Added repair is equivalent to membership in the added quotient image. See theorem 1.4.

`obstruction_mem_span` line 65

Each required residual class belongs to its declared obstruction span. See theorem 1.6.

`family_repairable_iff_span_le` line 69

All family equations are solvable exactly when the span is contained in the quotient image. See theorem 1.7.

`reconstruct_cancellation` line 82

A quotient witness supplies exact cancellation in the original target. See theorem 1.5.

B.2 Minimal

Source: `lean/KomorebiRepair/Minimal.lean`.

`repair_dimension_lower_bound` line 14

Every finite dimensional added space has dimension at least that of the obstruction span. See theorem 2.1.

`exists_obstruction_section` line 22

The vector space quotient admits a classical linear section. See theorem 2.2.

`exists_attaining_repair` line 30

Restricting a section yields a repair whose domain is the required span. See theorem 2.3.

`free_minimal_repair_rank` line 46

Attainment and the lower bound establish the least possible added dimension. See theorem 2.4.

B.3 ConstrainedMinimal

Source: `lean/KomorebiRepair/ConstrainedMinimal.lean`.

`admissible_subspace_attainment` line 11

A feasible legal subspace contains an attaining lift of the required span. See theorem 2.5.

`admissible_subspace_no_go` line 30

An excluded obstruction span rules out every added map valued in the legal subspace. See theorem 2.6.

B.4 Menu

Source: `lean/KomorebiRepair/Menu.lean`.

`menuMap_apply` line 18

The menu map is the actual finite sum of selected coefficient directions. See equation (3.1).

`menu_quotient_range` line 24

The quotient image of the menu is the span of the selected quotient directions. See theorem 3.1.

`finite_menu_repair_iff_span` line 40

Menu feasibility is equivalent to inclusion of the residual obstruction span. See theorem 3.2.

`menu_mono` line 47

Adding selected directions enlarges their quotient span. See theorem 3.3.

`repairable_menu_mono` line 53

Feasibility is preserved when a menu is enlarged. See theorem 3.3.

B.5 Conservation

Source: `lean/KomorebiRepair/Conservation.lean`.

`invariant_preserved` line 10

Original and added corrections leave the stated linear invariant unchanged. See theorem 3.5.

`conservation_no_go` line 17

A nonzero conserved residual invariant excludes a repair. See theorem 3.6.

`menu_preserves_invariant` line 28

An invariant that kills every selected direction kills the whole menu map. See theorem 3.7.

`conservation_menu_no_go` line 36

The invariant obstruction applies to the actual selected coefficient map. See theorem 3.7.

B.6 Transport

Source: `lean/KomorebiRepair/Transport.lean`.

`correction_range_maps` line 12

A commuting correction square transports the original image into the target image. See theorem 4.1.

`transport_obstruction_naturality` line 23

The induced quotient map carries the actual residual class to its transported class. See theorem 4.2.

`faithfulOn_iff_injective` line 32

Faithfulness on a submodule is injectivity of the restricted quotient map. See theorem 4.4.

`transport_faithfulness_on_obstruction_span` line 49

Faithfulness on the required span makes target correctability reflect source correctability. See theorem 4.5.

`nonfaithful_witness` line 63

Failure of faithfulness produces a specific nonzero class that disappears. See theorem 4.6.

B.7 Search

Source: `lean/KomorebiRepair/Search.lean`.

`bestByCheck_spec` line 20

The filtered finite selector returns a passing candidate of minimum cost. See theorem 5.3.

`bestByCheck_none_iff` line 30

The selector rejects exactly when no listed candidate passes its Boolean predicate. See theorem 5.3.

`repairMenuCheck_spec` line 45

The executable Boolean is equivalent to the original family repair equations. See theorem 5.1.

`mem_allMenus` line 54

Every subset of the finite candidate universe occurs in the enumerated list. See theorem 5.2.

`menu_optimality_certificate` line 70

A selected menu is allowed, feasible, and minimum cost in the declared universe. See theorem 5.4.

`menu_search_none_iff` line 86

Rejection is complete within the finite candidate and admissibility contract. See theorem 5.5.

B.8 Arithmetic

Source: `lean/KomorebiRepair/Arithmetic.lean`.

`Arithmetic.linear_digit_exists` line 13

Bezout data construct an integer solution of the new linear digit congruence. See theorem 6.1.

`Arithmetic.quadratic_digit_correction` line 20

The exact quadratic expansion proves one higher divisibility scale. See theorem 6.2.

Arithmetic.quadratic_hensel_step line 33

A coprime derivative supplies a next quadratic correction at a prime power scale. See theorem 6.3.

Arithmetic.hensel_seven_trace line 43

The seven recorded representatives satisfy their seven exact integer congruences. See section 6.3.

Arithmetic.singular_mod_four_impossible line 52

There is no square root of two in the ring of integers modulo four. See theorem 6.4.

Arithmetic.singular_derivative_can_still_lift line 54

The exact root one of the equation $x^2 = 1$ persists at every dyadic depth. See theorem 6.5.

B.9 Nonlinear

Source: `lean/KomorebiRepair/Nonlinear.lean`.

Nonlinear.nonlinear_root_despite_first_order_obstruction line 10

The derivative obstruction at zero does not exclude the displayed nonlinear root. See theorem 7.1.

Nonlinear.linearized_solution_without_real_root line 24

An actual solvable derivative equation does not imply a global real root. See theorem 7.2.

B.10 Examples

Source: `lean/KomorebiRepair/Examples.lean`.

Examples.first_menu_entry_insufficient line 24

The first rational menu direction alone cannot cancel the residual. See theorem 3.4.

Examples.second_menu_entry_insufficient line 33

The second rational menu direction alone cannot cancel the residual. See theorem 3.4.

Examples.two_menu_entries_repair line 39

Two selected rational directions supply the displayed exact correction. See equation (3.3).

Examples.one_free_direction_repairs line 44

One freely chosen rational direction repairs the same residual. See theorem 3.4.

Examples.coarse_information_loss line 50

Projection sends an uncorrectable rational residual to a correctable target residual. See theorem 4.7.

Examples.collapsing_transport_not_faithful line 59

The associated quotient transport fails the required span faithfulness condition. See theorem 4.7.

Examples.executable_menu_returns_both line 81

Kernel reduction proves the optimizer returns both finite field directions. See section 5.3.

Examples.executable_restricted_menu_rejects line 88

Kernel reduction proves rejection under the one entry admissibility rule. See section 5.3.

Examples.executable_menu_is_globally_optimal line 96

The generic optimum theorem proves the two entry lower bound for every feasible subset. See section 5.3.

APPENDIX C

Verification and provenance

C.1 The evidence attached to the formal release

The v0.2.0 verification record reports successful compilation of ten proof modules, an axiom audit of all 49 named theorem declarations, and individual replay of those modules through Lean's kernel. The allowed transitive axiom set is

$$\{\texttt{propext}, \texttt{Classical.choice}, \texttt{Quot.sound}\}.$$

No additional assumption axiom, placeholder axiom, or native decision axiom is allowed in the accepted theorem library.

The environment recorded by the release is Lean `4.34.0-rc2`. Its commit is

`6a10ac8c22beadecabdbb0919c2b50214762f91d`

with Mathlib revision

`aeb72f1e9981b6d35b426b16dca7759be3e7ba06`.

All dependency pins remain in the original `lake-manifest.json`.

The recorded `leanchecker` replay uses Lean's own kernel against its imported environment. It is not an independently implemented checker and is not a source rebuild of every imported Mathlib theorem. The small source archive does not bundle the compiler or its full dependency cache.

Recorded formal release check	Coverage
Proof module compilations	10
Named theorem axiom audits	49
Individual same kernel module replays	10
Intended compiler rejection controls	5
Separate forbidden-axiom policy control	1
Python unit tests in the recorded replay	20
Individual cases in the recorded Python audit	3,307
Explicit assertions in that audit	71

The original archive's final extraction check also records successful rebuilding and replay after extraction. These are historical execution records carried with the unchanged source. Those historical receipts remain unchanged. The expanded v0.3.0 execution has its own fresh compilation and replay record, described in the additional verification appendix.

C.2 Negative controls and their meaning

The five compiler controls intentionally assert a false equality, omit required faithfulness, claim an insufficient menu succeeds, claim a false divisibility gain, or introduce a proof placeholder. The verifier requires the intended diagnostics, rather than accepting an unrelated parsing or missing instance failure as a successful negative test.

A sixth control deliberately adds a false axiom. Lean can correctly verify a theorem conditional on that axiom. The separate release policy then rejects the dependency. This distinguishes logical checking under assumptions from acceptance of those assumptions as part of a mathematical release.

The development evidence retains failed proof drafts and superseded verification runs. In particular, the release ledger records corrections to an axiom-name expectation and to a negative control that initially failed for a missing decidability instance. Only the hardened final record governs the stated formal status.

C.3 Experimental evidence is a different layer

The earlier exact audit [5] includes rational and finite-field linear calculations, Euler inverse identities, selected cyclotomic coefficients, finite transport graphs, and arithmetic and nonlinear controls. Its 3,307 case records are retained in the formal handoff. These computations do not prove the unrestricted versions of the surrounding arithmetic or representation-theoretic problems.

The finite graph and Heisenberg matrix calculations are not a formal construction of a local-field representation. The Euler and cyclotomic tests are not a new proof of the full Euler conjecture. The original ten Lean modules do not contain an integrated replay of Papers III or XI. No result here resolves Paper XI's Conjecture 66 or provides a universal arithmetic repair theorem.

C.4 The monograph edition

The source of this expanded edition is a standalone LaTeX file. It contains the bibliography and all commutative diagrams, with no external image or font files required. The original formal release is included unchanged alongside the document.

The unchanged v0.2.0 formal source archive has SHA-256

`d906e98ddc2a661f3201ba51e1f1049a49dfa9326078384dfa851b608b80f337`.

Its checksum agrees with the recorded extraction report. The twelve positive source and audit-harness files listed in the verification record match their stored SHA-256 values. The declaration registry matches the theorem names in those source files and their recorded axiom entries.

The publication evidence separates these integrity checks from LaTeX compilation and layout inspection. It records the resulting page count and document hashes. The ten original proof modules were not changed. New matrix modules and fresh proof evidence are recorded separately. No mathematical completion claim is inferred merely from producing a PDF.

APPENDIX D

Rational bridge declaration register

The twenty general declarations below supplement the unchanged original register. All source paths are relative to the release root. The full machine-readable register in `docs/theorem_registry.json` includes every concrete case and every named rejection theorem as well.

D.1 RationalCertificates

Source: `lean/KomorebiRepair/RationalCertificates.lean`.

`RationalCertificates.checkRepair_sound` line 45

An accepted rational coefficient table repairs the fixed original residual family.

`RationalCertificates.annihilator_kills_residual` line 51

An annihilator of both correction ranges annihilates every successfully repaired residual.

`RationalCertificates.separator_sound` line 64

A nonzero detected residual rules out a correction by that fixed menu.

`RationalCertificates.checkSeparator_sound` line 69

The Boolean separating-row checker is sound.

`RationalCertificates.checkOptimal_sound` line 79

A successful menu with separators for all cheaper admissible menus is globally minimum cost within the declared menu.

`RationalCertificates.checkNoRepair_sound` line 96

Separators for every admissible subset certify rejection in that finite universe.

`RationalCertificates.dimension_certificate_right_inverse` line 104

Every competing repair supplies an explicit right inverse of the detected added map.

`RationalCertificates.width_le_of_right_inverse` line 121

A rectangular rational map with a right inverse has at least as many source as target coordinates.

`RationalCertificates.dimension_certificate_lower_bound` line 129

The matrix identities imply the lower bound against every competing rational augmentation.

`RationalCertificates.checkDimension_sound` line 142

A successful repair and a lower certificate certify attained minimum dimension.

`RationalCertificates.checkRank_sound` line 157

A factorization and an identity minor certificate prove the declared exact rank.

D.2 ConstructiveMatrix

Source: `lean/KomorebiRepair/ConstructiveMatrix.lean`.

`RationalCertificates.split_constructs_repair` line 13

The split and residual factorization explicitly construct the correction coefficients.

`RationalCertificates.split_correctable_iff` line 30

The complete detector vanishes exactly when the original rational matrix equation is solvable.

`RationalCertificates.checkOptimalConstruction_sound` line 58

Four matrix identities construct an attaining repair and prove its minimum added dimension.

`RationalCertificates.constructed_repair_in_legal_image` line 77

An additional image identity certifies that the constructed directions are legal.

`RationalCertificates.decodeCell_zero` line 89

A typed rational cell with denominator zero is rejected.

`RationalCertificates.decodeCell_nonzero` line 92

A cell with nonzero denominator decodes to the stated rational value.

`RationalCertificates.decodeCell_sound` line 96

Successful cell decoding entails a nonzero denominator and the correct value.

D.3 Wire

Source: `lean/KomorebiRepair/Wire.lean`.

`RationalCertificates.checkWireMatrix_sound` line 20

The finite matrix validity check certifies all denominators.

`RationalCertificates.decodeWireMatrix_cell` line 23

Matrix decoding agrees pointwise with successful cell decoding.

D.4 Concrete exported problems

Each row below denotes one original problem and its separately supplied certificate. Every case has six named proofs: `input_encoding_valid`, `certificate_encoding_valid`, `certificate_checked`, `original_equations_checked`, `output_matches_construction`, and `optimal`. The last theorem quantifies over every competing rational augmentation, not only the candidates tested by Python.

Case	m	n	s	d	t	Example class
Case000	3	1	2	2	1	worked
Case001	2	1	1	1	1	fractional
Case002	0	2	3	0	0	empty target
Case003	3	0	2	3	2	empty original

Case	m	n	s	d	t	Example class
Case004	3	1	0	2	0	empty residuals
Case005	3	1	2	2	0	zero residual
Case006	3	3	2	0	0	surjective
Case007	3	2	2	3	2	zero original matrix
Case008	0	0	0	0	0	all empty
Case009	4	2	2	3	2	nontrivial rank two
Case010	3	1	2	3	2	random 000
Case011	3	3	1	0	0	random 005
Case012	3	2	2	2	2	random 006
Case013	3	4	3	3	3	random 007
Case014	2	2	0	0	0	random 008
Case015	2	1	2	1	1	random 010
Case016	3	2	0	3	0	random 013
Case017	0	3	4	0	0	random 014
Case018	2	3	4	0	0	random 015
Case019	1	4	1	1	0	random 016
Case020	3	1	3	2	2	random 017
Case021	4	4	0	0	0	random 018
Case022	0	1	1	0	0	random 019
Case023	1	4	4	1	1	random 020

The input and certificate sources are grouped in `Generated/Inputs00.lean` through `Inputs03.lean`, and `Generated/Certificates00.lean` through `Certificates03.lean`. Groups contain six cases each. The JSON index gives the exact original input hash for each row.

D.5 Menu and corruption examples

The eight `RationalMenus` declarations check the two-entry optimum, restricted rejection, invalid data, and zero- and one-rank certificates. Six further named proofs in `RejectedCertificates.lean` establish that deliberately corrupted checks return false. Their companion invalid files attempt to use these false equalities as true equalities and are required to fail for the intended type mismatch.

APPENDIX E

Rational bridge verification and use

E.1 The v0.3.0 checkpoint

The following records describe the preceding release. Its source modules are retained unchanged and freshly checked again in the v0.4.0 run. The v0.4.0 menu extension is documented in the next appendix. A later appendix records the v0.5.0 tree extension, followed by the present release contract.

The runtime was recovered from the previously supplied archive parts and its joined checksum was matched before extraction. The compiler reports Lean `4.34.0-rc2`; the actual Mathlib checkout has revision `aeb72f1e9981b6d35b426b16dca7759be3e7ba06`. The combined project was rebuilt from its own source files after removing its previous object files. The dependency cache was retained.

Fresh check	Result
Proof modules compiled and individually replayed	23 / 23
Named theorem dependencies audited	227 / 227
Unchanged original declarations	49
New general certificate and decoding declarations	20
New concrete declarations	158
Exported original problems with optimality proofs	24 / 24
Paired corruption rejection controls	6 / 6
Separate forbidden-axiom policy control	1 / 1
Lean input echoes compared with original JSON	24 / 24
New Python unit tests	34 / 34
Exact matrix cases with independent SymPy ranks	264 / 264
Inherited Python unit tests rerun	20 / 20
Inherited exact audit case records reproduced	3,307

The 158 new concrete declarations consist of 144 input/certificate proofs for the 24 original problems, eight menu/rank examples, and six rejection examples. These are coverage counts, not counts of independent discoveries. The general theorems and the concrete proofs have separately labeled entries in the source registry.

Every audited dependency set is contained in {`propext`, `Classical.choice`, `Quot.sound`}. No added assumption, placeholder or native-decision axiom is accepted. The module replay invokes the same Lean kernel, not an independently implemented proof checker. It does not re-prove all imported Mathlib declarations.

E.2 The single-command example

The release includes `scripts/certify_one.py`. Given an original JSON input and the pinned offline kit, it constructs a certificate in a new local directory, emits the input separately from the witness, compiles the generic modules and the concrete proof, audits all relevant axioms, replays the modules, and compares a Lean-decoded echo with the original numeric data.

```
export KOMOREBI_LEAN_KIT=/path/to/Komorebi_Portable_Lean_4.34.0-rc2

python scripts/certify_one.py examples/corpus/Case001_input.json --output /tmp/my-repair
```

This exact workflow was executed on the fractional example. It ended with `KERNEL_CHECKED_OPTIMAL_REPAIR` and $t = 1$. It requires no network access or publication service. An existing output directory is refused so that previous evidence is not silently overwritten. The large compiler and dependency kit are not duplicated in the small source handoff.

E.3 Failures and their corrections

The first compilation attempt with a broad tactic import was interrupted. The implemented modules use the required direct imports. Plain `decide` did not reduce the first rational matrix fixture, so the producer now emits `decide +kernel`. The typed validity predicate initially lacked a reducible decidability path and was corrected. An ambiguous corruption diagnostic was replaced by paired rejection proofs. In the final control, the false checker result is itself proved before an invalid true result is rejected.

The development snapshots and logs remain in `evidence/development`. Interrupted and superseded runs are not counted as successful checks. The inherited v0.3.0 record is preserved under `evidence/verification/v0.3.0_inherited`. The v0.4.0 receipt is preserved under `evidence/verification/v0.4.0_inherited`. That historical combined receipt is now preserved under `evidence/verification/v0.5.0_inherited`. The current receipt governs the release candidate described in the final appendix.

E.4 Remaining boundaries

The new code is a standalone extension. It has not been integrated into the authoritative Komorebi Python library or Paper III source tree. At that checkpoint, the rational-menu exporter was still pending. Chapter 10 now supplies that path. Full formalization of the external JSON parser and a formal connection of the matrix rank-difference formula to the earlier quotient API remain separate tasks. The current free-augmentation exporter, certificate soundness, attaining construction, and finite literal proofs are complete within their stated rational matrix scope.

The construction does not establish unrestricted nonlinear repair, integral or arithmetic realization, or a universal semantic theorem for Komorebi Space. It does not alter Conjecture 66. Its purpose is

narrower and operational: allow an untrusted numerical search to submit exact witnesses that are checked against the original problem.

APPENDIX F

Menu certificates: sources and fresh verification

This appendix preserves the v0.4.0 continuation record, rather than describing the full v0.5.0 count. The 23 proof modules from v0.3.0 remain byte-for-byte unchanged. Fourteen additional modules supply a seven-theorem cover interface, 24 exported rational-menu problems, and eight paired corruption controls. The full combined project is compiled and replayed again, rather than accepting the prior receipts as a substitute for the new run.

F.1 General declarations

The new general source is `lean/KomorebiRepair/RationalMenuCover.lean`. The following records give its exact current line locations. Definitions, decidability instances, and diagnostic JSON display functions are not counted as theorems.

`KomorebiRepair.RationalCertificates.mul_masked_entry` line 16

A column-mask identity for exact matrix multiplication.

`KomorebiRepair.RationalCertificates.separates_subset` line 22

A valid separator for a larger menu separates every subset.

`KomorebiRepair.RationalCertificates.covered_infeasible` line 48

A covered selection is infeasible when every listed block is valid.

`KomorebiRepair.RationalCertificates.checkCoveredOptimal_sound` line 59

Acceptance proves admissibility, the original equations, and minimum cost among admissible feasible selections.

`KomorebiRepair.RationalCertificates.checkCoveredNoRepair_sound` line 77

Acceptance excludes every selection admitted by the original rule.

`KomorebiRepair.RationalCertificates.tableAllowed_iff` line 94

The Boolean admissibility table is equivalent to membership in its fixed input list.

`KomorebiRepair.RationalCertificates.additiveCost_empty` line 97

The empty selection has zero additive cost.

F.2 Exported original problems

Every input is emitted independently of its coefficient and separation witnesses. The six input modules and six certificate modules under `MenuGenerated/` hold four cases each. The full source registry, JSON files, and hashes are included in the package.

Case	Main test	Outcome	Cost	Blocks
Menu000	two entry optimum	optimum	2	2
Menu001	size one restriction	none	–	2
Menu002	pair cheaper than singleton	optimum	2	2
Menu003	fractional witness	optimum	7	1
Menu004	full infeasible cover	none	–	1
Menu005	free redundant directions	optimum	3	1
Menu006	zero cost tie	optimum	0	0
Menu007	nonhereditary allowed	optimum	2	0
Menu008	empty admissible universe	none	–	0
Menu009	empty residual family	optimum	0	0
Menu010	zero target dimension	optimum	0	0
Menu011	zero original domain	optimum	2	2
Menu012	zero candidates correctable	optimum	0	0
Menu013	zero candidates blocked	none	–	1
Menu014	multiple residuals	optimum	6	2
Menu015	inadmissible free solution	optimum	1	0
Menu016	seeded 000	optimum	3	0
Menu017	seeded 001	optimum	0	0
Menu018	seeded 002	optimum	1	1
Menu019	seeded 003	optimum	0	0
Menu020	seeded 004	optimum	0	0
Menu021	seeded 005	optimum	0	0
Menu022	seeded 006	optimum	4	2
Menu023	seeded 007	optimum	1	0

There are twenty positive optimal-menu results and four no-admissible-menu results. One of the latter has an empty admissibility table. Its rejection is explicitly vacuous, not a claim that a nonzero cokernel class has been detected.

F.3 Fresh verification results

Check	Result
Combined proof modules compiled and replayed	37 / 37
Named theorem dependencies audited	378 / 378
Unchanged v0.3.0 declarations	227
Additional general declarations	7
Additional concrete declarations	144
New original menu problems exported to Lean	24 / 24
New matrix, admissibility, and cost echoes	24 / 24
New paired corruption controls	8 / 8
Previous paired corruption controls rerun	6 / 6
Separate forbidden-axiom policy control	1 / 1
New Python unit tests	50 / 50
Exact menu problems independently compared	120 / 120
Individual subset rank comparisons	709 / 709
Previous rational Python unit tests rerun	34 / 34
Original Python unit tests rerun	20 / 20
Original exact audit records reproduced	3,307

The 144 additional concrete declarations comprise 136 input and certificate statements for the 24 problems and eight positive proofs that corrupted checks return false. This is a coverage count, not a count of new mathematical discoveries. The underlying finite linear alternative, monotonicity of linear span, and finite optimization arguments are classical.

All transitive axiom lists are contained in {`propext`, `Classical.choice`, `Quot.sound`}. Concrete certificates use `decide +kernel`. Replay invokes Lean's own kernel against the pinned imports. It is not an independently implemented checker and does not rebuild or re-prove all Mathlib dependencies.

The independent Python comparison uses SymPy matrix ranks to assess every subset of each recorded problem. It checks the producer's selected minimum or infeasibility outcome, rather than merely comparing two calls to the same elimination routine. The seeded run contains 120 problems and 709 individual subset comparisons. All rational arithmetic is exact.

F.4 Rejection controls and command execution

The new corruption controls remove cheaper-menu coverage, replace a separator by zero, overstate a block's scope, change correction coefficients, propose an expensive singleton despite a cheaper repair, select a forbidden menu, claim rejection of a feasible family, and attempt to use an unselected column. Each corrupted Boolean check is first proved false. A separate invalid source then tries to use that result as true and must fail with the intended type mismatch. An unrelated missing instance or syntax failure is not accepted as a rejection result.

The single-command workflow was executed on the fractional problem and on the four-candidate globally infeasible menu. The former returns `KERNEL_CHECKED_OPTIMAL_MENU`, cost seven, with

coefficient $-5/4$. The latter returns `KERNEL_CHECKED_NO_ADMISSIBLE_MENU`, using one separator for sixteen subsets. Both commands compile the sources, audit all relevant theorem dependencies, replay the modules, and match the Lean input echo to the original matrices and constraints.

A separate trial-budget control deliberately permits zero trials. It returns `UNRESOLVED` before claiming a mathematical outcome. The log is retained. This distinction is part of the operational contract, not merely a closing qualification.

F.5 Development record and remaining scope

The first new cover-module compilation failed because `if_true` and `if_false` are deprecated in the pinned runtime and warnings are treated as errors. They were replaced by `ite_true` and `ite_false`; the subsequent source compiled. That failed attempt and its exact source snapshot remain in the development evidence.

The existing free-repair command was also adjusted to select only the general declarations it actually imports, rather than every declaration later assigned the general-layer label. Its mathematical modules are unchanged. This prevents an expanded registry from causing the older command to request nonexistent imports.

The external JSON parser, source emission, hashing, and file system are tested but not formally verified. The checked mathematical statements apply to the literal typed input compiled by Lean. Separate input emission, hashes, and a Lean-decoded echo provide executed evidence that those literals match the intended problem. This boundary remains explicit.

The v0.4.0 producer was exhaustive and bounded. Its implementation remains unchanged as a reference; the following continuation supplies a separate branch-and-bound path. It does not minimize evidence size. No integration into the authoritative Komorebi Python package or Paper III source tree is claimed. No theorem here constructs native arithmetic hosts, proves nonlinear convergence, or establishes a universal realization theorem for Komorebi Space. Those tasks remain separate from the now-operational costed rational-menu certificate path.

APPENDIX G

Branch certificate verification

This appendix records the v0.5.0 continuation. The full 37-module v0.4.0 proof tree is retained byte-for-byte. Fourteen additional modules supply the general branch checker, 24 exported problems, and paired corruption controls. Earlier appendices remain historical records of their respective releases.

G.1 General declaration correspondence

Source: `lean/KomorebiRepair/BranchCertificate.lean`. Each declaration is proved with explicit matrix, cost and admissibility data. No field assumes that an optimal repair exists.

`KomorebiRepair.RationalCertificates.intervalExcluded_sound` line 42

A checked exclusion of an interval from the original optional admissibility table rules out every admitted selection in that interval.

`KomorebiRepair.RationalCertificates.additiveCost_mono` line 53

Nonnegative additive column costs are monotone under inclusion of the selected set.

`KomorebiRepair.RationalCertificates.boundCheck_mono` line 68

A checked lower cost bound propagates to a larger cost. No threshold means impossibility, so a cost leaf cannot apply.

`KomorebiRepair.RationalCertificates.checkBranch_sound` line 95

Structural induction on an accepted tree gives a lower cost bound or infeasibility for every admitted feasible selection in its interval.

`KomorebiRepair.RationalCertificates.checkTreeOptimal_sound` line 131

A successful original coefficient equation and an accepted root tree certify minimum cost within the declared candidate universe.

`KomorebiRepair.RationalCertificates.checkTreeNoRepair_sound` line 148

An accepted root tree in impossibility mode excludes every selection admitted by the original rule.

G.2 Exported case register

The following cases are emitted from independent original JSON inputs and exact witness records. The tree-node count is the submitted certificate size, not a count of separately evaluated subsets. Cost is omitted for a rejection result.

Case	Description	Candidates	Cost	Nodes
Branch000	two entry optimum	2	2	5
Branch001	size one restriction	2	–	5
Branch002	cheaper pair	3	2	13
Branch003	fractional	1	7	3
Branch004	zero cost	2	0	5
Branch005	nonhereditary only pair	2	2	5
Branch006	empty admissible table	2	–	1
Branch007	empty residuals	2	0	1
Branch008	zero target	2	0	1
Branch009	zero domain	2	2	5
Branch010	no candidates feasible	0	0	1
Branch011	no candidates impossible	0	–	1
Branch012	multiple residuals	2	6	5
Branch013	free solution forbidden	2	1	5
Branch014	duplicate costs	3	3	7
Branch015	incumbent improvement	3	4	9
Branch016	non downward rules	3	5	11
Branch017	four menu impossible	4	–	1
Branch018	large 24 optimum	24	5	5
Branch019	large 24 impossible	24	–	1
Branch020	large 24 zero cost	24	0	3
Branch021	large 24 finite admissibility	24	5	5
Branch022	large 24 empty table	24	–	1
Branch023	nonempty rules empty residuals	2	0	3

The register contains 18 optimal-menu certificates and 6 no-admissible-menu certificates. An empty admissibility table is explicitly vacuous. It is not evidence that a nonzero arithmetic obstruction was discovered.

G.3 Fresh combined verification

The current evidence record is `evidence/verification/current/verification.json`. The prior current record is preserved under `evidence/verification/v0.4.0_inherited`. The full combined library is recompiled from source and each module is replayed through Lean's kernel against the pinned imported environment.

Check	Result
Combined proof modules compiled and replayed	51 / 51
Named theorem dependencies audited	526 / 526
Unchanged v0.4.0 proof modules	37 / 37
Additional general declarations	6
Additional concrete declarations	142
New exported original-input echoes	24 / 24
New paired corruption controls	10 / 10
Previous paired corruption controls rerun	14 / 14
Separate forbidden-axiom policy control	1 / 1
New Python unit tests	50 / 50
New exact branch problems rechecked	204 / 204
Small problems compared with independent exact ranks	199 / 199
Individual subset rank comparisons	3,173 / 3,173
Comparisons with the previous exhaustive solver	199 / 199
Previous rational and menu unit tests rerun	84 / 84
Original Python unit tests rerun	20 / 20
Original audit case records reproduced	3,307

The additional 142 concrete declarations are input-encoding, certificate, equation, cost and rejection proofs. They are not a count of independent mathematical discoveries. The six general results specialize classical finite optimization and linear obstruction arguments to the declared proof-tree interface.

All audited theorem dependencies are contained in {`propext`, `Classical.choice`, `Quot.sound`}. Concrete reductions use `decide +kernel`. No new assumption axiom, `sorryAx`, or native-decision axiom is admitted. Replay uses Lean's own kernel, not a second independently implemented checker and not a fresh proof of all Mathlib dependencies.

The external comparison enumerates subsets only in the small benchmark oracle, never in the new production checker. SymPy supplies independent exact ranks for 3,173 subsets of 199 small problems. The earlier exhaustive producer agrees on the outcome and minimum cost. Ties need not return the same selected labels. Five larger structured problems are not exhaustively tested by that oracle; their exported trees receive the formal check instead.

G.4 Controls, command execution, and provenance

The new negative controls forge a root cost bound, falsely exclude an all-admitted interval, use a zero separator, repeat a resolved split, duplicate a cost child, swap branch meanings, alter correction coefficients, use an unselected column, choose a forbidden selection, and claim no repair despite a valid incumbent. Each corrupted checker result is proved false. A separate invalid source must then fail specifically when treating that equality as true. Missing instances and syntax errors do not count as successful mathematical rejection.

The large optimal and infeasible cases were executed through the single-command interface. The command independently emits the original matrices, weights and optional admissibility table, then

the witnesses. Compilation, axiom auditing, same-kernel replay, and an exact Lean input echo follow. Unlike the old menu echo, the new echo does not enumerate every derived selection: it returns the exact original policy data.

A deliberately zero node budget produces `UNRESOLVED`, not a rejection of the mathematical problem. Rejected external evidence has the same unresolved status. A no-repair status requires an accepted certificate in the mode that forbids cost leaves.

The first branch-module proof used an incorrect Boolean conjunction lemma name. The pinned compiler rejected it. The corrected proof uses `Bool.and_eq_true_iff`; both source and log versions are preserved.

A later audit review found that inherited compilation labels used only file stems. Generated modules in distinct namespaces could therefore overwrite one another's diagnostic log and check key. That run is preserved as superseded. The verifier now uses full relative module names, rejects duplicate command labels, and repeats the full combined verification. No theorem statement, proof assumption, or certificate was weakened by this correction.

A separate evaluator inspection caught a more consequential implementation issue. The first version wrote a bounded-looking proposition over the admissibility list and asked Lean to infer its decision procedure. Lean instead selected a finite-type universal quantifier over candidate subsets. The theorem remained sound, but an exclusion leaf could still perform a powerset scan. The implementation now calls `List.all` explicitly and proves its connection to the mathematical exclusion predicate. A 24-candidate empty-table regression and a nonhereditary 24-candidate table regression exercise this path. The final verifier also inspects the elaborated definition and rejects the previous hidden powerset quantifier. Both evaluator outputs and the superseded source are retained.

G.5 Trust and remaining scope

Typed rational decoding and the mathematical certificate theorems are formalized. The complete JSON parser, untrusted search, source emitter, hashing and filesystem are tested, not formally verified. The kernel certifies the literal typed input in the generated source. Separate source emission, hashes, and a complete original-data echo test its binding to the supplied JSON.

The producer accepts up to 24 candidates and has finite resource guardrails. Search and certificate size can remain exponential. The greedy order is not proved optimal and proof-tree size is not minimized. The certificate soundness theorem does not require the producer to exhaust its entire budget or follow a particular search order.

Direct integration into the authoritative Komorebi Python library and Paper III source tree, full parser verification, and a formal quotient-to-matrix rank-difference bridge remain separate tasks. There is no new native arithmetic host construction, nonlinear convergence theorem, change to Conjecture 66, or unrestricted Komorebi Space realization claim.

APPENDIX H

Release contract and reproducibility

H.1 A frozen mathematical component

Release candidate 1.0.0rc3 is the standalone exact rational repair component described in this monograph. It does not assign a final version to all of Operation Komorebi. The 51 proof modules and 526 named theorem declarations are unchanged from v0.5.0. No new mathematical assertion is introduced by the release number.

The component constructs freely chosen augmentations, chooses minimum cost repairs from prescribed menus, and verifies branch certificates accounting for skipped alternatives. Every acceptance concerns the original matrices, coefficient ring, costs, and admissibility rules. No claim about integral coefficients, nonlinear convergence, native arithmetic realization, or Conjecture 66 follows merely from a successful rational matrix check.

The previous candidates' API corrections, encoding regression, and runtime pin checks are retained. Their detailed histories remain in the source archive's documents and logs. This appendix consolidates the current release contract rather than accumulating another chronological release report inside the mathematical exposition.

H.2 Using the installed interface

The Python wheel has no third party runtime dependencies. Its public interface accepts exact matrices and returns finite certificate data. For example:

```
from komorebi_repair import ExactMatrix, menu_problem, solve, verify

D = ExactMatrix.of([[1], [0], [0]])
E = ExactMatrix.of([[0, 0], [1, 0], [1, 1]])
R = ExactMatrix.of([[0], [1], [0]])
p = menu_problem(D, E, R, costs=[1, 1])
answer = solve(p)
assert answer.status == "PYTHON_CERTIFICATE_VALID"
assert answer.certificate["cost"] == 2
assert verify(p, answer.certificate).status == "PYTHON_CERTIFICATE_VALID"
assert answer.kernel_checked is False
```

A rational entry is an integer or an exact `fractions.Fraction` value. Floating point entries are rejected rather than approximated. Matrix dimensions and index types are checked. The wire format represents each rational as a reduced numerator and positive denominator. Malformed input,

invalid evidence, and exhausted resource budgets remain unresolved outcomes, not impossibility results.

The installed command can produce and recheck certificates. Kernel acceptance is performed by the separate source workflow with the pinned Lean kit:

```
python scripts/certify_branch.py \
  examples/branch_corpus/Branch018_input.json \
  --kit /path/to/Komorebi_Portable_Lean_4.34.0-rc2 \
  --output /tmp/new-kernel-evidence
```

Every output directory must be new. The workflow retains the original input, the submitted or constructed certificate, the emitted Lean source, axiom lists, replay logs, and the original-input echo. The producer remains outside the formally proved portion of the system. A failed producer does not invalidate the certificate theorem, and a successful producer does not replace kernel checking.

H.3 A gate for the exact distributed artifact

A release receipt can be genuine yet stale. It may describe yesterday's source while a current wheel contains different code. A correct PDF may no longer correspond to the distributed LaTeX. A proof file may have changed after the recorded compilation. File integrity alone cannot resolve these questions unless the relevant artifacts are bound to their evidence.

The current release gate checks the strict manifest and then makes those bindings explicit. It compares every current proof source with the formal receipt, requires the complete axiom inventory, and checks that every module has a compilation and replay record. It compares the wheel's actual Python members with the source checkout and checks that installed acceptance used that same wheel. The acceptance receipt binds the Python modules, scripts, tests, README, and build metadata. A separate document receipt binds the LaTeX source and compiled PDF.

```
python scripts/release_gate.py
```

The command is intended for a freshly extracted, unmodified release. It suppresses local bytecode creation so that checking does not itself change the sealed file inventory. Further experiments should use separate working copies or output directories.

> **What a passing gate means**
>
> The gate checks consistency of the supplied source, artifacts, and receipts. It does not authenticate their author. Unsigned hashes are not signatures. An attacker able to replace the code and all receipts can fabricate mutually consistent files. The included fresh verification workflows permit the reader to perform the checks again with a separately trusted runtime.

The gate also keeps technical readiness separate from permission to publish. The current publication decision records unselected code and document licenses. Technical acceptance is therefore reported with a pending author decision, not as authority to make a public release.

H.4 Compact provenance without recursive archives

The predecessor distribution repeatedly embedded earlier release ZIPs, which themselves embedded earlier ZIPs. The current package removes that recursive duplication without removing the active program, proof sources, examples, mathematical chapters, declaration registers, or ordinary evidence files.

Historical proof identity is retained through 121 exact source snapshots: ten from v0.2.0, 23 from v0.3.0, 37 from v0.4.0, and 51 from v0.5.0. The lineage index records their original archive hashes, filenames, byte counts, and source hashes. Each snapshot was extracted from the identified archive and compared with the current source. The full verifier repeats those source comparisons before rebuilding the current modules.

The whole predecessor ZIPs remain separate historical artifacts. They are not required to run the current verifier. Earlier reports describing bundled archives describe those earlier editions. They are retained as historical records rather than being silently rewritten into claims about this distribution.

The compiler and Mathlib cache also remain separate. The distribution contains the mathematical and software sources needed for its own workflows, not a second copy of the large pinned runtime.

H.5 Acceptance and supported environment

The formal verification rebuilds and replays all 51 modules and audits all 526 named declarations. The permitted axiom set remains `propext`, `Classical.choice`, and `Quot.sound`. The 24 paired rejection controls and the separate forbidden-axiom policy control are retained. The full run comprises 141 commands and 230 passing checks.

Current acceptance component	Count
Public API, portability, and artifact-binding unit tests	100
Retained rational and menu unit tests	134
Original prototype unit tests	20
Original exact audit records replayed	3,307
Original input/certificate pairs exercised from the wheel	72
Installed-package assertions	369
Installed-platform workflow checks	24

The artifact tests include changed sources, stale wheels, absent proof coverage, forbidden axioms, duplicated records, and missing replay entries. A separate disposable-archive test refreshes the ordinary manifest after intentional corruption. The evidence bindings must still reject the altered release. These are tests of the release gate, not additional mathematical impossibility theorems.

The release-tested configuration is Linux x86_64 with CPython 3.13.5. The C-locale regression runs with Python's UTF-8 mode disabled. File and subprocess text encodings are explicit. Paths containing spaces and Unicode are tested on that same Linux host. None of these checks constitutes a Windows simulation.

Native Windows, WSL, macOS, ARM, PyPy, and other Python interpreters remain untested. The earlier attempt to acquire additional Python versions failed at DNS resolution. A Python 3.10+

syntax or installation target is not a runtime-support claim. The offline acceptance command records whichever actual environment executes it:

```
python scripts/accept_release.py --output /tmp/new-python-evidence
```

This creates a fresh isolated environment, installs only the local wheel, checks the retained cases, exercises the installed command, and saves its receipts. The temporary environment is removed after those receipts are written. Python acceptance is not labeled as Lean acceptance.

H.6 Remaining decisions and limits

Public code and document licenses remain the author's unmade publication decision. No license is inferred from a wish to release openly, and no GitHub, PyPI, or other public upload has been performed. A later permission or metadata change requires an appropriate artifact rebuild and renewed checks.

The external parser and source emitter are tested rather than fully formally verified. Replay uses Lean's own kernel against the pinned imports, not an independently implemented checker or a fresh rebuild of all Mathlib. Search and certificate size can be exponential. Main-library integration and authoritative Paper III imports remain separate tasks.

These limits do not require another foundational development cycle. They delimit what the standalone component proves, what its release checks establish, and which decisions remain outside the mathematics.

References

[1] Komorebi P. *Constructive Repair v0.5.0: Proof Carrying Branch Search*. Structural tree checker, exact rational producer, kernel proofs, input echoes, and reproducible evidence, 19 October 2026.

[2] Komorebi P. *Constructive Repair v0.4.0: Costed Rational Menus and Rejection Covers*. Exact producer, checked cover theorems, typed certificates, verification records, and original-input echoes, 19 October 2026.

[3] Komorebi P. *Constructive Repair v0.3.0: Exact Rational Certificate Bridge*. Python producer, Lean certificate modules, fixed-input records, fresh compilation and same-kernel replay evidence, 19 October 2026.

[4] Komorebi P. *Constructive Repair: Lean Continuation v0.2.0*. Formal source release, 19 October 2026. Ten proof modules, declaration registry, verification records, and reproducibility scripts. Historical archive: `Komorebi_Constructive_Repair_Lean_v0.2.0.zip`. Exact proof snapshots and its hash are retained in the current provenance index.

[5] Komorebi P. *Constructive Repair Calculus: Exact Audit v0.1.0*. Mathematical audit, ordinary proofs, exact Python implementation, and case records, 19 October 2026. Preserved in the inherited material of the formal source release.

[6] Komorebi P. *Projector Resolved Seed Calculus I*. Traditional monograph in Operation Komorebi. Section 11, realization export and semantic realization criterion. Source context for the preservation contract.

[7] Komorebi P. *Projector Resolved Seed Calculus III: Typed Transfer, Nonlinear Packets, and Obstruction Theory*. Modern monograph, 8 October 2026. Sections 21–24, correction maps, cokernel obstructions, nonlinear correction, and filtered lifting.

[8] Komorebi P. *Toward Arithmetic Packet Realization*. Research companion to Paper XI. Restricted constructions, failed hosts, and the remaining obligations of Conjecture 66. Source context for the distinction between finite comparison and arithmetic existence.

[9] Python Software Foundation. *Python 3.13 Library Reference: io, Text Encoding*. Documentation of locale-dependent defaults and explicit encoding selection. `https://docs.python.org/3.13/library/io.html#text-encoding`.