

CORE SERIES IX

Euler Congruences

Completion, factorial strengthening, and sharp dyadic geometry

Lean and evidence guide

Komorebi P.

Monograph reading edition / 10 October 2026

Reading scope

This guide records the formal coverage of the final Euler paper and teaches the retained arithmetic, finite obstruction, polynomial tower, quantitative deformation, and analytic regularity proofs. The opening chapter maps the strengthened congruences to their existing Lean endpoints and identifies the new proofs that remain to be formalized. The complete original teaching chapters, exact source excerpts, and named-source catalog follow.

The Lean compilation evidence is inherited. Fresh work in this edition reconciles source hashes, retained logs, declaration inventories, source syntax, and strict axiom records. Lean and Lake are unavailable in the present environment; no new compiler run is claimed. The new factorial and mixed-difference theorems are proved in the main paper, with their proof and computation evidence supplied separately, but have no compiled Lean endpoint in this release.

Contents

1 The strengthened Euler paper and its exact formal coverage **4**

1.1 One normalization for the paper and the guide 4

1.2 Current claim map 4

1.3 What the Nishibuchi capstone actually proves 5

1.4 The four-packet theorem already passed the row threshold 6

1.5 The strongest homogeneous congruence to formalize next 7

1.6 General prime-power closure: a reusable target 7

1.7 Divided powers, factorial gains, and equality 8

1.8 Analytic consequences require their own bridge 9

1.9 Fresh review, inherited compilation, and reproducible replay 10

1.10 A clean completion criterion for the new formal layer 10

2 How to read the formal development **12**

2.1 Statements, constructions, audits, and identifiers 12

2.2 Reading the Lean notation 12

2.3 What the retained proof evidence establishes 13

3 Exact coordinates before arithmetic estimates **14**

3.1 Residue projectors are coefficient filters 14

3.2 Triangular reconstruction constructs the solution 14

3.3 EGF multiplication is Hurwitz convolution 16

3.4 Finite differences and the Mahler basis 16

3.5 Finite units and Chinese remainders 17

4 From packet coefficients to Nishibuchi's conjecture **19**
4.1 Rational precision includes zero . . . 19
4.2 A Newton expansion converts differences into a period . . . 20
4.3 Packet address and degree address are different indices . . . 20
4.4 The completion theorem exposes its full range . . . 21

5 Reconstruction, sharp dyadic displacement, and isometry **23**
5.1 Odd reconstruction separates identity from precision . . . 23
5.2 The dyadic transform has its own coefficients . . . 23
5.3 Dominance gives an exact valuation . . . 25
5.4 The normalized packet extends to a p-adic isometry . . . 26

6 Finite obstructions and a common correction **27**
6.1 Residuals become quotient classes . . . 27
6.2 Two solvable equations need one shared digit . . . 28
6.3 A nonempty fiber is a kernel torsor . . . 29

7 Actual polynomial Taylor expansion and lift counts **31**
7.1 A quadratic remainder is proved algebraically . . . 31
7.2 One nonlinear step becomes a finite digit equation . . . 31
7.3 Digits correspond to residue lifts, not all integer representatives . . . 33

8 Compatible integer towers and p-adic endpoints **34**
8.1 The tower is constructed, not supplied . . . 34
8.2 Completeness changes the coefficient domain . . . 34
8.3 Uniqueness requires a square solved system . . . 35
8.4 Coefficient stability retains the promised precision . . . 36

9 Local charts and prescribed parameter paths **37**
9.1 The solved block is the correct Jacobian . . . 37
9.2 The continuous chart includes both directions . . . 37
9.3 An arbitrary p-adic parameter is reached by integer digits . . . 38
9.4 A full-rank system admits a suitable coordinate split . . . 39

10 Weighted inverse bounds and quantitative root motion **41**
10.1 A fraction-field bound has an integral certificate . . . 41
10.2 Polynomial maps become contractions after scaling . . . 41
10.3 The threshold and the uniqueness ball are explicit . . . 42

10.4 Coefficient perturbation has a first-order formula . . . 43
10.5 Parameter and coefficient motion share one forcing term . . . 44

11 Actual derivatives and analytic local graphs 46
11.1 The polynomial Jacobian becomes a strict derivative . . . 46
11.2 The strict local inverse is the analytic inverse . . . 46
11.3 The augmented derivative exposes the sign . . . 47
11.4 The graph is exact in a product neighborhood . . . 47
11.5 A supplied continuous root selection acquires regularity . . . 49
11.6 Integral coefficients are evaluated in the fraction field . . . 50

12 Evidence replay and limits of the final edition 52
12.1 What was freshly repeated . . . 52
12.2 The negative controls mark mathematical boundaries . . . 52
12.3 How to perform a new compiler replay . . . 53
12.4 Unclosed obligations and recovered history . . . 53

13 Current manuscript-to-guide navigation 55
13.1 Strengthening and worked-example routes . . . 56

14 The retained verification frontiers 58

15 Latest polynomial dependency map 59

16 Complete named-source catalog 61

1 The strengthened Euler paper and its exact formal coverage

The final paper brings together the completed external conjecture, the retained all-row four-packet theory, and stronger congruences proved during the present continuation. Those layers have different verification evidence. The original Nishibuchi capstone, the four-packet step-sensitive bound, the exact first-difference periods, and the isometry construction have preserved Lean sources and successful historical compilation records. The new general prime-power closure, mixed factorial estimates, exact higher-order laws, and Taylor-radius results have mathematical proofs and recorded audits, but no compiled Lean endpoint in this edition.

The distinction is attached to claims, rather than to whole documents. A new proof may include a previously formalized special case. Conversely, an inherited endpoint about a polynomial analytic graph does not certify the new Euler interpolation radius merely because both use p-adic analysis. The tables and contracts below identify the actual overlap and the remaining work.

1.1 One normalization for the paper and the guide

For a positive modulus N and an address $0 \le j < N$, define

$$G_{N,j}(t) = j! \sum_{n\ge 0} \frac{t^{Nn}}{(Nn+j)!}, \qquad G_{N,j}(t)^{-1} = \sum_{n\ge 0} C_{N,j}(n) \frac{t^{Nn}}{(Nn)!}.$$

The constant coefficient of both normalized series is one. The original generalized Euler number at degree Nn is

$$E_{Nn}^{(N,j)} = j!\, C_{N,j}(n).$$

Thus `Congruential.coeff N j n` corresponds to $C_{N,j}(n)$ and `Congruential.unnormalized N j n` corresponds to $E_{Nn}^{(N,j)}$. The source `NishibuchiCompletion.generalizedEuler N hN j d` instead uses the literal Euler degree d. Its theorem `generalizedEuler_at_packet` is the bridge between these indices.

Write $\Delta_H f(n) = f(n+H) - f(n)$. The source `Mahler.difference H k f n` is the homogeneous difference $\Delta_H^k f(n)$. A mixed difference applies several possibly unequal steps. Existing homogeneous endpoints cannot be substituted for a mixed endpoint without a proof relating those operators.

The notation $v_p(0) = +\infty$ makes lower bounds zero safe on paper. In the sources, `GlobalEulerBoundary.precision p r x` supplies the corresponding zero-safe rational predicate. Actual integral lifts express the same congruence as divisibility by $(p : \mathbb{Z}_p)^r$. Lean's rational valuation of zero is not itself an infinite-valued function. Exact valuation endpoints therefore include a separate nonzero conclusion or require a nonzero hypothesis. That distinction is essential when translating a lower bound into a typed target.

1.2 Current claim map

Main-paper topic	Retained Lean coverage	Present status
Nishibuchi Conjecture 3.9, all permitted primes and addresses	`NishibuchiCompletion.conjecture_3_9`; original integrality and degree-shift bridges	Historically compiled. Source and evidence reconciled anew.
Four-packet all-row Kummer divisibility	`DyadicFourAllRows.normalized_precision`	Historically compiled; includes row zero and every difference order.

Main-paper topic	Retained Lean coverage	Present status
Four-packet common-step precision, including the address factorial	`DyadicFourFinalStrengthening.normalized_step_precision` and `unnormalized_step_precision`	Historically compiled; exponent $k(1+v_2H)$, with $v_2(j!)$ for the original numbers.
Four-packet sharp first differences, least periods, and isometry	`DyadicFourLeastPeriods` and `DyadicFourIsometry`	Historically compiled under the stated address and positive-step hypotheses.
All prime-power moduli and all mixed steps	No matched general endpoint	Proved in the new continuation; not Lean formalized here.
Four-packet factorial gain and address surpluses	No matched higher-order endpoint	Proved in the new continuation; not Lean formalized here.
Exact higher differences at addresses zero and one; mixed leading-digit criterion	No matched endpoint	Proved in the new continuation; not Lean formalized here.
Address-two stronger growth and exact power-of-two orders	No matched endpoint	Proved in the new continuation; not Lean formalized here.
Small-rank metric results beyond $N = 4$; all-rank lower-half metric theorem	No matched endpoint	Proved in the new continuation; not Lean formalized here.
Exact Euler Taylor radii at addresses zero, one, and two	No matched endpoint	Proved in the new continuation; not implied by the retained polynomial regularity modules.
General exact higher-order law at addresses two and three; unrestricted upper-half metric law	No completed endpoint	The stated unrestricted laws remain open or are refuted in the main paper.

The chapter titles in this table identify the mathematical topics. The crosswalk at the end of this guide now resolves the inherited statement index against the current monograph, with physical PDF pages and current theorem numbers. The historical crosswalk and its older paper identity remain in the source archive. Separate routes identify the strengthening statements and the eight exact worked examples.

1.3 What the Nishibuchi capstone actually proves

Let p be prime, let $m > 0$ satisfy $m \mid p-1$ or $m = 2$, put $q = \operatorname{lcm}(2, p-1)$, and let $0 \le j < mp$. The capstone proves p-integrality of every original packet coefficient and, for every $r \ge 1$, the degree-indexed congruence

$$E^{(mp,j)}_{mpn+qp^r} \equiv E^{(mp,j)}_{mpn} \pmod{p^r} \qquad (n \ge r).$$

Its existential formulation supplies the positive threshold $n_0 = r$, so it also supplies $n_0 \le r$. It does not assert the smallest possible threshold. The source has no premise asking the caller to assume source flatness, a weighted certificate, or the desired congruence.

The divisor branch supplies the stronger sufficient row condition $r \le mn$. The arithmetic parameter reduction isolates the exceptional case $p = 2$, $m = 2$. At $p = 2$, $q = 2$; replacing q by $p - 1 = 1$ would change the conjecture. The degree-shift theorem proves that the packet step

$$p^{r-1}\frac{q}{m}$$

produces exactly the Euler-degree increment qp^r.

The 0.11.0 completion checkpoint contains 79 active proof modules, 836 theorem commands, 162 definitions, and two abbreviations, giving 1,000 named axiom reports. Its recorded nine new modules account for 83 theorem commands, 14 definitions, and one abbreviation. These are source-command counts within that checkpoint, not counts of distinct mathematical discoveries. The broader retained final source collection has 1,876 declaration/source-version rows under 199 canonical source files; it includes subsequent strengthening and lifting frontiers, with overlaps and retained revisions. The two sets must not be added.

The completion record retains successful source compilation, axiom checks, 37 negative controls, and 25 extension infrastructure tests. Its capstone axiom report contains only `propext`, `Classical.choice`, and `Quot.sound`. These historical outcomes remain historical in the present edition. The separate source and log reconciliation executed now does not rerun the Lean kernel.

1.4 The four-packet theorem already passed the row threshold

The earlier completion extracted a bound with exponent $\min(k, n)$. The retained strengthening subsequently replaced that cap by k using the unshifted operator

$$B = \frac{D^4 - 1}{2}.$$

Its charged reciprocal theorem proves that the source's flatness survives inversion, with the charge shift made explicit. The module `DyadicFourAllRows` then extracts the difference directly at degree $4n$. It has no $k \le n$ assumption and no artificial extraction at a negative degree.

The endpoint `DyadicFourFinalStrengthening.normalized_step_precision` states

$$v_2(\Delta_H^k C_{4,j}(n)) \ge k(1 + v_2 H) \qquad (j < 4,\ n, k \in \mathbb{N},\ H \ge 1),$$

with its zero-safe precision interpretation. The original-number endpoint adds $v_2(j!)$. For positive H, this is exactly the four-packet homogeneous specialization of the general mixed-step estimate in the new paper. The source also includes zero step using its finite natural valuation `padicValNat 2 0 = 0`: every positive-order zero-step difference vanishes. At order zero, the statement is just the appropriate integrality assertion. Thus the source never forms the undefined paper expression $0 \cdot v_2(0)$.

For these four packets, the conjectured dyadic period already starts at row zero in the retained Lean source. The literal degree endpoint is `all_rows_period_degree`. The exact-period development goes further and proves

$$v_2(E^{(4,j)}_{4(n+H)} - E^{(4,j)}_{4n}) = v_2 H + B_j, \qquad (B_0, B_1, B_2, B_3) = (1, 1, 5, 3),$$

for positive H. It follows that the least positive row period modulo 2^r is

$$2^{\max(0, r - B_j)}.$$

The normalized bonuses are $(1, 1, 4, 2)$. The isometry endpoint uses a translated and rescaled packet, not the raw original sequence. Undoing its normalization gives the original image ball $j! + 2^{B_j}\mathbb{Z}_2$.

The new edition preserves these achievements and their formal evidence. The new work extends the modulus range and operator patterns, improves higher-order exponents by factorial and address gains, and derives exact higher-order and analytic consequences. It is not necessary to relabel an already formalized four-packet statement as an unformalized new theorem in order to explain that extension.

1.5 The strongest homogeneous congruence to formalize next

For $j \in \{0, 1, 2, 3\}$, $n \geq 0$, and $k, H \geq 1$, the final main-paper congruence is

$$\sum_{i=0}^{k} (-1)^{k-i} \binom{k}{i} E^{(4,j)}_{4(n+iH)} \equiv 0 \pmod{2^{\,v_2(k!)+kv_2H+A_j(k)}},$$

where

$$A_0(k) = A_1(k) = k, \qquad A_2(k) = \max\{k+4,\, 2k + \lfloor k/2 \rfloor + 1\}, \qquad A_3(k) = k + 2.$$

This displayed exponent includes the address factorial. Using it directly for the normalized $C_{4,j}$ would introduce an extra digit at addresses two and three. The lower bound holds at every base row, including zero. Equality holds for every positive order and positive homogeneous step at addresses zero and one. At address two, the exact laws proved in the paper cover orders that are powers of two, including one and two, with the explicit small-order exceptions stated there. This is not a theorem asserting equality at every address-two order.

A direct formal target should use `Mahler.difference H k (Congruential.unnormalized 4 j) n`, the natural exponent `padicValNat 2 k.factorial + k * padicValNat 2 H + A j k`, and `GlobalEulerBoundary.precision`. Its hypotheses are $j < 4$, $0 < k$, and $0 < H$. That target reuses the original definitions and preserves the paper's quantifiers. It should not replace the rational sequence by a separately named experimental recurrence without a coefficient-identity bridge.

The proof of the new factorial term is a divided-power argument. At addresses zero and one, its external arithmetic input is the exact classical Frobenius congruence for ordinary Euler numbers, in the form checked in the main paper's source record. The Bernoulli central-factorial argument at address two proves its divided-power integrality internally. A Lean completion will need the exact ordinary-Euler input formalized or imported as a proved theorem with a matched statement. A custom axiom standing in for that input would change the formal coverage and must not be hidden under an unchanged completion label.

1.6 General prime-power closure: a reusable target

For $N = p^a$, $a \geq 1$, and $j < N$, put $C(n) = C_{N,j}(n)$. The general theorem is most naturally phrased for the sign-adjusted sequence

$$A(n) = (-1)^n C(n).$$

For positive steps $H_1, \ldots, H_k$, it proves

$$v_p(\Delta_{H_1} \cdots \Delta_{H_k} A(n)) \geq k + \sum_{i=1}^{k} v_p H_i.$$

For the raw sequence, define $T_H f(n) = f(n+H) - (-1)^H f(n)$. The equivalent original-number statement is

$$v_p(T_{H_1} \cdots T_{H_k} E^{(N,j)}(n)) \geq v_p(j!) + k + \sum_{i=1}^{k} v_p H_i, \qquad E^{(N,j)}(n) = E_{Nn}^{(N,j)}.$$

At $p = 2$, the sign adjustment and an integral charge shift give the ordinary mixed-difference theorem for $C_{N,j}$ itself. The degenerate dyadic modulus $N = 1$ has $C(n) = (-1)^n$ and is handled directly. At an odd prime, ordinary unsigned differences of the raw sequence need not satisfy the same estimate: $C_{N,0}(0) = 1$ and $C_{N,0}(1) = -1$, so the first unit-step difference is -2.

A useful formalization order is:

1. Prove integral Hurwitz coordinates are closed under products, derivatives, and inversion of a unit constant coefficient. Retained `HurwitzConvolution`, `HurwitzInverse`, and `DworkAlgebra` provide reusable constructions; their exact coefficient domains must be matched before reuse.
2. Prove interior binomial divisibility $p \mid \binom{p^a}{r}$ for $0 < r < p^a$, and record the integral coefficients in the charged operator product formula.
3. Generalize the charged reciprocal closure from D^4 to D^N, keeping the source and inverse charges visible. The critical induction isolates the unique term containing the highest inverse order. No hypothesis $k \leq n$ is permitted.
4. Prove flatness of the actual source $g_{N,j}(n) = \prod_{i=1}^{j}(1 + (N/i)n)^{-1}$. Every slope N/i belongs to $p\mathbb{Z}_p$ because $1 \leq i < N = p^a$. Prove the reciprocal affine-factor finite-difference identity before taking products.
5. Decode the reciprocal coefficient at a nonnegative packed degree, then transfer to rational precision and to the original factorial normalization.
6. Prove mixed-step amplification, with independent step valuations added, and specialize to homogeneous steps only after the general target is established.

This sequence isolates the new universal arithmetic infrastructure. The retained $N = 4$ charged closure proves a genuine special case, but its source signature does not quantify over every prime-power modulus. A future generalized file should remain a new endpoint until its statement and axiom closure have been checked.

1.7 Divided powers, factorial gains, and equality

For four-packet normalized coefficients, the factorial theorem states

$$v_2(\Delta_{H_1} \cdots \Delta_{H_k} C_{4,j}(n)) \geq k + v_2(k!) + \sum_{i=1}^{k} v_2 H_i + \beta_j, \qquad (\beta_0, \beta_1, \beta_2, \beta_3) = (0, 0, 3, 1),$$

for $k \geq 1$. Multiplication by $j!$ changes the address surpluses to $(0, 0, 4, 2)$ for the original numbers. At address two, the additional bound

$$v_2(\Delta_{H_1} \cdots \Delta_{H_k} C_{4,2}(n)) \geq 2k + v_2(k!) + \lfloor k/2 \rfloor + \sum_{i=1}^{k} v_2 H_i$$

combines with the first bound by taking their maximum.

The formal proof must distinguish a valuation lower bound from the evaluation of its leading unit. Divided-power integrality proves the lower bound. Exactness additionally requires a parity calculation after dividing by the claimed power of two. That calculation is where unequal steps can cancel. At

addresses zero and one, let

$$V = k + v_2(k!) + \sum_{i=1}^{k} v_2 H_i.$$

The new paper proves exact valuation V if k is odd or the number of even steps is even. If k is even and the number of even steps is odd, the leading digit cancels and the valuation is at least $V + 1$. That last case does not carry a uniform formula for the deeper valuation.

For a homogeneous step, the number of even steps is either zero or k. The criterion therefore always gives exactness at addresses zero and one. This explains why a true homogeneous equality can fail when a single step is changed. For example,

$$\Delta_1 \Delta_2 C_{4,0}(0) = -33728,$$

has valuation 6, whereas the mixed baseline is 4. The same pattern at address one has value $-150208/585$ and valuation 6. These examples should become exact rational negative controls for a proposed unconditional mixed equality; they should not be used as a substitute for the general parity proof.

At address two, the exact mixed-step exponent is $4 + \sum v_2 H_i$ at order one, $6 + \sum v_2 H_i$ at order two, and

$$2k + v_2(k!) + k/2 + \sum v_2 H_i$$

at powers of two $k \geq 4$. The split between these three targets is deliberate. A theorem with the final expression at order one would not state the correct sharp exponent. General exact higher-order laws at addresses two and three remain outside the completed formal and mathematical register.

The proof route uses two Gaussian-scaled Euler or Bernoulli umbrae over $\mathbb{Q}_2(i)$ and a descent back to rational coefficients. Lean obligations include the base change, the Gaussian scalar identities, the divided-power product and composition laws, and the rational descent. Treating the symbolic complex expression as though it were already a rational integral element would omit a substantive bridge. The unbounded exactness subsequence at address two also needs its parity proof; a computed prefix through order 512 does not supply that bridge.

1.8 Analytic consequences require their own bridge

The factorial Newton estimates are converted into Taylor coefficients using the Stirling change of basis, with convergence and uniformity justified. They imply an exact Taylor radius 2 at every integral center for addresses zero and one. The address-two bounds, together with the sharp power-of-two coefficient subsequence, give exact radius $2^{5/2}$. The address-three result gives a radius of at least 2; its exact radius is not claimed.

A formal endpoint should specify the coefficient field, the absolute value normalization $|2|_2 = 1/2$, the chosen center, and the meaning of radius. If the radius is viewed through an extension containing points outside $\mathbb{Q}_2$, the relevant convergence statement needs that ambient extension specified. The root-test coefficient rate itself can be stated independently. A generic Kummer bound alone does not establish these analytic radii. The retained `PolynomialAnalyticEvaluation`, `PolynomialAnalyticBlocks`, and local implicit-graph modules concern different objects and do not close this argument.

A future Lean proof should separately verify Newton interpolation equals the original rational packet on natural inputs, the Newton-to-Taylor coefficient identity, the coefficient lower bounds, the exact subsequence supplying the upper radius bound, and convergence on the asserted disk. No result about finitely many numerical coefficients can be promoted to the exact radius without that unbounded subsequence argument.

1.9 Fresh review, inherited compilation, and reproducible replay

The present static review reran the retained source and evidence reconciler. It matched all 15 inherited frontiers, 644 source-hash comparisons, and 713 recorded log-hash comparisons. It reproduced the 1,876-row declaration/source-version catalog byte for byte, resolved all 202 links in the retained 62-statement index, and matched the preserved manuscript source hashes. At the latest polynomial frontier it freshly inspected 55 proof modules, eight support modules, and 13 contract modules, and matched all 515 strict axiom records to the exact expected names. Twelve retained negative-control logs have matching source and log hashes, recorded nonzero exits, and the expected diagnostics. The unchanged Python parser regression was also rerun successfully.

These results are recorded in `audit/current_reconciliation/PROVENANCE_AUDIT.json`. No Lean or Lake executable was found, and no compiler was executed. No fresh rejection of a Lean negative control, fresh project build, or fresh Mathlib compilation is claimed. Four older affine logs lack a historical digest; their current matching fingerprints retain that provenance limit. Two stale inventory scalars are recorded separately while their archival files remain unchanged.

All source locators in the retained catalog are relative to the root `preserved/PRSC_IX_Consolidated_Manuscript`. A locator containing `!` denotes an exact member of the specified ZIP. For example, the completion endpoint is inside `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip`, while later four-packet endpoints are plain files under `strengthening/formalization`. The complete root is preserved with this package so that the following fresh static check can be repeated from the guide source directory:

```
python3 audit_sources.py \
  --source ../../preserved/PRSC_IX_Consolidated_Manuscript \
  --output audit/a_new_static_run
```

The PDF build is separate:

```
python3 build.py
```

It uses XeLaTeX, not Lean, and needs only this guide's editable sources, generated tables, and excerpts. The historical compiler replay drivers and portable-runtime instructions are preserved in the source root and described in the replay chapter. A new kernel run requires the pinned Lean 4.34.0-rc2 and Mathlib revision `aeb72f1e9981b6d35b426b16dca7759be3e7ba06`. Retain its new output directory and all compilation, axiom, contract, and control results. Do not overwrite the inherited logs.

1.10 A clean completion criterion for the new formal layer

The next formal release can legitimately mark a new theorem compiled only after its exact statement is bound to a source hash, its dependencies are identified, the intended positive examples compile, the relevant false strengthenings fail for the expected reason, and its axiom closure is recorded. The important controls include unsigned raw differences at odd primes, an unconditional mixed equality, a homogeneous exactness statement at zero step, the missing address factorial, and a universal upper-half metric pattern refuted by the recorded higher-rank examples.

The proof files should contain no placeholder proofs, custom axioms for the desired conclusion, or precomputed arrays replacing the actual coefficient sequence. For imported classical arithmetic, the release must identify the exact theorem used and prove or explicitly bound its bridge to the paper's convention. The endpoint register must distinguish common-step estimates from mixed-step estimates and finite co-

efficient checks from universal statements. The inherited capstone remains complete while this stronger formal layer is developed; its scope does not expand automatically with the new manuscript.

2 How to read the formal development

The development uses two complementary mathematical languages. Arithmetic packets begin with sequences and formal power series, then transport divisibility through exact coordinate changes. Polynomial lifting begins with a finite residual equation, passes to compatible digit towers, and finally enters complete p-adic and analytic spaces. The transition between these languages is explicit: a rational coefficient has to be shown integral before it becomes a p-adic integer; an integer congruence tower has to be shown Cauchy before it becomes a root; a formal polynomial derivative has to be identified with a strict Fréchet derivative before it supports an analytic inverse theorem.

This guide follows those transitions. Read the *contract* before the excerpt: it states the coefficient domain, ambient dimensions, and hypotheses that may be declared earlier in the source. The excerpt is an exact contiguous selection from a preserved Lean file. Its line range and source hash are supplied in the accompanying excerpt inventory. The proof reading then explains what the displayed tactics accomplish mathematically. A theorem imported from another module is a dependency, not a hidden new assumption; a hypothesis supplied to a theorem remains a mathematical assumption even when its name does not appear in the theorem's axiom closure.

2.1 Statements, constructions, audits, and identifiers

The retained preceding consolidated manuscript links 62 statements to 202 named source declarations. That index resolves names and source identities. It does not prove that every informal sentence equals a Lean statement. Several declarations may establish one displayed theorem, while one source theorem may support several expository consequences. The restored Bernoulli specialization has a classical formal-series derivation in the manuscript, but no dedicated Lean bridge was matched. The manuscript's open problems retain their open status.

The full declaration index has 1,876 rows. A row records a declaration kind, qualified name, and entire source-file hash. Byte-identical copies are collapsed; different source versions remain separate. There are 1,863 distinct kind/name pairs, including 1,542 distinct names whose command kind is `theorem`. These are identifier counts, not counts of independent mathematical results. The rows also contain constructions, statement contracts, examples and control support. Thirteen names have more than one source version.

At the latest polynomial regularity checkpoint, the 55 proof modules contain 322 explicit theorem commands, 76 definitions and seven abbreviations. Eight positive control modules add 61 theorem commands and 14 definitions. Thirteen statement-contract modules add 32 theorem commands, two definitions and one abbreviation. The resulting 515 named declarations have matching retained axiom records. The frequently quoted number 415 is the total theorem-command count across these three groups. It includes the 61 control and 32 contract commands.

2.2 Reading the Lean notation

`Fin d → R` means a vector with exactly d coordinates over R. In rectangular lifting, d counts unknowns and e counts equations. In a parameter split, `Fin (e+k) → R` separates e solved coordinates and k free parameters. A theorem using `Fin (m+k)` may instead have m solved coordinates and e equations; the distinction matters when a fiber has p^{m-e} elements.

`A →ₗ[R] B` is a linear map, while `A →L[R] B` is a continuous linear map. An arrow decorated `≃L[R]` is a continuous linear equivalence. The analytic derivative is a continuous linear map; a raw matrix becomes

one through the explicitly defined `matrixCLM`. `IsUnit J` for a square matrix means it is invertible in the matrix ring. It does not mean each matrix entry is a scalar unit.

`ZMod p` is the residue ring. The source uses `Fact p.Prime` when it needs a finite field, rank, or p-adic objects. Some affine and integer polynomial statements require only $p > 0$ or $q > 0$; their quotient algebra does not require primality. A hypothesis `1 ≤ L` supplies the depth needed for polynomial first-order linearization. The affine depth-zero case is valid because an affine map has no quadratic Taylor remainder.

`∃! b, P b` means there is exactly one object satisfying the full predicate P. A local root theorem's predicate includes its residue ball. It does not imply that the polynomial has only one root in the entire field. Likewise, an analytic graph theorem supplies open parameter and solved-coordinate neighborhoods; global uniqueness is outside its conclusion.

2.3 What the retained proof evidence establishes

The original compiler runs use Lean 4.34.0-rc2 and the recorded Mathlib revision `aeb72f1e9981b6d35b426b16dca7759be3e7ba06`. Their permitted axiom closure is `propext`, `Classical.choice`, and `Quot.sound`. This is a check for standard foundational dependencies and absence of a recorded admitted-proof axiom. It is not a claim that the proofs avoid classical choice, nor that every manuscript hypothesis has been discharged.

Fresh reconciliation reproduced all 15 retained evidence frontiers: 644 source-hash comparisons and 713 recorded log-hash comparisons match. Four older standalone affine logs have no digest in their original report; they are fingerprinted in the later index, so their historical provenance remains weaker. The fresh strict parser matched all 515 latest named axiom records to source declarations, and the unchanged Python parser regression rejected duplicated, mismatched, and forbidden-axiom records. These are static checks. Kernel checking remains the inherited compiler evidence.

3 Exact coordinates before arithmetic estimates

3.1 Residue projectors are coefficient filters

Contract. For an additive commutative monoid R, a positive modulus N, and a sequence $a : \mathbb{N} \to R$, the residue packets sum to a. The coefficient-level idempotent and orthogonality laws need only a zero element.

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/ResiduePackets.lean`.

Lines 38–46; `KomorebiPaperIX.ResiduePackets.packet_reconstruction`.

```
theorem packet_reconstruction [AddCommMonoid R] {N : ℕ} (hN : 0 < N) (a : ℕ → R) :
    ∑ j ∈ Finset.range N, packet N j a = a := by
  funext n
  simp only [Finset.sum_apply, packet]
  rw [Finset.sum_eq_single (n % N)]
  · simp
  · intro b _ hb
    simp [Ne.symm hb]
  · simp [Nat.mod_lt n hN]
```

Proof reading. Function extensionality reduces the sequence identity to one coefficient n. The only nonzero summand is indexed by n mod N. `Finset.sum_eq_single` separates that summand from the rest; `Nat.mod_lt n hN` proves its address lies inside the chosen finite range. Positivity is load-bearing: it justifies the finite address set. There is no division by N and no root-of-unity hypothesis.

A residue projector is a selector on the degree address. Applying it twice keeps the same coefficients. Projectors with distinct addresses cannot both retain the same degree. Refinement to a multiple modulus groups fine addresses according to their coarse remainder. These are exact identities before reduction modulo a prime. The power-series versions use coefficient extensionality, so the same proof mechanism acts on `PowerSeries R`.

3.2 Triangular reconstruction constructs the solution

The source does not assume that an infinite triangular system already has a solution. It defines forward substitution recursively, using only indices $k < n$ at row n. The `termination_by n` declaration verifies the recursion decreases. Over a field the formula is defined even when a diagonal vanishes, but the row equation and uniqueness theorem explicitly require nonzero diagonal entries.

Contract. F is a field, T is an infinite coefficient array, every T_{nn} is nonzero, and a satisfies every finite row equation. Then a equals the recursively constructed solution.

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/Triangular.lean`.

Lines 35–49; `KomorebiPaperIX.Triangular.solve_unique`.

```
theorem solve_unique (T : ℕ → ℕ → F) (b a : ℕ → F)
    (hT : ∀ n, T n n ≠ 0)
```

```
    (ha : ∀ n, (∑ k : Fin n, T n k * a k) + T n n * a n = b n) :
    a = solve T b := by
  funext n
  induction n using Nat.strong_induction_on with
  | h n ih =>
    have hs : (∑ k : Fin n, T n k * a k) = ∑ k : Fin n, T n k * solve T b k := by
      apply Finset.sum_congr rfl
      intro k _
      rw [ih k k.isLt]
    have hr := ha n
    rw [hs] at hr
    have he := solve_row T b hT n
    exact (mul_left_cancel₀ (hT n)) (add_left_cancel (hr.trans he.symm))
```

Proof reading. Strong induction first replaces each earlier unknown a_k by the recursive value. The two row equations then have identical earlier sums. Cancelling the sum leaves equality after multiplication by T_{nn}; nonzero diagonal cancellation gives equality of the current entries. Function extensionality assembles all rows. The argument needs no convergence or boundedness of an infinite matrix. Each row is a finite calculation.

Contract. For integer coefficients and integer right-hand side, the product $D_n = \prod_{r=0}^{n} T_{rr}$ clears the rational solution's denominator at row n. Every diagonal remains nonzero.

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/Triangular.lean`.

Lines 106–127; `KomorebiPaperIX.Triangular.solve_denominator_integral`.

```
theorem solve_denominator_integral (T : ℕ → ℕ → ℤ) (b : ℕ → ℤ)
    (hT : ∀ n, T n n ≠ 0) (n : ℕ) :
    Integral ((denominator T n : ℚ) * solve (fun i j => (T i j : ℚ)) (fun i => (b i : ℚ)) n) := by
  induction n using Nat.strong_induction_on with
  | h n ih =>
    cases n with
    | zero =>
      simp only [denominator_zero, solve_zero]
      have h0 : (T 0 0 : ℚ) ≠ 0 := Int.cast_ne_zero.mpr (hT 0)
      rw [mul_div_cancel₀ _ h0]
      exact integral_int _
    | succ n =>
      rw [solve_step, denominator_succ, Int.cast_mul]
      have hn : (T (n + 1) (n + 1) : ℚ) ≠ 0 := Int.cast_ne_zero.mpr (hT (n + 1))
      rw [mul_assoc, mul_div_cancel₀ _ hn]
      rw [mul_sub, Finset.mul_sum]
      apply integral_sub (integral_mul (integral_int _) (integral_int _))
      apply integral_sum
      intro k _
      rw [mul_left_comm]
      exact integral_mul (integral_int _)
        (integral_promote (denominator_dvd T (Nat.le_of_lt_succ k.isLt)) (ih k k.isLt))
```

Proof reading. The induction separates row zero from a successor row. At a successor, $D_{n+1} = D_n T_{n+1,n+1}$ cancels the new denominator. Earlier denominators divide D_n, allowing `integral_promote` to place every earlier term under one common integer multiplier. The proof establishes an integral multiple of the actual recursive solution. It does not assert that the product is the smallest denominator. Negative diagonal entries are allowed, since only nonvanishing and divisibility are used.

3.3 EGF multiplication is Hurwitz convolution

Contract. Over a characteristic-zero field in the source coordinates, encoding divides the nth sequence entry by $n!$, and decoding multiplies the power-series coefficient by $n!$. The two operations are inverse.

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/EGFCoordinates.lean`.

Lines 52–60; `KomorebiPaperIX.EGF.egf_product`.

```
theorem egf_product (h a : ℕ → K) (n : ℕ) :
    decode (encode h * encode a) n =
      ∑ r ∈ Finset.range (n+1), (n.choose r : K) * h r * a (n-r) := by
  simp only [decode, coeff_mul, Finset.Nat.sum_antidiagonal_eq_sum_range_succ_mk,
    encode_coeff, Finset.sum_mul]
  apply Finset.sum_congr rfl
  intro r hr
  rw [Nat.cast_choose K (Finset.mem_range_succ_iff.mp hr)]
  ring
```

The binomial coefficient in the displayed convolution records the factorial change of coordinates. Ordinary power-series multiplication uses an unweighted coefficient convolution; the exponential coordinate multiplication uses $\binom{n}{k}$. It is therefore essential to distinguish a raw formal coefficient from its degree-indexed generalized Euler number. Replacing one by the other without restoring the factorial changes p-adic precision.

Proof reading. The proof unfolds the product coefficient over finitely many pairs with sum n, substitutes the encoded entries, and proves the factorial identity behind the binomial coefficient. Nonzero factorial casts justify field cancellation. The resulting law transports exact inverse identities and reconstruction statements into sequence coordinates. The formal series need not converge analytically.

3.4 Finite differences and the Mahler basis

Contract. R is a commutative ring. The finite Mahler identity reconstructs any sequence value $f(n)$ from its first $n + 1$ forward-difference coefficients; all sums are finite.

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/FiniteMahler.lean`.

Lines 55–57; `KomorebiPaperIX.Mahler.finite_mahler_reconstruction`.

```
theorem finite_mahler_reconstruction (f : ℕ → R) (n : ℕ) :
    f n = ∑ r ∈ Finset.range (n+1), coefficient f r * (n.choose r : R) := by
  simpa [coefficient, difference, nsmul_eq_mul, mul_comm] using shift_eq_sum_fwdDiff_iter 1 f n 0
```

The important change of basis is

$$f(n) = \sum_{k=0}^{n} \binom{n}{k} \Delta^k f(0).$$

The inverse relation expresses $\Delta^k f(0)$ as a signed binomial sum of the first $k+1$ values. No infinite Mahler-series convergence theorem is required for this finite identity. Later p-adic bounds put valuations on these exact coefficients and turn a finite coordinate theorem into precision estimates.

A weight condition on the kth coefficient controls higher differences by divisibility. The weighted difference modules formulate that condition explicitly and prove the translation by finite sums. A theorem about all orders therefore depends on the weight hypothesis for all the relevant coefficients; it cannot be inferred from a small computed prefix.

3.5 Finite units and Chinese remainders

Contract. For a prime p and $L \in \mathbb{N}$, the finite-unit module decomposes a unit modulo p^{L+1} into the source's torsion and principal components. Its uniqueness proof uses the uniquely lifted torsion representative and cancellation.

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/FiniteUnits.lean`.

Lines 134–144; `KomorebiPaperIX.FiniteUnits.decomposition_unique`.

```
theorem decomposition_unique (L : ℕ) (a w v : (ZMod (p^(L+1)))ˣ)
    (ha : w*v=a) (hw : w^(p-1)=1) (hv : residueUnits p L v=1) :
    w=omega p L a ∧ v=principal p L a := by
  have hr : residueUnits p L w=residueUnits p L a := by
    rw [← ha, map_mul, hv, mul_one]
  have he := omega_unique p L a w hw hr
  refine ⟨he, ?_⟩
  subst w
  apply mul_left_cancel (a := omega p L a)
  rw [ha]
  exact (finite_unit_decomposition p L a).1.symm
```

The displayed source theorem applies to prime p, including $p = 2$. It compares the reduced residue of w with that of a, invokes the uniquely lifted torsion representative, and then cancels that factor to identify the principal component. For odd primes the familiar coprime torsion/principal orders explain this structure, but the Lean endpoint uses its proved lifted-root uniqueness. The dyadic development additionally supplies the finer sign/principal decomposition in its separate module.

Contract. The displayed CRT equivalence identifies addresses modulo $2^a M$ with addresses modulo 2^a and M, where M is odd. Oddness proves the coprimality needed by the ring equivalence.

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/CRTPackets.lean`.

Lines 13–17; `KomorebiPaperIX.CRTPackets.addressEquiv`.

```
def addressEquiv (a M : ℕ) (hM : Odd M) :
    ZMod (2^a*M) ≃+* ZMod (2^a) × ZMod M :=
```

```
ZMod.chineseRemainder (by
  apply Nat.Coprime.pow_left
  exact (Nat.coprime_two_left).mpr hM)
```

Proof reading. The equivalence carries both directions and their inverse laws. Packet factorization then compares the retained degree address through that equivalence. The algebra concerns residue addresses and coefficients, not an analytic tensor-product completion. This is why the finite arithmetic layer can be reused without adding convergence claims.

4 From packet coefficients to Nishibuchi's conjecture

4.1 Rational precision includes zero

Contract. Fix a prime p. For a rational number x and an integer L, `precision p L x` means $x = 0$ or $v_p(x) \geq L$.

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/GlobalEulerBoundary.lean`.

Lines 18–18; `KomorebiPaperIX.GlobalEulerBoundary.precision`.

```
def precision (L : ℤ) (x : ℚ) : Prop := x=0 ∨ L≤padicValRat p x
```

The explicit zero alternative avoids relying on a finite-valued implementation of the valuation at zero. It also allows negative precision exponents: dividing an integral p-adic quantity by an integer normalization can reduce the bound below zero. The source's `precision` predicate is not divisibility inside $\mathbb{Q}$, where every nonzero scalar is a unit. Divisibility statements live in $\mathbb{Z}$, $\mathbb{Z}_p$, or a stated coefficient ring, and a separate bridge transports them to rational valuations.

Let $N = mp$ with $m > 0$, $m \mid p-1$, and $0 \leq j < N$. The unnormalized sequence is lifted to $\mathbb{Z}_p$ only after integrality is proved. Its higher differences use a packet step that is a multiple of $(p-1)/m$. The arithmetic core supplies a bound of the form

$$p^{\min(k,mn)} \mid \Delta^k_{u(p-1)/m} A_{N,j}(n).$$

Contract. The prime hypothesis, divisor condition $m \mid p-1$, positive m, and address bound $j < mp$ are all required. The sequence in the theorem is the actual integral lift of the unnormalized generating-series coefficients.

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/AllResidueEuler.lean`.

Lines 79–85; `KomorebiPaperIX.AllResidueEuler.global_unnormalized_bound`.

```
theorem global_unnormalized_bound (m j u k n : ℕ)
    (hm : m|p-1) (hmpos : 0<m) (hj : j<m*p) :
    (p:ℤ_[p])^(min k (m*n)) |
      difference (u*((p-1)/m)) k (unnormalizedLift p m j hm hmpos hj) n := by
  exact packet_divisibility p m hm hmpos
    (unnormalizedSeries (m*p) (Nat.mul_pos hmpos (Fact.out : p.Prime).pos).ne' j)
    (all_residue_flat p m hm hmpos j hj) (unnormalized_support (m*p) _ j) u k n
```

Proof reading. The theorem invokes `packet_divisibility` with three distinct inputs: the actual unnormalized series, its all-residue flatness theorem, and its support theorem. The interface does not permit an arbitrary series to acquire this bound. The difficult recurrence/flatness propagation resides in its imported arithmetic modules, whose explicit declarations remain in the complete catalog. The short endpoint proof is a composition of those established inputs, not an independent numerical experiment.

The normalization transfer must retain its factorial loss. A bound on $A_{N,j}$ gives a bound on the normalized coefficient only after subtracting the valuation of the normalization factor. The retained normalization counterexample shows why a universal loss-free transfer is false. A theorem with a stronger hypothesis in a standard window must not be silently extended to every row without that transfer.

4.2 A Newton expansion converts differences into a period

Contract. R is a commutative ring, p is prime, $r > 0$, and every order-k difference at the chosen base point is divisible by $p^{\min(k,r)}$. The conclusion is one period displacement at that base point.

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/PeriodFromDifferences.lean`.

Lines 46–67; `KomorebiPaperIX.PeriodFromDifferences.prime_power_period`.

```
theorem prime_power_period (f : ℕ → R) (h n r : ℕ) (hr : 0<r)
    (H : ∀ k, (p:R)^(min k r) | difference h k f n) :
    (p:R)^r | f (n+p^(r-1)*h)-f n := by
  rw [displacement_expansion]
  apply Finset.dvd_sum
  intro k hk
  have hkr : k+1 ≤ p^(r-1) := by have := Finset.mem_range.mp hk; omega
  by_cases hsmall : k+1 < r
  · have hd := H (k+1)
    rw [min_eq_left (by omega : k+1≤r)] at hd
    obtain ⟨a,ha⟩ := hd
    rw [ha, ←mul_assoc]
    have hb := binomial_precision p (r-1) (k+1) (by omega) hkr
    rw [Nat.sub_add_cancel (by omega : 1≤r)] at hb
    have hcast : (p:R)^r | (((p^(r-1)).choose (k+1):ℕ):R)*(p:R)^(k+1) := by
      obtain ⟨b,hb⟩ := hb
      refine ⟨(b:R), ?_⟩
      simpa only [Nat.cast_mul, Nat.cast_pow] using congrArg (fun a:ℕ => (a:R)) hb
    exact dvd_mul_of_dvd_left hcast a
  · have hd := H (k+1)
    rw [min_eq_right (by omega : r≤k+1)] at hd
    exact dvd_mul_of_dvd_right hd _
```

Proof reading. The displacement expansion writes the change at p^{r-1} steps as a finite sum of binomial coefficients times higher differences. For an order $k < r$, the difference contributes p^k and the binomial coefficient contributes the missing digits. For $k \geq r$, the difference itself supplies the whole p^r. The proof explicitly splits these cases. It does not require division in R; integer divisibility of binomial coefficients is cast into the ring.

When the packet bound supplies $\min(k, mn)$, a sufficient threshold $r \leq mn$ gives the hypothesis above. The endpoint uses the simpler threshold $n \geq r$ valid across the allowed parameter families. This is a sufficient starting index. The proof makes no assertion that the threshold is the smallest possible one.

4.3 Packet address and degree address are different indices

The generalized Euler sequence is degree-indexed, while a residue packet sequence is indexed by its packet row n. The source proves its reciprocal generating identity before presenting the external conjecture: the unnormalized series multiplied by the original exponential section is X^j. Thus the degree sequence is connected to the defining source series, not merely assigned a suggestive name.

Set $q = \operatorname{lcm}(2, p-1)$. The allowed divisor branch has packet shift $p^{r-1}(q/m)$, which becomes degree shift qp^r after multiplication by mp. The exceptional allowed family is $p = 2, m = 2$. For odd primes, the option $m = 2$ already belongs to the divisor branch because $2 \mid p-1$.

Contract. The integral quotient q/m is justified by $m \mid p-1$ or $m = 2$. The degree identity also requires $r > 0$ so that $p^r = p^{r-1}p$.

Source: baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/NishibuchiCompletion.lean.

Lines 89–96; KomorebiPaperIX.NishibuchiCompletion.original_degree_shift.

```
theorem original_degree_shift (m n r:ℕ) (h:m|p-1 ∨ m=2) (hr:0<r) :
    m*p*(n+p^(r-1)*(displacementFactor p/m)) =
      m*p*n+displacementFactor p*p^r := by
  have hq := Nat.mul_div_cancel' (modulus_divides_displacement p m h)
  have hp : p^r=p^(r-1)*p := by rw [←pow_succ,Nat.sub_add_cancel (by omega : 1≤r)]
  calc
    _ = m*p*n+(m*(displacementFactor p/m))*(p^(r-1)*p) := by ring
    _ = _ := by rw [hq,hp]
```

Proof reading. The proof establishes $m(q/m) = q$ and rewrites the exponent successor using $r \geq 1$. The remaining identity is ring arithmetic. Natural-number division is not an approximate operation here: its exact cancellation is justified by divisibility. Forgetting that hypothesis would change the intended displacement.

4.4 The completion theorem exposes its full range

Contract. p is prime, $m > 0$, $j < mp$, and $m \mid p-1$ or $m = 2$. Every degree-mpn coefficient is p-integral. For each $r > 0$, a positive starting index $n_0 \leq r$ exists, and all $n \geq n_0$ satisfy the stated degree displacement congruence.

Source: baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/NishibuchiCompletion.lean.

Lines 128–141; KomorebiPaperIX.NishibuchiCompletion.conjecture_3_9.

```
theorem conjecture_3_9 (m j:ℕ) (hmpos:0<m) (hj:j<m*p) (h:m|p-1 ∨ m=2) :
    (∀n, ((generalizedEuler (m*p) (Nat.mul_ne_zero hmpos.ne' (Fact.out:p.Prime).ne_zero)
      j (m*p*n):ℚ):ℚ_[p]) ∈ PadicInt.subring p) ∧
    ∀r:ℕ, 0<r → ∃n0:ℕ, 0<n0 ∧ n0≤r ∧ ∀n:ℕ, n0≤n →
      precision p (r:ℤ)
        (generalizedEuler (m*p) (Nat.mul_ne_zero hmpos.ne' (Fact.out:p.Prime).ne_zero) j
          (m*p*n+displacementFactor p*p^r)-
         generalizedEuler (m*p) (Nat.mul_ne_zero hmpos.ne' (Fact.out:p.Prime).ne_zero) j (m*p*n)) :=
          ↪ by
  constructor
  · intro n
    rw [generalizedEuler_at_packet]
    exact full_integrality p m j n hmpos hj h
  · intro r hr
    exact ⟨r,hr,le_rfl,fun n hn => full_period_degree p m j r n hmpos hj h hr hn⟩
```

Proof reading. The conjunction separates integrality from eventual periodicity. Integrality first rewrites the degree coefficient at mpn into the packet coefficient, then invokes the proved integral lift. The period witness is exactly $n_0 = r$. The degree-period theorem is supplied for every later n, so the existential

threshold is constructive at the statement level even though the ambient development uses classical mathematics.

This endpoint proves the original range recorded in the retained external comparison. The prime, parameter, address, and displacement conventions all remain visible. It supplies an eventual period and integrality, while the later dyadic modules prove sharper exact periods in their own specified family. It does not extend the conjecture to arbitrary m, identify an optimal starting index, or establish a least period in every prime family.

5 Reconstruction, sharp dyadic displacement, and isometry

5.1 Odd reconstruction separates identity from precision

Contract. The odd Euler reconstruction is an exact identity over $\mathbb{Z}$ for every odd length p, even when that length is composite. The prime-only corollary is a specialization.

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/OddEulerReconstruction.lean`.

Lines 80–87; `KomorebiPaperIX.OddEuler.odd_euler_reconstruction`.

```
theorem odd_euler_reconstruction (p n : ℕ) (hp : Odd p) :
    eulerEven n = ∑ m ∈ Finset.range (n+1),
      ((2*n).choose (2*m) : ℤ) * inverseEven m * (p:ℤ)^(2*m) * source p (2*(n-m)) := by
  have h := reconstruction_all_indices p (2*n) hp (even_two_mul n)
  rw [euler_even] at h
  rw [sum_even_terms (fun r => ((2*n).choose r : ℤ)*inverseCoeff r*(p:ℤ)^r*source p (2*n-r))
    (fun m => by rw [inverse_odd_zero _ (by exact ⟨m,rfl⟩)]; simp)] at h
  simpa only [inverseEven, Nat.mul_sub_left_distrib] using h
```

The series identity underlying this theorem multiplies a rescaled squared reciprocal series by the even part of the source series. It then extracts exponential coordinates, proves odd terms vanish, and changes from arbitrary indices to even indices. The coefficient $\binom{2n}{2m}$ and the factor p^{2m} come from that exact coordinate calculation. The source separately proves the signed zigzag identity for the inverse coefficient; the sign is not inferred from a plotted prefix.

Contract. Truncation includes $m = 0, \dots, \min(M, n)$. Every omitted summand has the stated even weight.

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/OddEulerReconstruction.lean`.

Lines 144–150; `KomorebiPaperIX.OddEuler.odd_euler_supercongruence`.

```
theorem odd_euler_supercongruence (p n M : ℕ) (hp : Odd p) :
    (p:ℤ)^(2*M+2) | eulerEven n - truncation p n M := by
  rw [odd_euler_reconstruction p n hp]
  have h := Filtered.even_weight_tail (p:ℤ)
    (fun m => ((2*n).choose (2*m):ℤ)*inverseEven m*source p (2*(n-m))) n M
  rw [filter_range_le] at h
  simpa only [truncation, mul_assoc, mul_left_comm, mul_comm] using h
```

Proof reading. The full sum is split into retained and omitted indices. For an omitted m, $m \geq M+1$, so its factor p^{2m} contains p^{2M+2}. The tail theorem is therefore an integer divisibility statement. If $M \geq n$, there are no omitted terms and the truncation equals the full value exactly. No convergence question arises, because each row has only $n+1$ terms.

5.2 The dyadic transform has its own coefficients

Contract. The dyadic Euler reconstruction uses its specifically constructed dyadic source and integral transform coefficients. It is not obtained by substituting $p = 2$ into a theorem whose hypothesis is `Odd p`.

Source: baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/DyadicReconstruction.lean.

Lines 26–45; KomorebiPaperIX.DyadicReconstruction.dyadic_reconstruction.

```
theorem dyadic_reconstruction (n : ℕ) :
    (eulerEven n : ℚ) = ∑ m ∈ Finset.range (n+1), summand n m := by
  have hs := congrArg (fun f => EGF.decode f (2*n)) dyadic_reconstruction_series
  have hp := EGF.egf_product (EGF.decode (rescale (2:ℚ) inverseSeries))
    (EGF.decode DyadicSeries.sourceSeries) (2*n)
  simp only [EGF.encode_decode] at hp
  have he : EGF.decode OddPower.sech (2*n) = (eulerEven n : ℚ) := by
    rw [← euler_egf, EGF.decode_encode, euler_even]
  rw [hp,he] at hs
  rw [OddEuler.sum_even_terms _ (fun m => by
    simp [EGF.decode, coeff_rescale, inverse_odd_coefficient])] at hs
  rw [← hs]
  apply Finset.sum_congr rfl
  intro m hm
  simp only [EGF.decode, coeff_rescale, inverse_even_coefficient,
    ← Nat.mul_sub_left_distrib, source_even_coefficient]
  unfold summand
  rw [Nat.descFactorial_eq_factorial_mul_choose]
  push_cast
  field_simp
```

The dyadic coefficient analysis includes the valuation of the transform coefficients and of the source terms. Its sharper truncation bound is

$$2^{4M+3} \mid E_{2n} - T_M(n)$$

in the p-adic integral coordinates. That exponent comes from the omitted index bound and the valuation of the dyadic coefficient κ_m. The first omitted term is separately analyzed to show sharpness where the stated row/depth hypotheses apply.

Contract. The displayed uniform sharp tail bound holds for all natural n, M. Full-depth truncations have zero error. A nonzero exact valuation for the first omitted error has its own boundary hypotheses.

Source: strengthening/formalization/DyadicSharpTower.lean.

Lines 176–189; KomorebiPaperIX.DyadicSharpTower.dyadic_supercongruence_sharp.

```
theorem dyadic_supercongruence_sharp (n M : ℕ) :
    (2:ℤ_[2])^(4*M+3) | (EulerCoefficients.eulerEven n:ℤ_[2]) - truncationPadic n M := by
  rw [dyadic_reconstruction_padic, truncationPadic, ← OddEuler.filter_range_le]
  classical
  rw [← Finset.sum_filter_add_sum_filter_not (Finset.range (n+1)) (fun m => m ≤ M)
    (summandPadic n), add_sub_cancel_left]
  apply Finset.dvd_sum
  intro m hm
  obtain ⟨_, hm⟩ := Finset.mem_filter.mp hm
  have hpos : 0 < m := by omega
```

```
  have hpow : (2:ℤ_[2])^(4*M+3) | (2:ℤ_[2])^(4*m-1) := pow_dvd_pow 2 (by omega)
  rw [summandPadic_kappa]
  exact dvd_mul_of_dvd_left (dvd_mul_of_dvd_left
    (hpow.trans (kappaPadic_divisible m hpos)) _) _
```

Proof reading. The proof splits the finite reconstruction sum by the predicate $m \le M$. In the complement, $m > 0$, allowing the coefficient divisibility theorem. The exponent comparison $4M + 3 \le 4m - 1$ supplies the uniform bound. The calculation is a universal finite-sum proof; checking numerical rows is independent bounded evidence about examples, not the reason this theorem holds.

5.3 Dominance gives an exact valuation

A lower bound on every difference does not itself determine a least period. The sharp modules isolate a nonzero leading Newton contribution whose valuation is smaller than all remaining contributions. In a nonarchimedean field, that strict separation forces the whole sum to have the leading term's valuation. A tie in the lowest valuations could allow cancellation, so the strict tail estimate is essential.

Contract. For each of the four normalized dyadic addresses $j < 4$ and each positive natural displacement T, the difference is nonzero and its valuation is $v_2(T) + b_j$, where b_j is the explicitly defined prefix bonus.

Source: `strengthening/formalization/DyadicFourLeastPeriods.lean`.

Lines 26–32; `KomorebiPaperIX.DyadicFourLeastPeriods.normalized_exact_displacement`.

```
theorem normalized_exact_displacement (j n T : ℕ) (hj : j < 4) (hT : 0 < T) :
    coeff 4 j (n + T) - coeff 4 j n ≠ 0 ∧
    padicValRat 2 (coeff 4 j (n + T) - coeff 4 j n) =
      (padicValNat 2 T : ℤ) + (DyadicFourPrefix.bonus j : ℤ) := by
  exact NewtonDominance.exact_displacement (coeff 4 j) (DyadicFourPrefix.bonus j)
    (DyadicFourPrefix.first_nonzero j hj) (DyadicFourPrefix.first_valuation j hj)
    (DyadicFourPrefix.tail_precision j hj) n T hT
```

Proof reading. The endpoint supplies four inputs to the generic Newton dominance theorem: first nonvanishing coefficient, its exact valuation, the strict tail-precision bound, and positivity of T. The conjunction records nonzero difference as well as valuation equality. At $T = 0$ the difference is zero, so the exact nonzero statement would be false; the theorem excludes that case.

The normalized bonus and the unnormalized bonus are distinguished. The latter incorporates the relevant factorial normalization. The exact-period classification is stated using the actual unnormalized sequence and its own bonus c_j.

Contract. A period in the source is a positive displacement. At precision r, the four-address unnormalized packet has period T exactly when $T > 0$ and $2^{\max(r-c_j,0)} \mid T$. Natural subtraction in the source expresses the maximum with zero.

Source: `strengthening/formalization/DyadicFourLeastPeriods.lean`.

Lines 58–62; `KomorebiPaperIX.DyadicFourLeastPeriods.period_iff_dvd`.

```
theorem period_iff_dvd (j r T : ℕ) (hj : j < 4) :
    ExactPeriodClassification.IsPeriod 2 (unnormalized 4 j) r T ↔
```

```
      0 < T ∧ 2 ^ (r - originalBonus j) | T :=
  ExactPeriodClassification.isPeriod_iff_dvd 2 (unnormalized 4 j)
    (originalBonus j) r T (fun n S hS => exact_displacement j n S hj hS)
```

Proof reading. Exact displacement reduces a universal congruence at every base row to an inequality $v_2(T)+c_j \geq r$. The generic classification translates that inequality into divisibility. If $r \leq c_j$, the exponent is zero and every positive displacement is a period. The least positive one is then one. If $r > c_j$, the least period is the indicated power of two. Zero remains excluded by the period predicate.

5.4 The normalized packet extends to a p-adic isometry

Contract. For $j < 4$, the appropriately rescaled rational packet extends uniquely to an isometric equivalence $\mathbb{Z}_2 \simeq \mathbb{Z}_2$ agreeing with its values at natural integers.

Source: `strengthening/formalization/DyadicFourIsometry.lean`.

Lines 78–84; `KomorebiPaperIX.DyadicFourIsometry.existsUnique_isometry`.

```
theorem existsUnique_isometry (j : ℕ) (hj : j < 4) :
    ∃! e : ℤ_[2] ≃ᵢ ℤ_[2], ∀ n : ℕ,
      (e (n : ℤ_[2]) : ℚ_[2]) = (DyadicPacketRational.packet j n : ℚ_[2]) := by
  refine ⟨isometry j hj, isometry_nat j hj, ?_⟩
  intro e he
  apply IsometryEquiv.ext
  exact congrFun (unique_continuous j hj e e.continuous he)
```

Proof reading. The source first constructs the isometry and proves agreement on natural inputs. Uniqueness then applies the theorem that continuous extensions agreeing on the dense natural subset agree everywhere. The isometric-equivalence conclusion includes surjectivity, so a merely distance-preserving embedding would not suffice. The rescaling and basepoint translation are part of the packet definition; the raw unnormalized sequence has a different image ball.

The concrete normalization is

$$(b_0, b_1, b_2, b_3) = (1, 1, 4, 2), \qquad (c_0, c_1, c_2, c_3) = (1, 1, 5, 3),$$

where b_j are normalized-coefficient bonuses and c_j are original-coefficient bonuses. Writing $U_{4,j}$ for the unnormalized packet, the isometric packet is

$$P_j(n) = \frac{U_{4,j}(n) - j!}{2^{c_j}}.$$

Its extension maps the unit ball onto the unit ball. Undoing that formula shows the original unnormalized extension has image $j! + 2^{c_j}\mathbb{Z}_2$.

This distinction explains the manuscript's affine-image consequence: undoing the normalization sends the unit ball to a specified affine residue ball. Neither the period theorem nor the isometry endpoint proves an unrestricted arithmetic statement for all moduli. Their exact hypotheses identify the four-address dyadic family being closed.

6 Finite obstructions and a common correction

6.1 Residuals become quotient classes

Let $A : \mathbb{Z}^d \to \mathbb{Z}^e$ be linear and suppose $Ax - b = p^L r$. A same-residue correction has the form $y = x + p^L t$. Its next residual is $p^L(r + At)$. Thus the next level exists precisely when $\bar{r} + \bar{A}\bar{t} = 0$ modulo p. The obstruction is the class of $\bar{r}$ in the cokernel of the digit map. Zero class means the residual can be cancelled by some digit.

Contract. The affine theorem requires $p > 0$, but neither primality nor $L \geq 1$. It supplies an integer residual, a finite obstruction space, and an equivalence between same-residue next-level lifts and zero obstruction.

Source: `polynomial_regularity/formalization/AffineObstructionCore.lean`.

Lines 152–171; `Komorebi.NativeAffine.finite_affine_obstruction`.

```
theorem finite_affine_obstruction (p L : ℕ) (hp : 0 < p)
    (A : (Fin d → ℤ) →ₗ[ℤ] (Fin e → ℤ)) (b : Fin e → ℤ) (x : Fin d → ℤ)
    (hx : ∀ i, (p : ℤ)^L | (A x - b) i) :
    ∃ r : Fin e → ℤ,
      A x - b = (p : ℤ)^L • r ∧
      Finite (Obstruction p A) ∧
      ((∃ y : Fin d → ℤ,
        (∀ i, (p : ℤ)^L | (y - x) i) ∧
        (∀ i, (p : ℤ)^(L+1) | (A y - b) i)) ↔ obstruction p A r = 0) := by
  obtain ⟨r, hr⟩ := residual_exists p L A b x hx
  let : NeZero p := ⟨Nat.ne_of_gt hp⟩
  refine ⟨r, hr, obstruction_finite p A, ?_⟩
  rw [← affine_lift_iff_obstruction_zero p L hp A b r x hr]
  constructor
  · rintro ⟨y, hy, hAy⟩
    obtain ⟨t, rfl⟩ := (same_residue_iff ((p : ℤ)^L) x y).mp hy
    exact ⟨t, hAy⟩
  · rintro ⟨t, ht⟩
    exact ⟨x + (p : ℤ)^L • t,
      (same_residue_iff ((p : ℤ)^L) x _).mpr ⟨t, rfl⟩, ht⟩
```

Proof reading. Residual existence is coordinatewise integer divisibility. The source installs `NeZero p` to make the residue ring finite. It then rewrites arbitrary same-residue y as $x + p^L t$ using `same_residue_iff`. That rewriting supplies both directions: a successful lift gives a digit, and a successful digit constructs a lift. At $L = 0$, $p^L = 1$ and all integer points are in the same initial residue class; the affine calculation still works.

The finite quotient should not be confused with a rank formula. Finiteness over `ZMod p` works for any nonzero modulus. A dimension argument and the count $p^{d-\rho}$ need p prime so that the residue ring is a field.

Contract. Adding an image vector As to the residual does not change its obstruction class. No rank or surjectivity assumption is required.

Source: `polynomial_regularity/formalization/AffineObstructionCore.lean`.

Lines 70–79; `Komorebi.NativeAffine.obstruction_representative_independent`.

```
theorem obstruction_representative_independent (p : ℕ)
    (A : (Fin d → ℤ) →ₗ[ℤ] (Fin e → ℤ)) (r : Fin e → ℤ) (s : Fin d → ℤ) :
    obstruction p A (r + A s) = obstruction p A r := by
  change Submodule.Quotient.mk (reduction p e (r + A s)) = _
  rw [map_add, Submodule.Quotient.mk_add]
  have hs : (Submodule.Quotient.mk (reduction p e (A s)) : Obstruction p A) = 0 := by
    rw [Submodule.Quotient.mk_eq_zero]
    exact ⟨s, rfl⟩
  rw [hs, add_zero]
  rfl
```

Proof reading. The reduction map preserves addition. The quotient map then sends the reduced image vector to zero because it belongs to the digit-map range. Adding zero leaves the class unchanged. This is an exact statement about replacing a representative by a known image term. It does not establish the separate open problem of presentation invariance under every possible native equivalence.

6.2 Two solvable equations need one shared digit

For two maps A and B with separately soluble digit equations, choose individual correcting digits u and v. The first allows the coset $u + \ker \bar{A}$ and the second allows $v + \ker \bar{B}$. A common digit exists exactly when these cosets meet, equivalently when

$$u - v \in \ker \bar{A} + \ker \bar{B}.$$

The shared obstruction is the quotient of the common digit space by that sum. It is generally not the product of the two separate cokernels: those cokernels already vanish for the individually soluble pair, while the common-correction obstruction can still be nonzero.

Contract. Both chosen correcting digits must solve their own reduced residual equations. The theorem then identifies a joint next lift with equality of their classes in the shared quotient. Positivity of the modulus is retained; $L = 0$ is allowed.

Source: `polynomial_lifting/formalization/AffineSharedCorrection.lean`.

Lines 60–75; `Komorebi.AffinePair.joint_lift_iff_class_eq`.

```
theorem joint_lift_iff_class_eq (p L : ℕ) (hp : 0 < p)
    (A : (Fin d → ℤ) →ₗ[ℤ] (Fin e → ℤ))
    (B : (Fin d → ℤ) →ₗ[ℤ] (Fin f → ℤ))
    (b r : Fin e → ℤ) (c s : Fin f → ℤ) (x u v : Fin d → ℤ)
    (hr : A x - b = (p : ℤ)^L • r)
    (hs : B x - c = (p : ℤ)^L • s)
    (hu : reduction p e (r + A u) = 0)
    (hv : reduction p f (s + B v) = 0) :
    HasJointLift p L A B b c x ↔
    (Submodule.Quotient.mk u : SharedObstruction p A B) =
      Submodule.Quotient.mk v := by
  rw [joint_lift_iff_digit_correction p L hp A B b r c s x hr hs,
```

```
    quotient_mk_eq_iff_common]
  exact exists_congr fun t => and_congr
    (digit_correction_iff_same_image p A r u t hu)
    (digit_correction_iff_same_image p B s v t hv)
```

Proof reading. The first rewrite translates joint lifting into a simultaneous digit equation. The second identifies class equality with a common representative. The final `exists_congr` compares the two individual conditions by their image fibers. This separates the existence of separate corrections from their compatibility. A claim that separate solvability automatically gives a joint lift would drop the equality of quotient classes and is false.

Changing u by a digit in $\ker \bar{A}$ or changing v by one in $\ker \bar{B}$ leaves the shared class unchanged. The quotient has the expected universal property for maps annihilating both kernels, and its finite image/range formulations are proved in separate modules. Those endpoint statements retain their own exact coefficient conventions.

6.3 A nonempty fiber is a kernel torsor

Contract. $A : U \to V$ is linear over a ring, and a base point u has been supplied in the fiber over b. The fiber is equivalent to $\ker A$ by subtraction from u and translation back. The ambient U, V are modules and additive commutative groups.

Source: `polynomial_regularity/formalization/FiniteFiberCounts.lean`.

Lines 75–79; `Komorebi.FiniteFiberCounts.fiberEquivKernel`.

```
def fiberEquivKernel (A : U →ₗ[K] V) (b : V) (u : Fiber A b) : Fiber A b ≃ A.ker where
  toFun t := difference A b t u
  invFun v := translate A b v u
  left_inv t := translate_difference A b t u
  right_inv v := difference_translate A b v u
```

Proof reading. The forward map sends t to $t - u$ and uses linearity to prove it lies in the kernel. The inverse sends a kernel vector v to $u+v$ and proves its image is b. The two inverse identities are cancellation in the additive group. The chosen u is an origin for a torsor, not a distinguished zero of the original fiber. An empty fiber admits no such choice.

Contract. For a linear digit map $A : (\mathbb{F}_p)^d \to (\mathbb{F}_p)^e$ with rank ρ, the fiber over b has $p^{d-\rho}$ elements when nonempty and zero elements otherwise.

Source: `polynomial_regularity/formalization/FiniteFiberCounts.lean`.

Lines 138–144; `Komorebi.FiniteFiberCounts.fiber_card`.

```
theorem fiber_card (A : (Fin d → ZMod p) →ₗ[ZMod p] (Fin e → ZMod p))
    (b : Fin e → ZMod p) :
    Nat.card (Fiber A b) = if Nonempty (Fiber A b) then p ^ (d - rank p A) else 0 := by
  classical
  split_ifs with h
  · exact fiber_card_of_nonempty p A b h
  · exact fiber_card_of_empty A b h
```

Proof reading. The nonempty branch transports cardinality through the fiber–kernel equivalence and uses rank-nullity plus the finite-field cardinality theorem. The empty branch has cardinality zero. The `if Nonempty` guard cannot be discarded for a general right-hand side. In a surjective case every right-hand side enters the nonempty branch; in an injective square case the nonempty fiber has exactly one element.

7 Actual polynomial Taylor expansion and lift counts

7.1 A quadratic remainder is proved algebraically

Contract. P is an actual multivariate polynomial with integer coefficients, $x, t \in \mathbb{Z}^d$, and $a \in \mathbb{Z}$. The directional derivative uses the evaluated formal partial derivatives. Then a^2 divides $P(x + at) - P(x) - aDP(x)t$.

Source: `polynomial_regularity/formalization/PolynomialTaylor.lean`.

Lines 41–53; `Komorebi.PolynomialTaylor.taylor_dvd`.

```
theorem taylor_dvd {d : ℕ} (P : MvPolynomial (Fin d) ℤ)
    (x t : Fin d → ℤ) (a : ℤ) :
    a ^ 2 | eval (fun i => x i + a * t i) P - eval x P - a * directional P x t := by
  induction P using MvPolynomial.induction_on with
  | C c => simp [directional]
  | add P Q hP hQ =>
    simp only [map_add, directional_add]
    convert dvd_add hP hQ using 2 <;> first | rfl | ring
  | mul_X P j hP =>
    obtain ⟨r, hr⟩ := hP
    refine ⟨r * (x j + a * t j) + directional P x t * t j, ?_⟩
    simp only [map_mul, eval_X, directional_mul, directional_X]
    linear_combination (x j + a * t j) * hr
```

Proof reading. The proof inducts on the multivariate polynomial syntax supplied by Mathlib: a constant, an addition, or multiplication by a variable. Constants have zero remainder; sums add remainders. In the multiplication case, the induction witness r is combined with the linear term and the new variable displacement. `linear_combination` verifies the resulting polynomial identity. This is exact integer algebra and requires no differentiability or limiting argument.

The quadratic factor is what permits one-digit lifting. Put $a = q^L$. The remainder is divisible by q^{2L}. To neglect it modulo q^{L+1}, one needs $2L \geq L + 1$, hence $L \geq 1$. An affine map has no quadratic remainder and works at $L = 0$; a general polynomial does not. The source does not assume that an arbitrary nonlinear map comes with the same exact expansion.

7.2 One nonlinear step becomes a finite digit equation

Write $F(x) = q^L r$ coordinatewise. The actual Jacobian $J_F(x)$ is defined by formal partial derivatives evaluated at x, assembled as an integer linear map. The next-level equation is

$$F(x + q^L t) \equiv 0 \pmod{q^{L+1}} \quad \Longleftrightarrow \quad r + J_F(x)t \equiv 0 \pmod{q}.$$

Contract. $q > 0$, $L \geq 1$, and the given residual identity is exact. No prime assumption is needed for this equivalence.

Source: `polynomial_regularity/formalization/PolynomialObstruction.lean`.

Lines 97–108; `Komorebi.PolynomialObstruction.next_level_iff_digit`.

```
theorem next_level_iff_digit (q L : ℕ) (hq : 0 < q) (hL : 1 ≤ L)
    (F : Fin e → MvPolynomial (Fin d) ℤ) (x : Fin d → ℤ)
    (r : Fin e → ℤ) (hr : evalSystem F x = (q : ℤ)^L • r)
    (t : Fin d → ℤ) :
    (∀ i, (q : ℤ)^(L+1) | evalSystem F (x + (q : ℤ)^L • t) i) ↔
      reduction q e (r + jacobian F x t) = 0 := by
  rw [reduction_zero_iff]
  apply forall_congr'
  intro i
  rw [dvd_iff_of_dvd_sub _ _ _ (taylor_residual_dvd q L hL F x r hr t i)]
  have hqL : (q : ℤ)^L ≠ 0 := pow_ne_zero _ (Int.natCast_ne_zero.mpr (Nat.ne_of_gt hq))
  rw [pow_succ, mul_dvd_mul_iff_left hqL]
```

Proof reading. The source rewrites reduced zero as coordinatewise divisibility. Taylor remainder divisibility replaces the actual polynomial value by its residual-plus-linear term. The remaining factor q^L can be cancelled because $q > 0$ implies $q^L \neq 0$ in $\mathbb{Z}$. This cancellation gives the modulo-q equation. Both directions are proved, so the linearization is an exact next-digit criterion rather than an approximation.

Contract. The finite polynomial obstruction theorem starts with an approximate root modulo q^L and constructs its residual. The obstruction space uses the range of the actual Jacobian digit map.

Source: `polynomial_regularity/formalization/PolynomialObstruction.lean`.

Lines 141–154; `Komorebi.PolynomialObstruction.finite_polynomial_obstruction`.

```
theorem finite_polynomial_obstruction (q L : ℕ) (hq : 0 < q) (hL : 1 ≤ L)
    (F : Fin e → MvPolynomial (Fin d) ℤ) (x : Fin d → ℤ)
    (hx : ∀ i, (q : ℤ)^L | evalSystem F x i) :
    ∃ r : Fin e → ℤ,
      evalSystem F x = (q : ℤ)^L • r ∧
      Finite (Obstruction q (jacobian F x)) ∧
      ((∃ y : Fin d → ℤ,
        (∀ i, (q : ℤ)^L | (y - x) i) ∧
        (∀ i, (q : ℤ)^(L+1) | evalSystem F y i)) ↔
          obstruction q (jacobian F x) r = 0) := by
  obtain ⟨r, hr⟩ := residual_exists q L F x hx
  let : NeZero q := ⟨Nat.ne_of_gt hq⟩
  exact ⟨r, hr, obstruction_finite q (jacobian F x),
    polynomial_same_residue_lift_iff q L hq hL F x r hr⟩
```

Proof reading. Residual existence supplies r. The finite quotient construction is reused from the affine layer, while the polynomial same-residue theorem supplies the nonlinear lifting equivalence. Reusing the quotient algebra does not require treating the polynomial as affine; its application is justified precisely at this one-digit Taylor interface.

A nonzero obstruction means no same-residue next lift exists. A zero obstruction proves existence for that step. It does not by itself prove persistence at all later depths: the Jacobian and the residual may change, and singular digit maps can develop higher obstructions. The smooth tower theorem adds surjectivity and proves it persists along the constructed tower.

7.3 Digits correspond to residue lifts, not all integer representatives

The lift-counting development distinguishes an integer lift from its residue class modulo p^{L+1}. Without that quotient, a soluble congruence has infinitely many integer representatives. `ActualLifts` is a finite type of residue lifts, and `DigitSolutions` is the finite fiber of the digit equation.

Contract. $q > 0$, $L \geq 1$, and $F(x) = q^L r$. The residue digit equivalence takes a digit solution modulo q to a residue lift modulo q^{L+1}.

Source: `polynomial_regularity/formalization/PolynomialLiftCounts.lean`.

Lines 43–48; `Komorebi.PolynomialLiftCounts.digitEquivLifts`.

```
def digitEquivLifts (q L : ℕ) (hq : 0 < q) (hL : 1 ≤ L)
    (F : Fin e → MvPolynomial (Fin d) ℤ) (x : Fin d → ℤ)
    (r : Fin e → ℤ) (hr : evalSystem F x = (q : ℤ)^L • r) :
    DigitSolutions q F x r ≃ ActualLifts q L F x :=
  Equiv.subtypeEquiv (digitEquivResidue q L hq x)
    (fun t => (digit_residue_solution_iff q L hq hL F x r hr t).symm)
```

Proof reading. The source first has an equivalence between all digit addresses and all residue addresses above the initial class. `Equiv.subtypeEquiv` then restricts it to the subsets satisfying the equations. The previous next-digit theorem supplies the required equivalence of predicates. Thus cardinalities are preserved through a genuinely bijective map; a mere surjection would not establish the exact count.

Contract. Now p is prime. If the digit Jacobian has rank ρ, the number of residue lifts is zero for nonzero obstruction and $p^{d-\rho}$ for zero obstruction. The obstruction space has $p^{e-\rho}$ elements.

Source: `polynomial_regularity/formalization/PolynomialLiftCounts.lean`.

Lines 104–115; `Komorebi.PolynomialLiftCounts.finite_polynomial_lift_count`.

```
theorem finite_polynomial_lift_count (p L : ℕ) [Fact (Nat.Prime p)] (hL : 1 ≤ L)
    (F : Fin e → MvPolynomial (Fin d) ℤ) (x : Fin d → ℤ)
    (hx : ∀ i, (p : ℤ)^L | evalSystem F x i) :
    ∃ r : Fin e → ℤ,
      evalSystem F x = (p : ℤ)^L • r ∧
      Nat.card (ActualLifts p L F x) =
        (if obstruction p (jacobian F x) r = 0 then
          p ^ (d - FiniteFiberCounts.rank p (digitJacobian p F x)) else 0) ∧
      Nat.card (Obstruction p (jacobian F x)) =
        p ^ (e - FiniteFiberCounts.rank p (digitJacobian p F x)) := by
  obtain ⟨r, hr⟩ := Komorebi.PolynomialObstruction.residual_exists p L F x hx
  exact ⟨r, hr, lift_card p L hL F x r hr, obstruction_card p F x⟩
```

Proof reading. The endpoint reuses the digit–lift equivalence for the lift cardinality and the finite cokernel equivalence for the obstruction cardinality. Rank-nullity yields the two exponents in the input and output spaces respectively. The rank is bounded by both d and e, so natural subtraction expresses the ordinary nonnegative dimension difference in these hypotheses. Rectangular equations can have many lifts even when every obstruction vanishes.

At a surjective digit Jacobian, $\rho = e$ and each step has p^{d-e} residue lifts. When $d = e$, that number is one. This is the source of both smooth existence and square-case uniqueness. When $d > e$, existence is still valid but a claim of unique solved coordinates would discard the nontrivial kernel.

8 Compatible integer towers and p-adic endpoints

8.1 The tower is constructed, not supplied

Contract. $q > 0$, $L \geq 1$, F has integer coefficients, the starting point is a root modulo q^L, and its digit Jacobian over `ZMod q` is surjective. The theorem constructs a whole sequence z_n of integer points.

Source: `polynomial_regularity/formalization/PolynomialIntegerTower.lean`.

Lines 80–96; `Komorebi.PolynomialIntegerTower.exists_compatible_tower_strong`.

```
theorem exists_compatible_tower_strong (q L : ℕ) (hq : 0 < q) (hL : 1 ≤ L)
    (F : Fin e → MvPolynomial (Fin d) ℤ) (x : Fin d → ℤ)
    (hx : ∀ i, (q : ℤ)^L | evalSystem F x i)
    (hJ : Function.Surjective (digitJacobian q F x)) :
    ∃ z : ℕ → Fin d → ℤ,
      z 0 = x ∧
      (∀ n i, (q : ℤ)^(L+n) | evalSystem F (z n) i) ∧
      (∀ n i, (q : ℤ)^(L+n) | (z (n+1) - z n) i) ∧
      (∀ m n, m ≤ n → ∀ i, (q : ℤ)^(L+m) | (z n - z m) i) ∧
      (∀ n, digitJacobian q F (z n) = digitJacobian q F x) ∧
      (∀ n, Function.Surjective (digitJacobian q F (z n))) := by
  obtain ⟨z, hz0, hroot, hstep, hsurj⟩ := exists_compatible_tower q L hq hL F x hx hJ
  refine ⟨z, hz0, hroot, hstep, ?_, ?_, hsurj⟩
  · exact fun m n hmn => tower_pairwise_congruent q L z hstep m n hmn
  · intro n
    rw [← hz0]
    exact tower_digitJacobian_eq q L hL F z hstep n
```

The theorem returns six pieces of information together. It fixes $z_0 = x$; $F(z_n)$ vanishes modulo q^{L+n}; successive points differ by multiples of q^{L+n}; every later point agrees with an earlier point to that earlier depth; the digit Jacobian stays equal to the starting one; and surjectivity holds at every level. These statements keep existence, compatibility and smoothness separate.

Proof reading. The basic tower theorem chooses a correcting digit at each step using digit-map surjectivity. The stronger endpoint adds pairwise congruence by summing compatible increments and identifies every digit Jacobian with the initial map. Polynomial partial derivatives have integer coefficients, so same residue modulo q gives the same reduced derivative. Since $L \geq 1$, all constructed points stay in that initial residue class.

The construction is allowed to use classical choice. It proves an actual infinite sequence in the mathematical model, but it is not presented as extracted executable code. A finite loop that finds digits for a few depths is useful bounded evidence; it does not replace the compatible-tower theorem.

8.2 Completeness changes the coefficient domain

For a prime p, cast z_n coordinatewise into $\mathbb{Z}_p$. Pairwise congruence at depths tending to infinity makes the sequence Cauchy in each coordinate. Finite products of complete spaces are complete, so the vector sequence converges to a p-adic point b.

Contract. p is prime, $L \geq 1$, and the same initial root and surjectivity hypotheses hold. The theorem returns the integer tower, its p-adic limit, an exact polynomial root, and the retained congruence of the limit to every tower stage.

Source: `polynomial_regularity/formalization/PolynomialPadicLift.lean`.

Lines 57–77; `Komorebi.PolynomialPadicLift.exists_padic_root_with_tower`.

```
theorem exists_padic_root_with_tower (p L : ℕ) [Fact (Nat.Prime p)] (hL : 1 ≤ L)
    (F : Fin e → MvPolynomial (Fin d) ℤ) (x : Fin d → ℤ)
    (hx : ∀ i, (p : ℤ)^L | evalSystem F x i)
    (hJ : Function.Surjective (digitJacobian p F x)) :
    ∃ z : ℕ → Fin d → ℤ, ∃ a : Fin d → ℤ_[p],
      z 0 = x ∧
      (∀ n i, (p : ℤ)^(L+n) | evalSystem F (z n) i) ∧
      (∀ n i, (p : ℤ)^(L+n) | (z (n+1) - z n) i) ∧
      Tendsto (fun n i => (z n i : ℤ_[p])) atTop (𝓝 a) ∧
      (∀ n i, (p : ℤ_[p])^(L+n) | a i - (z n i : ℤ_[p])) ∧
      (∀ i, padicEval p F a i = 0) ∧
      (∀ n, digitJacobian p F (z n) = digitJacobian p F x) ∧
      (∀ n, Nat.card (PolynomialLiftCounts.ActualLifts p (L+n) F (z n)) = p^(d-e)) := by
  obtain ⟨z, hz0, hroot, hstep, _, hcount⟩ := exists_regular_tower p L hL F x hx hJ
  obtain ⟨a, ha, hprecision⟩ :=
    PadicTowerLimit.compatible_vector_tower_limit p L z hstep
  refine ⟨z, a, hz0, hroot, hstep, ha, hprecision,
    polynomial_tower_limit_root p L F z a hroot ha, ?_, hcount⟩
  intro n
  rw [← hz0]
  exact tower_digitJacobian_eq p L hL F z hstep n
```

Proof reading. The source combines the integer tower theorem with the general p-adic tower-limit theorem. Polynomial evaluation is continuous, so $F(z_n)$ converges to $F(b)$. Its divisibility by arbitrarily large powers of p forces its limit to zero. A fixed-depth congruence ball is closed, yielding the stronger precision statement $p^{L+n} \mid b - z_n$ for each fixed n.

This proof separates two limiting operations. Compatibility controls the input limit; increasing residual depth controls the output limit. Neither alone suffices: a convergent sequence of approximate points whose residual depth does not increase need not converge to an exact root.

8.3 Uniqueness requires a square solved system

Contract. The p-adic unique-root endpoint uses d equations in d unknowns and a surjective digit Jacobian, hence a bijective digit map over $\mathbb{F}_p$. It asserts uniqueness only in the initial depth-L residue ball.

Source: `polynomial_regularity/formalization/PolynomialPadicUnique.lean`.

Lines 146–156; `Komorebi.PolynomialPadicUnique.exists_unique_padic_root`.

```
theorem exists_unique_padic_root (p L : ℕ) [Fact (Nat.Prime p)] (hL : 1 ≤ L)
    (F : Fin d → MvPolynomial (Fin d) ℤ) (x : Fin d → ℤ)
    (hx : ∀ i, (p : ℤ)^L | evalSystem F x i)
```

```
    (hJ : Function.Surjective (digitJacobian p F x)) :
    ∃! a : Fin d → ℤ_[p],
      (∀ i, MvPolynomial.eval₂ (Int.castRingHom ℤ_[p]) a (F i) = 0) ∧
      (∀ i, (p : ℤ_[p])^L | a i - (x i : ℤ_[p])) := by
  obtain ⟨a, ha, hax⟩ := exists_padic_root p L hL F x hx hJ
  refine ⟨a, ⟨ha, hax⟩, ?_⟩
  intro b hb
  exact padic_root_unique p L hL F x hJ b a hb.1 ha hb.2 hax
```

Proof reading. Existence supplies one root. The uniqueness theorem compares any second root with that root by finite residues. At each successive depth, the digit equation has exactly one solution, so equal residues at the current depth imply equal residues at the next. Equality at all depths forces equality in $\mathbb{Z}_p$. The excerpt packages that comparison into `∃!`.

The uniqueness proof uses actual finite reductions of arbitrary p-adic roots, not an assumption that every candidate arose from the chosen integer tower. That distinction matters: uniqueness must quantify over every root in the specified ball. For a rectangular surjective system, the kernel has positive dimension and the same conclusion is generally false. The retained rectangular-uniqueness negative control guards this boundary.

8.4 Coefficient stability retains the promised precision

The deformation modules also compare a polynomial system with a coefficientwise congruent perturbation in the smooth, unit-Jacobian setting. Because evaluation preserves coefficient congruence at integral points, the original approximate root is an approximate root of the perturbed system. Its reduced Jacobian is unchanged when the coefficient perturbation has positive depth. The same smooth root theorem gives a nearby root, and the uniqueness comparison fixes the promised residue precision.

This reasoning supplies a bound at the declared perturbation depth. It does not imply an additional free digit. A linear equation with its constant term shifted by exactly p^M has its root shifted by exactly that precision. The retained `InvalidPerturbationExtraPrecision` control is based on that obstruction to strengthening the bound.

9 Local charts and prescribed parameter paths

9.1 The solved block is the correct Jacobian

Split the variables as (y, t), where y has m solved coordinates and t has k prescribed parameters. For e equations the reduced Jacobian has blocks

$$DF = (J\ K), \qquad J : \mathbb{F}_p^m \to \mathbb{F}_p^e, \quad K : \mathbb{F}_p^k \to \mathbb{F}_p^e.$$

A prescribed parameter digit v changes the residual by Kv. The solved digit u must solve

$$Ju = -\bar{r} - Kv.$$

Surjectivity of the full matrix $(J\ K)$ is insufficient when v has already been fixed. It is surjectivity of J that permits every prescribed digit.

Contract. For an integer polynomial system, the source defines the parameter residual $\bar{r}+Kv$, the solved digit map J, and the joined full digit (u, v). The split equation equals the full digit equation exactly.

Source: `polynomial_regularity/formalization/PolynomialPrescribedSystem.lean`.

Lines 77–82; `Komorebi.PolynomialPrescribedSystem.prescribed_equation_iff`.

```
theorem prescribed_equation_iff (q : ℕ) (F : Fin e → MvPolynomial (Fin (m + k)) ℤ)
    (x : Fin (m + k) → ℤ) (r : Fin e → ℤ) (v : Fin k → ZMod q) (u : Fin m → ZMod q) :
    parameterResidual q F x r v + solvedJacobian q F x u = 0 ↔
      reduction q e r + digitJacobian q F x (PolynomialParameterSystem.join u v) = 0 := by
  rw [digitJacobian_join]
  simp only [parameterResidual, add_assoc, add_comm (parameterJacobian q F x v)]
```

Proof reading. The block derivative theorem rewrites the full digit map on `join u v` as the sum of the solved and parameter actions. Associativity and commutativity then identify the residual equations. The parameter term has positive sign inside $\bar{r} + Kv + Ju = 0$, and consequently negative sign when solving for u. Losing this sign changes the actual root motion.

If the solved block has rank ρ, a soluble prescribed-digit fiber has $p^{m-\rho}$ elements. For a surjective J, this becomes p^{m-e} for every prescribed v. A square solved block gives one solved digit for each parameter digit. Rectangular solved blocks leave free solved directions, so prescribing the parameter alone does not generally select one root.

9.2 The continuous chart includes both directions

Contract. The chart endpoint uses e equations, e solved coordinates and q free parameters. The starting integer point is a root modulo p^L, $L \geq 1$, and the solved digit Jacobian is surjective. The homeomorphism is between the parameter residue ball and the exact root locus in the full residue ball.

Source: `polynomial_regularity/formalization/PolynomialParameterChart.lean`.

Lines 167–183; `Komorebi.PolynomialParameterChart.parameterHomeomorph`.

```
def parameterHomeomorph (p L : ℕ) [Fact (Nat.Prime p)] (hL : 1 ≤ L)
    (F : Fin e → MvPolynomial (Fin (e + q)) ℤ) (x : Fin (e + q) → ℤ)
```

```
    (hx : ∀ i, (p : ℤ)^L | evalSystem F x i)
    (hJ : Function.Surjective (solvedJacobian p F x)) :
    residueBall p L (freePart x) ≃ₜ {z : residueBall p L x // padicEval p F z.val = 0} where
  toFun t := ⟨chartPoint p L hL F x hx hJ t, chartPoint_root p L hL F x hx hJ t⟩
  invFun z := ⟨freePart z.val.val, fun j => z.val.property (Fin.natAdd e j)⟩
  left_inv t := Subtype.ext (chartPoint_free p L hL F x hx hJ t)
  right_inv z := by
    apply Subtype.ext
    exact (chartPoint_unique p L hL F x hx hJ _ z.val z.property rfl).symm
  continuous_toFun := by
    apply Continuous.subtype_mk
    exact continuous_chartPoint p L hL F x hx hJ
  continuous_invFun := by
    apply Continuous.subtype_mk
    exact continuous_freePart.comp (continuous_subtype_val.comp continuous_subtype_val)
```

Proof reading. The forward map takes a parameter to its uniquely solved root in the ball. The inverse is the free-coordinate projection. The first inverse identity says the constructed root has the requested parameter. The second invokes uniqueness of solved coordinates for any root in the ball. Continuity of the solution map and continuity of the projection establish a homeomorphism, not merely a set bijection.

This chart has explicit integral residue balls and strong congruence control. Its source proof precedes the analytic regularity layer. A homeomorphism of root spaces does not automatically identify an analytic derivative; the later polynomial/analytic bridge supplies that stronger conclusion under fraction-field nonsingularity.

9.3 An arbitrary p-adic parameter is reached by integer digits

The prescribed tower construction can begin with a supplied compatible integer parameter path t_n converging to τ. It constructs solved coordinates at every depth and tracks the actual parameter path, rather than guessing compatible roots independently. The endpoint also builds such a parameter path for every p-adic τ in the initial parameter ball.

Contract. p is prime, $L \geq 1$, J is the surjective solved digit map with m solved coordinates and e equations, and τ lies in the initial parameter residue ball. No solution tower or parameter path is an input to this endpoint.

Source: `polynomial_regularity/formalization/PolynomialPrescribedPadicTower.lean`.

Lines 57–76; `Komorebi.PolynomialPrescribedPadicTower.exists_prescribed_padic_root_with_tower`.

```
theorem exists_prescribed_padic_root_with_tower (p L : ℕ) [Fact (Nat.Prime p)]
    (hL : 1 ≤ L) (F : Fin e → MvPolynomial (Fin (m+k)) ℤ)
    (x : Fin (m+k) → ℤ) (hx : ∀ i, (p : ℤ)^L | evalSystem F x i)
    (hJ : Function.Surjective (solvedJacobian p F x))
    (τ : Fin k → ℤ_[p])
    (hτ : ∀ j, (p : ℤ_[p])^L | τ j - ((freePart x j : ℤ) : ℤ_[p])) :
    ∃ (z : ℕ → Fin (m+k) → ℤ) (b : Fin (m+k) → ℤ_[p]),
      z 0 = x ∧
      (∀ n i, (p : ℤ)^(L+n) | evalSystem F (z n) i) ∧
      (∀ n i, (p : ℤ)^(L+n) | (z (n+1) - z n) i) ∧
      (∀ n, solvedJacobian p F (z n) = solvedJacobian p F x) ∧
```

```
      Tendsto (fun n i => (z n i : ℤ_[p])) atTop (𝓝 b) ∧
      (∀ i, padicEval p F b i = 0) ∧ freePart b = τ ∧
      (∀ n i, (p : ℤ_[p])^(L+n) | b i - (z n i : ℤ_[p])) ∧
      (∀ n v, Nat.card (PrescribedLifts p (L+n) F (z n) v) = p^(m-e)) := by
  obtain ⟨t, ht0, ht, _, htlim⟩ :=
    PolynomialParameterPath.exists_parameter_path p L (freePart x) τ hτ
  obtain ⟨z, b, hz0, _, hroot, hstep, hfixed, hlim, hb, hbτ, hprecision, hcard⟩ :=
    exists_root_with_prescribed_path p L hL F x hx hJ t ht0 ht τ htlim
  exact ⟨z, b, hz0, hroot, hstep, hfixed, hlim, hb, hbτ, hprecision, hcard⟩
```

Proof reading. The parameter-path theorem supplies integer approximations with the correct initial value and increment divisibility. The prescribed integer tower fills in solved digits using J-surjectivity. A p-adic tower limit then gives the exact root with free part τ. The returned finite lift count p^{m-e} holds at every stage and for every next parameter digit, making the rectangular freedom explicit.

Contract. For $m = e$, the prescribed root in the initial ball is unique. The full predicate includes its exact parameter τ as well as its root equation and residue ball.

Source: `polynomial_regularity/formalization/PolynomialPrescribedPadicTower.lean`.

Lines 96–109; `Komorebi.PolynomialPrescribedPadicTower.exists_unique_prescribed_padic_root`.

```
theorem exists_unique_prescribed_padic_root (p L : ℕ) [Fact (Nat.Prime p)]
    (hL : 1 ≤ L) (F : Fin e → MvPolynomial (Fin (e+k)) ℤ)
    (x : Fin (e+k) → ℤ) (hx : ∀ i, (p : ℤ)^L | evalSystem F x i)
    (hJ : Function.Surjective (solvedJacobian p F x))
    (τ : Fin k → ℤ_[p])
    (hτ : ∀ j, (p : ℤ_[p])^L | τ j - ((freePart x j : ℤ) : ℤ_[p])) :
    ∃! b : Fin (e+k) → ℤ_[p],
      (∀ i, padicEval p F b i = 0) ∧ freePart b = τ ∧
      (∀ i, (p : ℤ_[p])^L | b i - (x i : ℤ_[p])) := by
  obtain ⟨b, hb, hbτ, hbx⟩ := exists_prescribed_padic_root p L hL F x hx hJ τ hτ
  refine ⟨b, ⟨hb, hbτ, hbx⟩, ?_⟩
  intro c hc
  exact PolynomialPrescribedTowerUnique.prescribed_root_unique p L hL F x hx hJ
    c b hc.1 hb hc.2.2 hbx (hc.2.1.trans hbτ.symm)
```

Proof reading. Existence and the prescribed-root comparison theorem are combined. Two candidates with the same parameter have zero parameter difference, so their next solved digit difference lies in the kernel of the bijective solved block. It vanishes at each depth. The theorem therefore controls every candidate root with that parameter, independent of the tower choices used to obtain the first one.

If $k = 0$, the parameter space has one vector and the theorem reduces to the square root theorem. If there are zero equations, the algebraic finite-product statements remain meaningful in dimension zero, subject to their stated typeclass and surjectivity hypotheses. The source does not impose a positive dimension merely to make a matrix display look conventional.

9.4 A full-rank system admits a suitable coordinate split

For a general e-equation system with surjective full reduced Jacobian, the minor-selection module chooses e independent columns. The coordinate-reordering module turns that choice into a permutation separating solved and free coordinates; it proves evaluation and derivative blocks transform

according to that actual permutation. The general-chart theorem then applies the square solved-block chart in those coordinates and transports the result back.

This is existence of a good coordinate split. It does not assert that every externally prescribed split has a surjective solved block. If a parameter policy has already fixed particular coordinates, its prescribed-lifting theorem must still check that policy's own solved block. The finite dimensions and explicit reorder equivalence preserve the number of free parameters, $d - e$.

10 Weighted inverse bounds and quantitative root motion

10.1 A fraction-field bound has an integral certificate

Let J be the actual square Jacobian at an integral p-adic center. It may be singular modulo p while remaining invertible over $\mathbb{Q}_p$. The relevant size condition is that $p^s J^{-1}$ is an integral matrix. This condition records the chosen exponent s directly; the proof does not replace it by a possibly larger determinant valuation.

Contract. p is prime and s is natural. Fraction-field invertibility together with an integral matrix representing $p^s J^{-1}$ is equivalent to an integral two-sided certificate $AJ = JA = p^s I$.

Source: `polynomial_regularity/formalization/PolynomialWeightedInverse.lean`.

Lines 142–155; `Komorebi.PolynomialWeightedInverse.bounded_inverse_iff_certificate`.

```
theorem bounded_inverse_iff_certificate (p s : ℕ) [Fact (Nat.Prime p)]
    (J : Matrix (Fin d) (Fin d) ℤ_[p]) :
    (IsUnit (rationalMatrix p J) ∧
      ∃ A : Matrix (Fin d) (Fin d) ℤ_[p],
        rationalMatrix p A = (p : ℚ_[p])^s • (rationalMatrix p J)⁻¹) ↔
    (∃ A : Matrix (Fin d) (Fin d) ℤ_[p],
      A * J = (p : ℤ_[p])^s • (1 : Matrix (Fin d) (Fin d) ℤ_[p]) ∧
      J * A = (p : ℤ_[p])^s • (1 : Matrix (Fin d) (Fin d) ℤ_[p])) := by
  constructor
  · rintro ⟨hJ, hA⟩
    exact (integral_inverse_iff_certificate p s J hJ).mp hA
  · rintro ⟨A, hAJ, hJA⟩
    have hJ := rationalMatrix_isUnit_of_certificate p s J A hAJ
    exact ⟨hJ, (integral_inverse_iff_certificate p s J hJ).mpr ⟨A, hAJ, hJA⟩⟩
```

Proof reading. The embedding of integral matrices into rational p-adic matrices preserves multiplication and scalar action. An integral representation of the inverse therefore gives the two identities after injectivity of the embedding. Conversely, the two-sided identities establish invertibility in the fraction field because $p^s \neq 0$. Both products matter: they support normalization and root equivalence in different directions. A single entrywise valuation statement without an actual matrix inverse is not this certificate.

10.2 Polynomial maps become contractions after scaling

Integral polynomial evaluation is Lipschitz for the p-adic sup metric. Multiplying it by p yields a strict contraction. The contraction map has the form $u \mapsto c + pH(u)$, so the fixed point theorem acts on the complete space of integral p-adic vectors.

Contract. H is an arbitrary vector of actual multivariate polynomials with $\mathbb{Z}_p$ coefficients and $c \in \mathbb{Z}_p^d$. The source establishes a unique fixed point of its specified contraction map.

Source: `polynomial_regularity/formalization/PolynomialPadicContraction.lean`.

Lines 101–105; `Komorebi.PolynomialPadicContraction.exists_unique_fixedPoint`.

```
theorem exists_unique_fixedPoint (p : ℕ) [Fact p.Prime]
    (H : Fin d → MvPolynomial (Fin d) ℤ_[p]) (c : Fin d → ℤ_[p]) :
    ∃! u : Fin d → ℤ_[p], contractionMap p H c u = u := by
  let hf := contracting_contractionMap p H c
  exact ⟨hf.fixedPoint _, hf.fixedPoint_isFixedPt, fun u hu => hf.fixedPoint_unique hu⟩
```

Proof reading. The source first proves the Lipschitz estimate for polynomial evaluation and the contraction constant after multiplication by p. `contracting_contractionMap` packages that estimate into the complete-space fixed point machinery. The final three fields are the fixed point, its fixed-point equation, and uniqueness against every other fixed point. The source's separate divisible-fixed-point theorem controls precision when c has a specified p-power factor and $H(0) = 0$.

The condition $H(0) = 0$ is required for that additional divisible-fixed-point precision argument; it is not a requirement of the bare fixed-point existence theorem displayed above. Keeping these two contracts separate prevents an accidental strengthening.

10.3 The threshold and the uniqueness ball are explicit

For $F(a)$ divisible by p^r and a certificate exponent s, the source imposes $r > 2s$. Write $b = a + p^{s+1}u$. Actual polynomial Taylor expansion and preconditioning put the equation into the normalized form

$$p^{2s+1}(C + u + pQ(u)) = 0, \qquad Q(0) = 0, \qquad p^{r-(2s+1)} \mid C.$$

The contraction solves for u, and its precision transfers back to b.

Contract. F is square with integral p-adic coefficients; $J_F(a)$ is invertible over $\mathbb{Q}_p$; $p^s J_F(a)^{-1}$ is integral; $F(a)$ is divisible by p^r; and $r > 2s$. The root is unique in the entire depth-$s+1$ ball and is congruent to a to depth $r - s$.

Source: `polynomial_regularity/formalization/PolynomialWeightedLift.lean`.

Lines 112–125; `Komorebi.PolynomialWeightedLift.exists_weighted_root`.

```
theorem exists_weighted_root (p s r : ℕ) [Fact p.Prime]
    (hr : 2 * s < r) (F : Fin d → MvPolynomial (Fin d) ℤ_[p])
    (a : Fin d → ℤ_[p]) (hJ : IsUnit (rationalMatrix p (jacobian F a)))
    (hbound : ∃ A : Matrix (Fin d) (Fin d) ℤ_[p],
      rationalMatrix p A = (p : ℚ_[p]) ^ s • (rationalMatrix p (jacobian F a))⁻¹)
    (hF : ∀ i, (p : ℤ_[p]) ^ r ∣ eval a (F i)) :
    ∃ b : Fin d → ℤ_[p],
      (∀ i, eval b (F i) = 0) ∧
      (∀ i, (p : ℤ_[p]) ^ (r - s) ∣ b i - a i) ∧
      (∀ i, (p : ℤ_[p]) ^ (s + 1) ∣ b i - a i) ∧
      (∀ z : Fin d → ℤ_[p], (∀ i, eval z (F i) = 0) →
        (∀ i, (p : ℤ_[p]) ^ (s + 1) ∣ z i - a i) → z = b) := by
  obtain ⟨A, hAJ, hJA⟩ := (bounded_inverse_iff_certificate p s (jacobian F a)).mp ⟨hJ, hbound⟩
  exact exists_weighted_root_certificate p s r hr F a A hAJ hJA hF
```

Proof reading. The source converts the fraction-field bound to the integral two-sided certificate. Its normalization theorem builds C, Q from the actual polynomials, so no Taylor expansion or contraction hypothesis is supplied by the caller. The normalized-root theorem obtains the fixed point and transfers its root equation, precision, and uniqueness back to the original coordinates. The root-equivalence step uses injectivity of the preconditioner in the fraction field.

At the least permitted depth $r = 2s + 1$, the precision $r - s = s + 1$ is exactly the uniqueness-ball depth. Larger r improves the retained root precision. The boundary $r = 2s$ is excluded; the retained weighted-boundary counterexample supplies an integral polynomial with that residual depth and no integral root. A linear example also shows the loss of s in the displacement is sharp. Thus neither replacing $>$ by $\geq$ nor adding a free precision digit follows from this theorem.

10.4 Coefficient perturbation has a first-order formula

Suppose $F(a) = 0$ and perturb the actual coefficient system to $F + p^M H$, with $M > 2s$. The same original inverse certificate supports the nearby root. The derivative of the perturbation is handled in the proof, rather than assumed to have exactly the old inverse.

Contract. The perturbed root is unique in the depth-$s + 1$ ball, has displacement divisible by p^{M-s}, and its rational p-adic first-order term is $-p^M J_F(a)^{-1} H(a)$. The integral error has depth $2M - 3s$.

Source: `polynomial_regularity/formalization/PolynomialRootMotion.lean`.

Lines 142–162; `Komorebi.PolynomialRootMotion.exists_unique_root_motion`.

```
theorem exists_unique_root_motion (p s M : ℕ) [Fact p.Prime]
    (hM : 2 * s < M) (F H : Fin d → MvPolynomial (Fin d) ℤ_[p])
    (a : Fin d → ℤ_[p]) (hFa : ∀ i, eval a (F i) = 0)
    (hJ : IsUnit (rationalMatrix p (jacobian F a)))
    (hbound : ∃ A : Matrix (Fin d) (Fin d) ℤ_[p],
      rationalMatrix p A = (p : ℚ_[p]) ^ s • (rationalMatrix p (jacobian F a))⁻¹) :
    (∃! b : Fin d → ℤ_[p], (∀ i, eval b (perturbSystem p M F H i) = 0) ∧
      (∀ i, (p : ℤ_[p]) ^ (s + 1) | b i - a i)) ∧
    (∀ b : Fin d → ℤ_[p], (∀ i, eval b (perturbSystem p M F H i) = 0) →
      (∀ i, (p : ℤ_[p]) ^ (s + 1) | b i - a i) →
      (∀ i, (p : ℤ_[p]) ^ (M - s) | b i - a i) ∧
      (∃ ε : Fin d → ℤ_[p], (∀ i, (p : ℤ_[p]) ^ (2 * M - 3 * s) | ε i) ∧
        ∀ i, (b i : ℚ_[p]) - (a i : ℚ_[p]) =
          -(p : ℚ_[p]) ^ M * ((rationalMatrix p (jacobian F a))⁻¹).mulVec
            (fun j => (eval a (H j) : ℚ_[p])) i + (ε i : ℚ_[p]))) := by
  obtain ⟨b, hb, hdepth, hball, huniq, herror⟩ := exists_root_motion p s M hM F H a hFa hJ hbound
  constructor
  · exact ⟨b, ⟨hb, hball⟩, fun z hz => huniq z hz.1 hz.2⟩
  · intro z hz hzb
    rw [huniq z hz hzb]
    exact ⟨hdepth, herror⟩
```

Proof reading. The existence theorem constructs a root using the original preconditioner. Its error estimate combines the quadratic Taylor remainder with the perturbation's change under the root displacement, then applies the inverse bound, costing the stated s digits. The unique-existence endpoint transports both the depth and the error formula to every root in the uniqueness ball. It does not assign

a fixed linear map to a nonlinear finite displacement; it estimates that displacement by a linear leading term plus a rigorously bounded remainder.

10.5 Parameter and coefficient motion share one forcing term

Now $F(y,t)$ has a square solved block J and parameter block K. Perturb the coefficients by $p^M H$ and prescribe $t' = t + p^M v$. The solved-coordinate leading displacement is

$$y' - y = -p^M J^{-1}(H(a) + Kv) + \varepsilon, \qquad p^{2M-3s} \mid \varepsilon.$$

The extra Kv term is required even when $H = 0$. The source uses the actual evaluated partial derivatives in both blocks.

Contract. The theorem requires the same inverse bound on the solved block, $M > 2s$, an exact initial root, and the stated parameter displacement. Uniqueness is among roots with that exact new parameter in the depth-$s+1$ ball.

Source: `polynomial_regularity/formalization/PolynomialJointMotion.lean`.

Lines 76–105; `Komorebi.PolynomialJointMotion.exists_joint_root_motion`.

```
theorem exists_joint_root_motion (p s M : ℕ) [Fact p.Prime]
    (hM : 2 * s < M) (F H : Fin e → MvPolynomial (Fin (e + k)) ℤ_[p])
    (a : Fin (e + k) → ℤ_[p]) (v : Fin k → ℤ_[p])
    (hFa : ∀ i, eval a (F i) = 0)
    (hJ : IsUnit (rationalMatrix p (solvedJacobian F a)))
    (hbound : ∃ A : Matrix (Fin e) (Fin e) ℤ_[p],
      rationalMatrix p A = (p : ℚ_[p]) ^ s • (rationalMatrix p (solvedJacobian F a))⁻¹) :
    ∃ b : Fin (e + k) → ℤ_[p],
      (∀ i, eval b (F i) + (p : ℤ_[p]) ^ M * eval b (H i) = 0) ∧
      freePart b = freePart a + (p : ℤ_[p]) ^ M • v ∧
      (∀ i, (p : ℤ_[p]) ^ (M - s) | b i - a i) ∧
      (∀ i, (p : ℤ_[p]) ^ (s + 1) | b i - a i) ∧
      (∀ z : Fin (e + k) → ℤ_[p],
        (∀ i, eval z (F i) + (p : ℤ_[p]) ^ M * eval z (H i) = 0) →
        freePart z = freePart a + (p : ℤ_[p]) ^ M • v →
        (∀ i, (p : ℤ_[p]) ^ (s + 1) | z i - a i) → z = b) ∧
      (∃ ε : Fin e → ℤ_[p], (∀ i, (p : ℤ_[p]) ^ (2 * M - 3 * s) | ε i) ∧
        ∀ i, ((solvedPart b i : ℤ_[p]) : ℚ_[p]) - ((solvedPart a i : ℤ_[p]) : ℚ_[p]) =
          -(p : ℚ_[p]) ^ M * ((rationalMatrix p (solvedJacobian F a))⁻¹).mulVec
            (fun j => ((eval a (H j) + (parameterJacobian F a).mulVec v j : ℤ_[p]) : ℚ_[p])) i +
          (ε i : ℚ_[p])) := by
  obtain ⟨A, hA⟩ := hbound
  obtain ⟨hAJ, hJA⟩ := certificate_of_integral_inverse p s (solvedJacobian F a) A hJ hA
  obtain ⟨b, hb, ht, hdepth, hball, huniq, herror⟩ :=
    exists_joint_root_certificate p s M hM F H a v A hAJ hJA hFa
  obtain ⟨ε, hε, hformula⟩ := PolynomialMotionRational.exists_rational_error
    p s M (2 * M - 3 * s) (by omega) (solvedJacobian F a) A hA
    (solvedPart a) (solvedPart b)
    ((fun j => eval a (H j)) + (parameterJacobian F a).mulVec v) herror
  exact ⟨b, hb, ht, hdepth, hball, huniq, ε, hε, hformula⟩
```

Proof reading. The source converts the solved inverse bound into a certificate, constructs the augmented root, and obtains an integral error equation. `exists_rational_error` then divides the certificate relation

in the fraction field and identifies the first-order term. The coordinate split ensures the final error concerns solved coordinates; free coordinates already have their exact prescribed displacement.

The retained controls reject omitting the parameter forcing term, reversing its sign, and demanding an extra uniform error digit. They use explicit boundary systems, so those exclusions are mathematical constraints rather than stylistic choices. The resulting theorem is local quantitative deformation, not a global branch-selection theorem.

11 Actual derivatives and analytic local graphs

11.1 The polynomial Jacobian becomes a strict derivative

The earlier integer Taylor lemma proves divisibility. Analytic regularity needs a different bridge: actual polynomial evaluation must have the strict Fréchet derivative represented by the evaluated formal Jacobian. The source proves the scalar derivative by polynomial induction and assembles finite coordinates.

Contract. $\mathbb{K}$ is a complete nontrivially normed field; F is a vector of actual multivariate polynomials over $\mathbb{K}$. The strict derivative of evaluation at a is the continuous linear map represented by the actual Jacobian.

Source: `polynomial_regularity/formalization/PolynomialAnalyticEvaluation.lean`.

Lines 76–85; `Komorebi.PolynomialAnalyticEvaluation.hasStrictFDerivAt_evalSystem`.

```
theorem hasStrictFDerivAt_evalSystem (F : Fin m → MvPolynomial (Fin n) 𝕜)
    (a : Fin n → 𝕜) :
    HasStrictFDerivAt (evalSystem F) (matrixCLM (PolynomialWeightedInverse.jacobian F a)) a := by
  have h := hasStrictFDerivAt_pi.mpr (fun i => hasStrictFDerivAt_eval (F i) a)
  have hder : ContinuousLinearMap.pi (fun i => polynomialDeriv (F i) a) =
      matrixCLM (PolynomialWeightedInverse.jacobian F a) := by
    ext v i
    simp [PolynomialWeightedInverse.jacobian, Matrix.mulVec, dotProduct]
  rw [← hder]
  convert! h using 1
```

Proof reading. Each scalar polynomial has its proved derivative. `hasStrictFDerivAt_pi` assembles those coordinate derivatives. Extensionality of continuous linear maps compares their action on every vector and coordinate; unfolding the matrix action yields the same finite dot product as the formal partial derivatives. The final conversion identifies the evaluated system with this coordinate assembly.

Analyticity of actual polynomial evaluation is supplied through the pinned Mathlib multivariate-polynomial theorem and finite coordinate assembly. It is not a hypothesis about a proposed solution map. The development also proves the derivative block identities for the actual augmented polynomial system; a user cannot supply an unrelated matrix and call it the Jacobian.

11.2 The strict local inverse is the analytic inverse

Contract. In the inverse helper, $\mathbb{K}$ is nontrivially normed; E, F are normed spaces over it; E is complete; and $i : E \simeq_L F$ is a continuous linear equivalence. The function is analytic at a and has strict derivative i there.

Source: `polynomial_regularity/formalization/PolynomialAnalyticInverse.lean`.

Lines 28–32; `Komorebi.PolynomialAnalyticInverse.analyticAt_inverse`.

```
theorem analyticAt_inverse (ha : AnalyticAt 𝕜 f a)
    (hd : HasStrictFDerivAt f (i : E →L[𝕜] F) a) :
    AnalyticAt 𝕜 (hd.localInverse f i a) (f a) := by
  exact (hd.toOpenPartialHomeomorph f).analyticAt_symm'
    hd.mem_toOpenPartialHomeomorph_source ha hd.hasFDerivAt.fderiv
```

Proof reading. The strict derivative's local inverse is the inverse of its associated open partial homeomorphism. The Mathlib analytic inverse theorem applies to that same inverse, using the derivative identity. The proof does not construct one analytic inverse and separately assume it agrees with the root-solving inverse; the two are linked by the explicit partial homeomorphism.

These helper hypotheses are slightly different from the polynomial endpoint's ambient hypotheses. In particular, the helper only declares completeness of its source space E, while the endpoint gets it from the complete field and finite-dimensional coordinates. Reading the full variable context prevents a short excerpt from concealing these assumptions.

11.3 The augmented derivative exposes the sign

For the map $\Phi(y,t) = (F(y,t),t)$, the actual derivative at a and its inverse are

$$D\Phi(a) = \begin{pmatrix} J & K \\ 0 & I \end{pmatrix}, \qquad D\Phi(a)^{-1} = \begin{pmatrix} J^{-1} & -J^{-1}K \\ 0 & I \end{pmatrix}.$$

The source constructs the continuous linear equivalence from these matrices and verifies both products. Inserting a pure parameter displacement $(0,v)$ into the inverse yields solved displacement $-J^{-1}Kv$.

Contract. J is a square invertible matrix over the normed field and K has the specified rectangular shape. The explicit augmented equivalence's inverse, restricted to parameter input and solved output, equals the continuous linear map of $-J^{-1}K$.

Source: `polynomial_regularity/formalization/PolynomialAnalyticBlocks.lean`.

Lines 59–66; `Komorebi.PolynomialAnalyticBlocks.inverse_solved_zero`.

```
theorem inverse_solved_zero (J : Matrix (Fin e) (Fin e) 𝕜)
    (K : Matrix (Fin e) (Fin k) 𝕜) (hJ : IsUnit J) (v : Fin k → 𝕜) :
    solvedPart ((upperEquiv J K hJ).symm (join (0 : Fin e → 𝕜) v)) =
      matrixCLM (-(J⁻¹ * K)) v := by
  rw [upperEquiv_symm_apply]
  simp only [blockCertificate, upperBlock_mulVec_join, Matrix.mulVec_zero,
    zero_add, solvedPart_join]
  rfl
```

Proof reading. The proof unfolds the actual inverse equivalence and its certificate, evaluates the upper block on the joined vector, and removes the zero solved input. The negative sign is therefore an algebraic consequence of the inverse block identities. The dedicated derivative-sign negative control checks a concrete contrary statement.

11.4 The graph is exact in a product neighborhood

Contract. $\mathbb{K}$ is a complete nontrivially normed field; $F \in \mathbb{K}[Y,T]^e$ is an actual polynomial system with e solved and k parameter coordinates; $F(a) = 0$; and the solved Jacobian J is invertible. The theorem constructs open neighborhoods U, V, an analytic solution ψ, and exact equivalence $F(y,t) = 0 \iff y = \psi(t)$ for $t \in U, y \in V$.

Source: `polynomial_regularity/formalization/PolynomialAnalyticImplicit.lean`.

Lines 151–196; `Komorebi.PolynomialAnalyticImplicit.exists_analytic_implicit`.

```
theorem exists_analytic_implicit
    (F : Fin e → MvPolynomial (Fin (e + k)) 𝕜)
    (a : Fin (e + k) → 𝕜) (hF : evalSystem F a = 0)
    (hJ : IsUnit (Komorebi.PolynomialJointSystem.solvedJacobian F a)) :
    ∃ (ψ : (Fin k → 𝕜) → (Fin e → 𝕜)) (U : Set (Fin k → 𝕜))
      (V : Set (Fin e → 𝕜)),
      IsOpen U ∧ IsOpen V ∧ freePart a ∈ U ∧ solvedPart a ∈ V ∧
      ψ (freePart a) = solvedPart a ∧ AnalyticOnNhd 𝕜 ψ U ∧
      (∀ t ∈ U, ψ t ∈ V) ∧
      (∀ t ∈ U, ∀ y ∈ V, evalSystem F (join y t) = 0 ↔ y = ψ t) ∧
      HasStrictFDerivAt ψ
        (matrixCLM (-((Komorebi.PolynomialJointSystem.solvedJacobian F a)⁻¹ *
          Komorebi.PolynomialJointSystem.parameterJacobian F a))) (freePart a) := by
  let J := Komorebi.PolynomialJointSystem.solvedJacobian F a
  let K := Komorebi.PolynomialJointSystem.parameterJacobian F a
  let A := upperEquiv J K hJ
  have hshape : evalSystem (Komorebi.PolynomialJointSystem.augmented F 0) =
      fun z => join (evalSystem F z) (freePart z) := by
    funext z
    change (fun i => MvPolynomial.eval z
      (Komorebi.PolynomialJointSystem.augmented F 0 i)) =
        join (fun i => MvPolynomial.eval z (F i)) (freePart z)
    simpa only [sub_zero] using
      Komorebi.PolynomialJointSystem.eval_augmented F (0 : Fin k → 𝕜) z
  have ha := analyticAt_evalSystem (Komorebi.PolynomialJointSystem.augmented F 0) a
  have hd : HasStrictFDerivAt
      (evalSystem (Komorebi.PolynomialJointSystem.augmented F 0))
      (A : (Fin (e + k) → 𝕜) →L[𝕜] (Fin (e + k) → 𝕜)) a := by
    rw [show (A : (Fin (e + k) → 𝕜) →L[𝕜] (Fin (e + k) → 𝕜)) =
      matrixCLM (Komorebi.PolynomialBlockInverse.upperBlock J K 1) from
      upperEquiv_toCLM J K hJ]
    simpa only [Komorebi.PolynomialJointSystem.jacobian_augmented] using
      hasStrictFDerivAt_evalSystem (Komorebi.PolynomialJointSystem.augmented F 0) a
  rw [hshape] at ha hd
  obtain ⟨ψ, U, V, hU, hV, ht, hy, hψa, hψan, hψV, hgraph, hder⟩ :=
    exists_analytic_graph (evalSystem F) a hF A ha hd
  refine ⟨ψ, U, V, hU, hV, ht, hy, hψa, hψan, hψV, hgraph, ?_⟩
  have heq : (solvedPartCLM (𝕜 := 𝕜) (e := e) (k := k)).comp
      ((A.symm : (Fin (e + k) → 𝕜) →L[𝕜] (Fin (e + k) → 𝕜)).comp
        (parameterInjectionCLM (e := e) (k := k))) = matrixCLM (-(J⁻¹ * K)) := by
    apply ContinuousLinearMap.ext
    intro v
    simp only [ContinuousLinearMap.comp_apply, solvedPartCLM_apply,
      parameterInjectionCLM_apply]
    exact inverse_solved_zero J K hJ v
  simpa only [heq] using hder
```

Proof reading. The proof names the actual solved and parameter blocks J, K, constructs their upper-block continuous equivalence, and identifies the augmented polynomial map with $z \mapsto (F(z), \pi_T z)$.

Actual polynomial analyticity and the strict derivative bridge provide the inputs to the analytic graph helper. The helper takes a local inverse g and defines $\psi(t) = \pi_Y g(0, t)$. The final linear-map identity restricts g's inverse derivative to obtain $-J^{-1}K$.

The graph helper shrinks the parameter neighborhood so that four requirements hold together: the inverse identities, analyticity, the requested product neighborhood, and $\psi(U) \subseteq V$. This shrinking is part of the proof. It is not enough to know an inverse exists at the base point and then claim every point in an arbitrary ball lies in its domain.

The function ψ is defined on the ambient parameter space, but the root and regularity conclusions are restricted to U. Uniqueness is among $y \in V$. Analyticity on the neighborhood implies differentiability there; the displayed strict derivative formula is at the specified base point. The open-set selection theorem later obtains the analogous derivative at every point by applying the local theorem repeatedly.

11.5 A supplied continuous root selection acquires regularity

Contract. Keep the same field and solved-block nonsingularity. A supplied map g is continuous at the base parameter, has the correct base value, and solves the exact polynomial equation on a neighborhood. No derivative or analyticity of g is assumed.

Source: `polynomial_regularity/formalization/PolynomialAnalyticSelection.lean`.

Lines 39–56; `Komorebi.PolynomialAnalyticSelection.continuous_selection_regularity`.

```
theorem continuous_selection_regularity
    (F : Fin e → MvPolynomial (Fin (e + k)) 𝕜)
    (a : Fin (e + k) → 𝕜)
    (hJ : IsUnit (PolynomialJointSystem.solvedJacobian F a))
    (g : (Fin k → 𝕜) → (Fin e → 𝕜))
    (hg : ContinuousAt g (freePart a)) (hg₀ : g (freePart a) = solvedPart a)
    (hroot : ∀ᶠ t in 𝓝 (freePart a), evalSystem F (join (g t) t) = 0) :
    AnalyticAt 𝕜 g (freePart a) ∧
      HasStrictFDerivAt g
        (matrixCLM (-((PolynomialJointSystem.solvedJacobian F a)⁻¹ *
          PolynomialJointSystem.parameterJacobian F a))) (freePart a) := by
  have hF : evalSystem F a = 0 := by
    simpa only [hg₀, join_parts] using hroot.self_of_nhds
  obtain ⟨ψ, U, V, hU, hV, ht, hy, _hψ₀, hψ, _hψV, hgraph, hdψ⟩ :=
    exists_analytic_implicit F a hF hJ
  have heq := continuous_selection_eventually_eq (evalSystem F) g ψ
    (freePart a) (solvedPart a) U V hU hV ht hy hgraph hg hg₀ hroot
  exact ⟨(hψ (freePart a) ht).congr heq.symm, hdψ.congr_of_eventuallyEq heq.symm⟩
```

Proof reading. The root equation at the base point follows from the eventual neighborhood equation and the given base value. The analytic graph theorem constructs ψ. Continuity places $g(t)$ in V for all sufficiently close t, while openness places t in U. The exact graph equivalence forces $g = \psi$ as germs. Analyticity and the strict derivative transfer through eventual equality.

For a continuous exact root selection on an open set, the source requires an invertible solved Jacobian at every point of that set. It then proves analyticity and the actual implicit derivative at every point. A single nonsingular point does not justify regularity of a branch through later singular points. Arbitrary global branch choices are outside this germ comparison.

11.6 Integral coefficients are evaluated in the fraction field

Contract. For any prime p, the coefficients and base point may lie in $\mathbb{Z}_p$. Only the solved Jacobian after inclusion in $\mathbb{Q}_p$ must be invertible. The resulting neighborhoods and analytic map live in the ambient $\mathbb{Q}_p$ vector spaces.

Source: `polynomial_regularity/formalization/PolynomialRegularityIntegral.lean`.

Lines 87–123; `Komorebi.PolynomialRegularityIntegral.exists_analytic_integral_implicit`.

```
theorem exists_analytic_integral_implicit (p : ℕ) [Fact (Nat.Prime p)]
    (F : Fin e → MvPolynomial (Fin (e + k)) ℤ_[p])
    (a : Fin (e + k) → ℤ_[p]) (hF : ∀ i, eval a (F i) = 0)
    (hJ : IsUnit (rationalMatrix p (Komorebi.PolynomialJointSystem.solvedJacobian F a))) :
    ∃ (ψ : (Fin k → ℚ_[p]) → (Fin e → ℚ_[p])) (U : Set (Fin k → ℚ_[p]))
      (V : Set (Fin e → ℚ_[p])),
      IsOpen U ∧ IsOpen V ∧ rationalPoint p (freePart a) ∈ U ∧
      rationalPoint p (solvedPart a) ∈ V ∧
      ψ (rationalPoint p (freePart a)) = rationalPoint p (solvedPart a) ∧
      AnalyticOnNhd ℚ_[p] ψ U ∧ (∀ t ∈ U, ψ t ∈ V) ∧
      (∀ t ∈ U, ∀ y ∈ V,
        (∀ i, eval₂ (algebraMap ℤ_[p] ℚ_[p]) (join y t) (F i) = 0) ↔ y = ψ t) ∧
      HasStrictFDerivAt ψ
        (matrixCLM (-((rationalMatrix p
          (Komorebi.PolynomialJointSystem.solvedJacobian F a))⁻¹ *
          Matrix.of (fun i j => algebraMap ℤ_[p] ℚ_[p] (parameterJacobian F a i j)))))
        (rationalPoint p (freePart a)) := by
  have hroot : evalSystem (rationalSystem p F) (rationalPoint p a) = 0 := by
    ext i
    exact rationalSystem_root p F a hF i
  have hunit : IsUnit (Komorebi.PolynomialJointSystem.solvedJacobian
      (rationalSystem p F) (rationalPoint p a)) := by
    simpa only [solvedJacobian_rationalSystem] using hJ
  obtain ⟨ψ, U, V, hU, hV, ht, hy, hψa, hψan, hψV, hgraph, hder⟩ :=
    Komorebi.PolynomialAnalyticImplicit.exists_analytic_implicit
      (rationalSystem p F) (rationalPoint p a) hroot hunit
  refine ⟨ψ, U, V, hU, hV, ht, hy, hψa, hψan, hψV, ?_, ?_⟩
  · intro t ht' y hy'
    have h := hgraph t ht' y hy'
    simpa only [evalSystem, rationalSystem, MvPolynomial.eval_map,
      funext_iff, Pi.zero_apply] using h
  · have hK : parameterJacobian (rationalSystem p F) (rationalPoint p a) =
        Matrix.of (fun i j => algebraMap ℤ_[p] ℚ_[p] (parameterJacobian F a i j)) := by
      ext i j
      exact eval_pderiv_rationalSystem p F a i (Fin.natAdd e j)
    rw [solvedJacobian_rationalSystem, hK, rationalPoint_freePart] at hder
    exact hder
```

Proof reading. The source proves that evaluation, formal partial derivatives, and both Jacobian blocks commute with the coefficient inclusion. It applies the actual polynomial implicit theorem to the included system and point, then rewrites its equation and derivative back in terms of the original integral coeffi-

cients. Invertibility modulo p is not required. The included solved Jacobian may therefore be singular modulo p while remaining invertible over $\mathbb{Q}_p$.

This is an ambient-field analytic theorem. It does not make $\mathbb{Z}_p$ a vector space over $\mathbb{Q}_p$, assert that every nearby analytic value is integral, or supply a new explicit integral radius. The earlier weighted and residue-ball theorems retain their own quantitative domains. Nor does it prove a branch through a point whose solved Jacobian is singular over the fraction field. General singular persistence and global compatibility remain separate research problems.

12 Evidence replay and limits of the final edition

12.1 What was freshly repeated

The accompanying `audit_sources.py` invokes the preserved Python evidence reconciler and writes to a separate guide audit directory. The reproduced evidence index and declaration CSV are byte-identical to the preserved index. Every one of the 202 manuscript/source links is hash-checked and resolves its explicitly named declaration. The final manuscript-source hashes in that historical statement index are also matched to the preserved consolidated source.

At the latest polynomial checkpoint, the source parser freshly inspects all 55 proof modules, eight positive support modules, and 13 contract modules. Its 515 named entries equal the recorded inventory. The strict axiom-log parser freshly compares its complete expected name set with the retained axiom report; it rejects duplicate names, missing or extra names, diagnostic errors, and an axiom outside the permitted three. All 12 negative-control source hashes and log hashes match; their recorded exits are nonzero, no timeout is recorded, and their expected diagnostic appears in the retained log. These checks inspect inherited outcomes. They do not execute those Lean controls again.

The unchanged Python source/axiom parser regression was actually run in the current environment. It retains legal apostrophes in distinct declaration names and rejects three deliberately corrupted audit records. This fresh parser regression establishes the behavior of the parser on those inputs. It does not validate the Lean kernel or replace the inherited theorem compilation.

The latest regularity report records a run beginning 8 September 2026 at 19:00:15.797459 UTC, with a reported duration of 372.804 seconds. It records Lean 4.34.0-rc2, Mathlib revision `aeb72f1e9981b6d35b426b16dca7759be3e7ba06`, fresh project compilation at that time, and no Mathlib rebuild. The compiler binary, adapter, Lake manifest and verifier identities remain recorded in that report. Present static checks authenticate available project scripts and source artifacts against their recorded hashes; the separately supplied runtime kit is absent, so its historical binary and dependency hashes cannot be freshly re-observed here.

12.2 The negative controls mark mathematical boundaries

Retained regularity control	Boundary being guarded
`InvalidGlobalUniqueness`	Local root uniqueness cannot be extended to all roots.
`InvalidRectangularUniqueness`	A surjective rectangular digit map can have a non-trivial kernel.
`InvalidPerturbationExtraPrecision`	A perturbation does not supply a free additional congruence digit.
`InvalidWeightedBoundary`	The weighted residual threshold requires $r > 2s$.
`InvalidWeightedExtraPrecision`	The inverse-bound loss in retained precision can be sharp.
`InvalidMotionExtraErrorDigit`	The first-order remainder exponent cannot be uniformly increased.
`InvalidPrescribedParameter`	A fixed parameter must satisfy its actual digit equation.

Retained regularity control	Boundary being guarded
`InvalidPrescribedFullJacobian`	Full Jacobian surjectivity does not replace solved-block surjectivity.
`InvalidPrescribedRectangularUniqueness`	Prescribed parameters do not eliminate a solved-block kernel.
`InvalidJointMissingParameterTerm`	Joint motion needs the actual Kv forcing term.
`InvalidJointParameterSign`	Solved motion has the negative inverse-Jacobian sign.
`InvalidRegularityDerivativeSign`	The analytic implicit derivative is $-J^{-1}K$.

The controls compile attempted statements using positive examples that contradict the strengthened conclusions. Their expected compiler rejection is recorded evidence that the false theorem application does not type-check in the inherited run. The mathematical explanation of the relevant examples appears in the manuscript and source control modules. A failed attempt alone does not prove the negation of an arbitrary unformalized conjecture.

12.3 How to perform a new compiler replay

The preserved package gives explicit drivers for each frontier. The newest polynomial driver is `replay_paper9_regularity.py`; the arithmetic/shared-correction driver is `replay_shared_correction.py`. They require the separately supplied Lean/Mathlib runtime kit and a new output directory. Use the exact pinned revisions, validate the actual kit, and retain all source, log, inventory, diagnostic, and axiom outputs from that new run. A clean replay should never overwrite a historical output directory or describe an old log as newly generated.

The supplied guide PDF has its own isolated deterministic XeLaTeX builder. It uses the copied instructional chapters, generated catalogs, and exact excerpt assets. It does not invoke Lean. The source-dependent preparation and audit scripts accept the consolidated archive root as an explicit argument, while the PDF builder needs only the companion's editable directory. Keeping these operations distinct makes the provenance reviewable.

12.4 Unclosed obligations and recovered history

Paper IX closes the recorded arithmetic, smooth finite lifting, prescribed towers, quantitative deformation, and local analytic regularity results under their stated hypotheses. It retains open unrestricted native presentation invariance, completed/global compatibility and descent, general singular persistence with higher obstructions, and other specified arithmetic extensions. The analytic inverse theorem does not close those programs.

The preserved original 28-page manuscript is available, but its original editable TeX, figure-generation sources, and complete historical experiment package have not been recovered. The reconstruction and all later continuation sources retain their own provenance. No formal blanket certificate is assigned to every historical text merely because it is preserved in the package. The Bernoulli boundary restoration has its classical derivation and an explicit unmatched Lean bridge.

Two strengthening inventory scalars were stale: the stored theorem count 179 differs from 215 exact theorem commands, and the stored definition count 12 differs from 20. The historical file is preserved

unchanged; the final index uses the actual source entries and named audit sets. This discrepancy is documented rather than silently repaired in the raw archive.

13 Current manuscript-to-guide navigation

This table resolves all 62 inherited statement entries against the current 232-page monograph. The paper column gives physical PDF pages, counting the cover. The guide column uses printed pages. The 202 exact formal source links are retained in the paired JSON without changing their source hashes. A separate classical example follows the indexed entries. Current monograph SHA-256: `1975d95d124cdb303452950c272917c2e3fc20c19fdc882ad39b8f9744d328ca`.

Statement	Paper	Teaching chapter	Guide
3.1 `def:arith-euler`	16	Arithmetic	19
3.4 `def:arith-precision`	18	Arithmetic	19
3.6 `thm:arith-nishibuchi`	19	Arithmetic	19
3.7 `thm:arith-divisor-base`	19	Arithmetic	19
3.10 `lem:arith-newton-period`	22	Arithmetic	19
3.11 `thm:arith-dyadic-differences`	22	Dyadic results	23
3.12 `cor:arith-amplified`	24	Dyadic results	23
3.13 `cor:arith-allrow-period`	24	Dyadic results	23
3.14 `lem:arith-dominance`	24	Dyadic results	23
3.15 `thm:arith-exact`	25	Dyadic results	23
3.16 `cor:arith-least-period`	26	Dyadic results	23
3.17 `def:arith-odd-source`	26	Dyadic results	23
3.18 `thm:arith-odd-reconstruction`	27	Dyadic results	23
3.19 `cor:arith-odd-ladder`	27	Dyadic results	23
3.20 `def:arith-truncation`	28	Dyadic results	23
3.21 `lem:arith-reconstruction`	28	Dyadic results	23
3.22 `lem:arith-kappa`	29	Dyadic results	23
3.23 `thm:arith-sharp-tower`	29	Dyadic results	23
3.24 `prop:arith-row-weight`	30	Dyadic results	23
3.25 `thm:arith-isometry`	30	Dyadic results	23
3.26 `cor:arith-affine-image`	31	Dyadic results	23
3.2 `prop:arith-denominator`	17	Arithmetic	19
3.5 `prop:arith-lucas-window`	18	Arithmetic	19
3.12 `eq:arith-signed-zigzag`	27	Dyadic results	23
3.8 `ex:arith-normalization-loss`	21	Arithmetic	19
3.9 `cor:arith-window-period`	21	Arithmetic	19
10.1 `lem:polynomial-taylor`	51	Polynomial digits	31
10.2 `thm:finite-polynomial-obstruction`	51	Polynomial digits	31
10.3 `prop:affine-obstruction`	52	Affine obstructions	27
10.4 `prop:representative-independence`	52	Affine obstructions	27
10.5 `thm:common-correction`	53	Affine obstructions	27
10.6 `cor:shared-polynomial`	53	Affine obstructions	27
10.7 `thm:lift-torsor-count`	54	Polynomial digits	31
10.8 `thm:smooth-tower`	55	Compatible towers	34
10.9 `thm:smooth-padic-root`	55	Compatible towers	34
10.11 `prop:smooth-coefficient-stability`	56	Compatible towers	34
10.12 `thm:continuous-parameter-chart`	56	Parameters and charts	37
10.13 `thm:prescribed-obstruction`	57	Parameters and charts	37
10.14 `thm:prescribed-integer-tower`	58	Parameters and charts	37
10.15 `thm:prescribed-padic-tower`	58	Parameters and charts	37
10.16 `thm:weighted-lifting`	59	Weighted lifting	41
10.18 `thm:coefficient-motion`	60	Weighted lifting	41

Statement	Paper	Teaching chapter	Guide
10.19 `thm:joint-deformation`	61	Weighted lifting	41
10.17 `ex:weighted-boundary`	60	Weighted lifting	41
10.20 `ex:motion-boundary`	62	Weighted lifting	41
2.1 `thm:residue-reconstruction`	10	Exact coordinates	14
2.2 `thm:egf-coordinates`	11	Exact coordinates	14
2.3 `thm:hurwitz-product`	11	Exact coordinates	14
2.4 `thm:triangular-foundation`	12	Exact coordinates	14
2.5 `thm:filtered-foundation`	12	Exact coordinates	14
2.6 `thm:mahler-foundation`	13	Exact coordinates	14
2.7 `thm:weighted-mahler-foundation`	13	Exact coordinates	14
2.8 `thm:unit-modes-foundation`	14	Exact coordinates	14
2.9 `thm:finite-unit-foundation`	14	Exact coordinates	14
2.10 `thm:companion-foundation`	15	Exact coordinates	14
2.11 `thm:crt-foundation`	15	Exact coordinates	14
11.1 `lem:actual-polynomial-derivative`	63	Analytic regularity	46
11.2 `lem:augmented-analytic-inverse`	63	Analytic regularity	46
11.3 `thm:analytic-local-graph`	64	Analytic regularity	46
11.4 `thm:selection-regularity`	64	Analytic regularity	46
11.5 `cor:selection-open-regularity`	65	Analytic regularity	46
11.6 `cor:integral-analytic-graph`	65	Analytic regularity	46
3.3 `ex:arith-bernoulli`	17	Classical derivation	12

13.1 Strengthening and worked-example routes

These locators navigate the newer written proofs and the exact worked atlas. Their formal coverage is discussed in Section 1. The page map does not turn a written proof or a finite example into a compiled Lean endpoint.

Current statement or example	Paper PDF page
4.2 `lem:translated-hurwitz-product`	32
4.3 `lem:translated-hurwitz-inverse`	33
4.4 `thm:all-prime-power-transfer`	34
4.5 `cor:all-dyadic-final`	34
5.1 `thm:four-factorial-bound`	35
5.2 `lem:bernoulli-central-pd`	39
6.1 `thm:zero-one-exact-mixed`	41
6.3 `thm:two-exact-power-orders`	42
7.1 `lem:newton-exact-isometry-criterion`	44
7.2 `thm:rank-four-exact-periods`	44
7.3 `cor:four-exact-images`	45
7.4 `thm:lower-half-all-dyadic-isometry`	46
7.5 `prop:two-four-eight-isometry-table`	46
8.1 `thm:four-taylor-radii`	47
9.1 `cor:final-euler-congruence`	49
F.1 `atlas:reciprocal`	223
F.2 `atlas:normalization`	224
F.3 `atlas:sign`	225
F.4 `atlas:newton`	226
F.5 `atlas:smooth`	227

Current statement or example	Paper PDF page
F.6 `atlas:blocked`	228
F.7 `atlas:shared`	229
F.8 `atlas:weighted`	230

14 The retained verification frontiers

Each row is one retained compilation checkpoint. Later polynomial rows carry earlier sources, so their counts must not be added. Compilation is historical; the source/log reconciliation was freshly repeated for this edition.

Frontier	Proof	Support	Contracts	Negative	Audits
baseline	79	0	0	0	1000
strengthening	22	0	0	14	246
shared_correction	6	0	2	3	60
polynomial_lifting	9	0	4	3	91
polynomial_counts	9	0	3	3	106
polynomial_towers	14	0	3	3	127
polynomial_deformation	17	1	3	3	159
polynomial_charts	25	2	6	4	244
polynomial_weighted	30	3	8	6	304
polynomial_motion	35	4	9	7	326
polynomial_prescribed	39	5	10	8	375
polynomial_parameter_towers	45	6	11	9	408
polynomial_joint_motion	49	7	12	11	460
polynomial_regularity	55	8	13	12	515
native_affine	2	0	0	0	17

15 Latest polynomial dependency map

These are direct imports of the 55 proof modules in the latest retained polynomial gate. The complete inventory preserves imports for control and contract sources as well. External Mathlib imports refer to the recorded pinned dependency, whose runtime is not freshly rebuilt or inspected in this edition.

Module	Direct project imports
`AffineObstructionCore`	Mathlib only
`FiniteFiberCounts`	Mathlib only
`PadicTowerLimit`	Mathlib only
`PolynomialAnalyticInverse`	Mathlib only
`PolynomialMinorSelection`	Mathlib only
`PolynomialPadicContraction`	Mathlib only
`PolynomialTaylor`	Mathlib only
`PolynomialTaylorRing`	Mathlib only
`PolynomialWeightedInverse`	Mathlib only
`ResidueLiftDigits`	Mathlib only
`IntegerFiniteCokernel`	`AffineObstructionCore`
`PolynomialAnalyticEvaluation`	`PolynomialWeightedInverse`
`PolynomialMotionRational`	`PolynomialWeightedInverse`
`PolynomialObstruction`	`AffineObstructionCore`; `PolynomialTaylor`
`PolynomialTaylorDisplacement`	`PolynomialTaylorRing`
`PolynomialWeightedNormalization`	`PolynomialTaylorRing`; `PolynomialWeightedInverse`
`PolynomialDigitBridge`	`PolynomialObstruction`
`PolynomialMotionEstimate`	`PolynomialTaylorDisplacement`; `PolynomialWeightedInverse`
`PolynomialMotionNormalization`	`PolynomialWeightedNormalization`
`PolynomialWeightedLift`	`PolynomialTaylorRing`; `PolynomialPadicContraction`; `PolynomialWeightedInverse`; `PolynomialWeightedNormalization`
`PolynomialLiftCounts`	`PolynomialDigitBridge`; `ResidueLiftDigits`; `FiniteFiberCounts`; `IntegerFiniteCokernel`
`PolynomialRootMotion`	`PolynomialWeightedLift`; `PolynomialMotionNormalization`; `PolynomialMotionEstimate`; `PolynomialMotionRational`
`PolynomialSurjectiveStep`	`PolynomialDigitBridge`
`PolynomialIntegerTower`	`PolynomialSurjectiveStep`
`PolynomialSmoothCounts`	`PolynomialIntegerTower`; `PolynomialLiftCounts`
`PolynomialTowerControls`	`PolynomialIntegerTower`; `PadicTowerLimit`
`PolynomialPadicLift`	`PolynomialSmoothCounts`; `PadicTowerLimit`
`PolynomialCoefficientPerturbation`	`PolynomialPadicLift`
`PolynomialPadicUnique`	`PolynomialPadicLift`
`PolynomialParameterSystem`	`PolynomialPadicLift`
`PolynomialBlockInverse`	`PolynomialWeightedInverse`; `PolynomialParameterSystem`
`PolynomialCoordinateReorder`	`PolynomialParameterSystem`; `PolynomialMinorSelection`
`PolynomialInverseCongruence`	`PolynomialPadicUnique`
`PolynomialLocalSurjective`	`PolynomialPadicUnique`

Module	Direct project imports
PolynomialParameterPath	PolynomialPadicUnique
PolynomialPrescribedSystem	PolynomialParameterSystem; FiniteFiberCounts
PolynomialRootStability	PolynomialPadicUnique; PolynomialCoefficientPerturbation
PolynomialAnalyticBlocks	PolynomialAnalyticEvaluation; PolynomialBlockInverse
PolynomialJointSystem	PolynomialParameterSystem; PolynomialMotionNormalization; PolynomialBlockInverse
PolynomialLocalInverse	PolynomialInverseCongruence; PolynomialLocalSurjective
PolynomialPrescribedLifts	PolynomialPrescribedSystem; PolynomialLiftCounts
PolynomialPrescribedTowerLimit	PolynomialParameterPath; PolynomialParameterSystem
PolynomialAnalyticImplicit	PolynomialAnalyticInverse; PolynomialJointSystem; PolynomialAnalyticBlocks
PolynomialJointMotion	PolynomialJointSystem; PolynomialBlockInverse; PolynomialRootMotion
PolynomialParameterChart	PolynomialLocalInverse; PolynomialParameterSystem
PolynomialPrescribedCounts	PolynomialPrescribedLifts; FiniteFiberCounts
PolynomialAnalyticSelection	PolynomialAnalyticImplicit
PolynomialGeneralChart	PolynomialParameterChart; PolynomialCoordinateReorder
PolynomialJointDeformation	PolynomialJointMotion
PolynomialPrescribedObstruction	PolynomialPrescribedLifts; PolynomialPrescribedCounts
PolynomialPrescribedStep	PolynomialPrescribedCounts; PolynomialSurjectiveStep
PolynomialPrescribedTowerUnique	PolynomialParameterChart; PolynomialPrescribedSystem
PolynomialRegularityIntegral	PolynomialAnalyticImplicit
PolynomialPrescribedIntegerTower	PolynomialPrescribedStep; PolynomialIntegerTower
PolynomialPrescribedPadicTower	PolynomialPrescribedIntegerTower; PolynomialPrescribedTowerLimit; PolynomialPrescribedTowerUnique

The proof chain deliberately retains its interfaces: integer Taylor identities feed digit obstructions and towers; contraction and inverse certificates feed quantitative weighted lifting; actual polynomial strict derivatives and augmented inverse blocks feed the analytic graph. A direct import is a dependency relation; it is not by itself a theorem correspondence or a fresh compiler check.

16 Complete named-source catalog

This catalog contains all 1,876 declaration/source-version rows from the preserved frontier inventory, grouped under 199 canonical source files. The paired CSV retains every exact-copy location, role, source version, frontier, and inherited axiom closure. A theorem command, a control, a construction, and a source version are different quantities. A new name does not establish a new mathematical result.

AllRatiosDwork

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/AllRatiosDwork.lean`.

SHA-256: `26a89250a7cd30e0b8bf22aa1159553965c182997a21dfd2c1c257cbc88a1948`; 4 rows.

Namespace: `KomorebiPaperIX.AllRatiosDwork`

`all_ratios_flat` (theorem; proof_source)

`packet_window_flat` (theorem; proof_source)

`ratio_window_flat` (theorem; proof_source)

`source_window_flat` (theorem; proof_source)

AllResidueEuler

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/AllResidueEuler.lean`.

SHA-256: `173902631c8b2ba6e52eecbf01e27be748ffe1e863811c1a9ec23ba46c55b9a9`; 12 rows.

Namespace: `KomorebiPaperIX.AllResidueEuler`

`all_residue_flat` (theorem; proof_source)

`decode_unnormalized` (theorem; proof_source)

`global_normalized_precision` (theorem; proof_source)

`global_unnormalized_bound` (theorem; proof_source)

`global_unnormalized_order` (theorem; proof_source)

`global_unnormalized_precision` (theorem; proof_source)

`unnormalizedLift` (def; proof_source)

`unnormalizedLift_coe` (theorem; proof_source)

`unnormalizedSeries` (def; proof_source)

`unnormalized_integral` (theorem; proof_source)

`unnormalized_successor` (theorem; proof_source)

`unnormalized_support` (theorem; proof_source)

AllResidueExamples

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/AllResidueExamples.lean`.

SHA-256: `8928578ff4f2e956636043443dc66fdc6a147e7318565018b7f26f289ad8eea4`; 8 rows.

Namespace: `KomorebiPaperIX.AllResidueExamples`

`no_loss_still_false` (theorem; positive_control)

`singular_second_difference` (theorem; positive_control)

`singular_second_valuation` (theorem; positive_control)

`singular_third` (theorem; positive_control)

`singular_unnormalized_second` (theorem; positive_control)

`upper_regular_coefficients` (theorem; positive_control)

`upper_regular_global_bound` (theorem; positive_control)

`upper_regular_unnormalized_second` (theorem; positive_control)

BinomialLogSeries

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/BinomialLogSeries.lean`.

SHA-256: `8370192c821ba82844293fac3beae8244d437ef97da65c2a57ea44b1d93b5bab`; 9 rows.

Namespace: `KomorebiPaperIX.BinomialLogSeries`

`coeff_logPullback` (theorem; proof_source)

`expAt` (def; proof_source)

`expAt_derivative` (theorem; proof_source)

`factorial_coeff_logPullback` (theorem; proof_source)

`fallingAt_succ` (theorem; proof_source)

`logPullback` (def; proof_source)

`logPullback_equation` (theorem; proof_source)

`logPullback_recurrence` (theorem; proof_source)

`logPullback_zero` (theorem; proof_source)

CRTPackets

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/CRTPackets.lean`.

SHA-256: `1613b875e76274f541f74804bc489f696caa636128eb73ffeec29dfeab439163`; 6 rows.

Namespace: `KomorebiPaperIX.CRTPackets`

`addressEquiv` (def; proof_source)

`address_bijective` (theorem; proof_source)

`address_components` (theorem; proof_source)

`packet_factorization` (theorem; proof_source)

`residue_equivalence` (theorem; proof_source)

`series_packet_factorization` (theorem; proof_source)

CompanionPackets

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/CompanionPackets.lean`.

SHA-256: `651d904dd712bc8aa0d0a6ed18fa07f3716639ebfb5cb6412fbc8fb0188452d5`; 19 rows.

Namespace: `KomorebiPaperIX.Companion`

`charpoly_coeff_dimension` (theorem; proof_source)

`charpoly_sum_powers` (theorem; proof_source)

`companion_jump` (theorem; proof_source)

`orbit` (def; proof_source)

`orbit_of_recurrence` (theorem; proof_source)

`orbit_succ` (theorem; proof_source)

`orbit_zero` (theorem; proof_source)

`packet` (def; proof_source)

`packet_advance` (theorem; proof_source)

`packet_as_jump_orbit` (theorem; proof_source)

`packet_charpoly_annihilation` (theorem; proof_source)

`packet_charpoly_recurrence` (theorem; proof_source)

`packet_coordinate_recurrence` (theorem; proof_source)

`packet_observable_recurrence` (theorem; proof_source)

`packet_refinement` (theorem; proof_source)

`packet_step` (theorem; proof_source)

`packet_zero_step` (theorem; proof_source)

`source_charpoly_recurrence` (theorem; proof_source)

`source_companion_jump` (theorem; proof_source)

CongruentialEuler

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/CongruentialEuler.lean`.

SHA-256: `349af85472b8e503e57e27292629e6699e1637e3ff7a81fab77eb09c3ccf49b1`; 23 rows.

Namespace: `KomorebiPaperIX.Congruential`

`clearing` (def; proof_source)

`clearing_equals_generic` (theorem; proof_source)

`clearing_product` (theorem; proof_source)

`coeff` (def; proof_source)

`coeff_zero` (theorem; proof_source)

`coefficient_recurrence` (theorem; proof_source)

`coefficient_step` (theorem; proof_source)

`coefficient_unique` (theorem; proof_source)

`diagonal_ne_zero` (theorem; proof_source)

`diagonal_positive` (theorem; proof_source)

`factorial_coefficient_identity` (theorem; proof_source)

`kernel_constant_one` (theorem; proof_source)

`packetSeries` (def; proof_source)

`packet_is_actual_inverse` (theorem; proof_source)

`packet_series_coeff` (theorem; proof_source)

`packet_times_kernel` (theorem; proof_source)

`rhs` (def; proof_source)

`row` (def; proof_source)

`sectionKernel` (def; proof_source)

`triangular_inverse` (theorem; proof_source)

`universal_denominator_clearing` (theorem; proof_source)

`unnormalized` (def; proof_source)

`unnormalized_zero` (theorem; proof_source)

ConjectureExamples

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/ConjectureExamples.lean`.

SHA-256: `46651c5e47bd95f70deb09764634766e6a53f21e02b3585f4b62f092e702bfae`; 7 rows.

Namespace: `KomorebiPaperIX.ConjectureExamples`

`dyadic_all_residues` (theorem; positive_control)

`dyadic_modulus_two` (theorem; positive_control)

`dyadic_nonzero_address_error` (theorem; positive_control)

`dyadic_nonzero_address_valuation` (theorem; positive_control)

`four_first_coefficients` (theorem; positive_control)

`four_first_unnormalized` (theorem; positive_control)

`odd_singular_address` (theorem; positive_control)

CubicDworkAlgebra

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/CubicDworkAlgebra.lean`.

SHA-256: `b102ebc4e04ed0c7bead62b87c653925e47a4c2d6359d699ccb6953e3e898310`; 2 rows.

Namespace: `KomorebiPaperIX.CubicDworkAlgebra`

`B` (def; proof_source)

`B_mul` (theorem; proof_source)

CubicDworkClosure

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/CubicDworkClosure.lean`.

SHA-256: `9ec22df4b14920247743f017401f253de62ed7ed335511d85e17123d66dbb256`; 22 rows.

Namespace: `KomorebiPaperIX.CubicDworkClosure`

`B_deriv` (theorem; proof_source)

`B_one` (theorem; proof_source)

`bounded` (def; proof_source)

`bounded_add` (theorem; proof_source)

`bounded_deriv` (theorem; proof_source)

`bounded_inverse` (theorem; proof_source)

`bounded_middle` (theorem; proof_source)

`bounded_mono` (theorem; proof_source)

`bounded_mul` (theorem; proof_source)

`bounded_nat_mul` (theorem; proof_source)

`bounded_neg` (theorem; proof_source)

`bounded_sub` (theorem; proof_source)

`bounded_succ` (theorem; proof_source)

`bounded_sum` (theorem; proof_source)

`bounded_zero` (theorem; proof_source)

`deriv_add` (theorem; proof_source)

`deriv_one_eq_zero` (theorem; proof_source)

`flat` (def; proof_source)

`flat_inverse` (theorem; proof_source)

`flat_mul` (theorem; proof_source)

`inverse_defect` (theorem; proof_source)

`iterate_B_deriv` (theorem; proof_source)

CyclicAmiceCoefficients

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/CyclicAmiceCoefficients.lean`.

SHA-256: `b964b86e9c28490a1c167c4e1e6f9eec20c4b1f9a8a87462a4be3eb56b92e271`; 14 rows.

Namespace: `KomorebiPaperIX.CyclicAmiceCoefficients`

`CyclicRing` (abbrev; proof_source)

`coeff_nat_mul` (theorem; proof_source)

`coeff_nat_pow_mul` (theorem; proof_source)

`coefficient` (def; proof_source)

`coefficient_weighted_integral` (theorem; proof_source)

`generator` (def; proof_source)

`generator_pow` (theorem; proof_source)

`generator_relation` (theorem; proof_source)

`normalizer` (def; proof_source)

`normalizer_integral` (theorem; proof_source)

`normalizer_nonzero` (theorem; proof_source)

`normalizer_valuation` (theorem; proof_source)

`numerator` (def; proof_source)

`numerator_divisibility` (theorem; proof_source)

CyclotomicFiltration

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/CyclotomicFiltration.lean`.

SHA-256: `cd237b7f6ca7687164778924b1e569e0bd361306732d161e41967bfca8947296`; 7 rows.

Namespace: `KomorebiPaperIX.CyclotomicFiltration`

`augmentation_mem` (theorem; proof_source)

`augmentation_pow_mem` (theorem; proof_source)

`factorial_power_bound` (theorem; proof_source)

`prime_mem` (theorem; proof_source)

`torsionIdeal` (def; proof_source)

`torsion_ideal_factorial_bound` (theorem; proof_source)

`torsion_ideal_pow` (theorem; proof_source)

DifferenceFiltration

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/DifferenceFiltration.lean`.

SHA-256: `572ce1909ca441ae5478f0bfd4fc4af64142c7c948344ab259eb71b8d8dfc9d6`; 4 rows.

Namespace: `KomorebiPaperIX.DifferenceFiltration`

`difference_map` (theorem; proof_source)

`saturated_bound_iff` (theorem; proof_source)

`saturated_bound_of_triangular` (theorem; proof_source)

`tail_period_higher_difference` (theorem; proof_source)

DifferenceStepMultiples

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/DifferenceStepMultiples.lean`.

SHA-256: `128fbae2dead2a970a25cb8ac21309520bdbfb7067e74f8489180d81c3946f62`; 6 rows.

Namespace: `KomorebiPaperIX.StepMultiples`

`difference_commute_one` (theorem; proof_source)

`difference_translate` (theorem; proof_source)

`divisibility_transfer` (theorem; proof_source)

`saturated_multiple_step` (theorem; proof_source)

`single_step_telescope` (theorem; proof_source)

`translateSum` (def; proof_source)

DworkAlgebra

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/DworkAlgebra.lean`.

SHA-256: `71164e13e361dffe632db205f5c1634a1aa031080d6c969e79ee1e3bbdd4fc6d`; 10 rows.

Namespace: `KomorebiPaperIX.DworkAlgebra`

`B` (def; proof_source)

`B_mul` (theorem; proof_source)

`deriv` (def; proof_source)

`deriv_leibniz` (theorem; proof_source)

`deriv_one` (theorem; proof_source)

`deriv_succ` (theorem; proof_source)

`deriv_succ_right` (theorem; proof_source)

`deriv_zero` (theorem; proof_source)

`middle` (def; proof_source)

`prime_power_leibniz` (theorem; proof_source)

DworkClosure

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/DworkClosure.lean`.

SHA-256: `c55b79b8105e0e276811fd6ef58e67dbcd3853183c7fff8c3e065f6c0f3fe9fc`; 22 rows.

Namespace: `KomorebiPaperIX.DworkClosure`

`B_deriv` (theorem; proof_source)

`B_one` (theorem; proof_source)

`bounded` (def; proof_source)

`bounded_add` (theorem; proof_source)

`bounded_deriv` (theorem; proof_source)

`bounded_inverse` (theorem; proof_source)

`bounded_middle` (theorem; proof_source)

`bounded_mono` (theorem; proof_source)

`bounded_mul` (theorem; proof_source)

`bounded_nat_mul` (theorem; proof_source)

`bounded_neg` (theorem; proof_source)

`bounded_sub` (theorem; proof_source)

`bounded_succ` (theorem; proof_source)

`bounded_sum` (theorem; proof_source)

`bounded_zero` (theorem; proof_source)

`deriv_add` (theorem; proof_source)

`deriv_one_eq_zero` (theorem; proof_source)

`flat` (def; proof_source)

`flat_inverse` (theorem; proof_source)

`flat_mul` (theorem; proof_source)

`inverse_defect` (theorem; proof_source)

`iterate_B_deriv` (theorem; proof_source)

DworkCriterion

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/DworkCriterion.lean`.

SHA-256: `62db8d69f80034062ae976c9155fc80e25bf4881feececf1a0b1155c6e251e4a`; 11 rows.

Namespace: `KomorebiPaperIX.DworkCriterion`

`defect` (def; proof_source)

`defectIter` (def; proof_source)

`defectIter_succ` (theorem; proof_source)

`defectIter_zero` (theorem; proof_source)

`defect_is_shifted_difference` (theorem; proof_source)

`defect_packet_extraction` (theorem; proof_source)

`flat` (def; proof_source)

`packed` (def; proof_source)

`packed_multiple` (theorem; proof_source)

`packet_difference` (theorem; proof_source)

`saturated_bound_of_flat` (theorem; proof_source)

DworkWindowExamples

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/DworkWindowExamples.lean`.

SHA-256: `daa9a6e4d11480fee843d6768568aa0fde3ca163fa430479e93277233d24f46c`; 8 rows.

Namespace: `KomorebiPaperIX.DworkWindowExamples`

`defect_cubic_constant` (theorem; positive_control)

`derivation_rhs_constant` (theorem; positive_control)

`modulus_three_all_addresses` (theorem; positive_control)

`normalized_defect_not_derivation` (theorem; positive_control)

`window_p3_j2_coefficients` (theorem; positive_control)

`window_second_difference` (theorem; positive_control)

`window_second_difference_bound` (theorem; positive_control)

`window_second_difference_valuation` (theorem; positive_control)

DyadicExamples

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/DyadicExamples.lean`.

SHA-256: `13b9524deedd38e67666626673176a95699632fe1715ef420d76a109c91b1ef8`; 14 rows.

Namespace: `KomorebiPaperIX.DyadicExamples`

`beta_four` (theorem; positive_control)

`beta_one` (theorem; positive_control)

`beta_three` (theorem; positive_control)

`beta_two` (theorem; positive_control)

`manuscript_error_three` (theorem; positive_control)

`manuscript_error_three_valuation` (theorem; positive_control)

`manuscript_error_zero` (theorem; positive_control)

`manuscript_error_zero_valuation` (theorem; positive_control)

`manuscript_terms` (theorem; positive_control)

`manuscript_truncations` (theorem; positive_control)

`ordinary_coefficient_not_exponential` (theorem; positive_control)

`scale_factor_cannot_be_omitted` (theorem; positive_control)

`source_anchor_values` (theorem; positive_control)

`zero_row_all_depths` (theorem; positive_control)

DyadicFourCongruences

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/DyadicFourCongruences.lean`.

SHA-256: `8ee06fbfedc303d1a98a287f6edbadb9a5500593e45649e7b3fa0d0bf6f88b4d`; 18 rows.

Namespace: `KomorebiPaperIX.DyadicFourCongruences`

`base_step_bound` (theorem; proof_source)

`decode_packet` (theorem; proof_source)

`dyadic_period` (theorem; proof_source)

`dyadic_period_precision` (theorem; proof_source)

`lifted_flat` (theorem; proof_source)

`normalizedLift` (def; proof_source)

`normalizedLift_coe` (theorem; proof_source)

`normalized_bound` (theorem; proof_source)

`normalized_difference_coe` (theorem; proof_source)

`normalized_precision` (theorem; proof_source)

`packet_integral_series` (theorem; proof_source)

`packet_support` (theorem; proof_source)

`unnormalizedLift` (def; proof_source)

`unnormalizedLift_coe` (theorem; proof_source)

`unnormalized_bound` (theorem; proof_source)

`unnormalized_difference_coe` (theorem; proof_source)

`unnormalized_integral` (theorem; proof_source)

`unnormalized_precision` (theorem; proof_source)

DyadicFourDwork

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/DyadicFourDwork.lean`.

SHA-256: `4d9a9782f70e31b2ce842deea60414412d2e907f6904520fb8e98e5958b40138`; 15 rows.

Namespace: `KomorebiPaperIX.DyadicFourDwork`

`B4` (abbrev; proof_source)

`CoeffFlat4` (def; proof_source)

`Flat4` (def; proof_source)

`actual_packet_flat` (theorem; proof_source)

`decode_B4` (theorem; proof_source)

`decode_B4_iter` (theorem; proof_source)

`defect_coe` (theorem; proof_source)

`flat4_encode_iff` (theorem; proof_source)

`flat4_reciprocal` (theorem; proof_source)

`packed_flat_of_kummer` (theorem; proof_source)

`packet_is_reciprocal` (theorem; proof_source)

`sourceSeries` (def; proof_source)

`sourceSeries_constant` (theorem; proof_source)

`sourceSeries_encoding` (theorem; proof_source)

`sourceSeries_flat` (theorem; proof_source)

DyadicFourKernel

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/DyadicFourKernel.lean`.

SHA-256: `504a069c490a2b005fe5e8e585e283226fdd8b963b09884aa322cc979296db59`; 10 rows.

Namespace: `KomorebiPaperIX.DyadicFourKernel`

`affineUnit` (def; proof_source)

`affineUnit_coe` (theorem; proof_source)

`affine_isUnit` (theorem; proof_source)

`invAffine` (def; proof_source)

`invAffine_cancel` (theorem; proof_source)

`invAffine_coe` (theorem; proof_source)

`invAffine_kummer` (theorem; proof_source)

`sourcePacket` (def; proof_source)

`source_coe` (theorem; proof_source)

`source_kummer` (theorem; proof_source)

DyadicIntegrality

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/DyadicIntegrality.lean`.

SHA-256: `513898a8b2ea87ed359a3bfaee2851c3da486e24f51b80a26e494ff485119eac`; 29 rows.

Namespace: `KomorebiPaperIX.Dyadic`

`beta` (def; proof_source)

`betaPadic` (def; proof_source)

`betaPadic_coe` (theorem; proof_source)

`beta_integral` (theorem; proof_source)

`beta_recurrence` (theorem; proof_source)

`beta_zero` (theorem; proof_source)

`integralRationals` (def; proof_source)

`integral_iff` (theorem; proof_source)

`integral_iff_completed_norm` (theorem; proof_source)

`integral_of_valuation` (theorem; proof_source)

`inverse_coefficients_in_subring` (theorem; proof_source)

`kernel` (def; proof_source)

`kernelCoeff` (def; proof_source)

`kernel_coefficient_integral` (theorem; proof_source)

`kernel_coefficient_valuation` (theorem; proof_source)

`kernel_constant` (theorem; proof_source)

`sourceEven` (def; proof_source)

`sourcePadic` (def; proof_source)

`sourcePadic_coe` (theorem; proof_source)

`sourcePadic_isUnit` (theorem; proof_source)

`source_denominator_valuation` (theorem; proof_source)

`source_integral` (theorem; proof_source)

`source_nonzero` (theorem; proof_source)

`source_norm_one` (theorem; proof_source)

`source_numerator_nonzero` (theorem; proof_source)

`source_numerator_valuation` (theorem; proof_source)

`source_valuation` (theorem; proof_source)

`toPadicInt` (def; proof_source)

`toPadicInt_coe` (theorem; proof_source)

DyadicReconstruction

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/DyadicReconstruction.lean`.

SHA-256: `5c6d7f50a2fe03f186c37410e9cfdfebc88fd216557f4d180e6f8bb29eac4f0a`; 14 rows.

Namespace: `KomorebiPaperIX.DyadicReconstruction`

`dyadic_reconstruction` (theorem; proof_source)

`dyadic_reconstruction_even_degree` (theorem; proof_source)

`dyadic_reconstruction_padic` (theorem; proof_source)

`dyadic_supercongruence` (theorem; proof_source)

`falling_factorial_fraction` (theorem; proof_source)

`full_depth_exact` (theorem; proof_source)

`full_depth_exact_padic` (theorem; proof_source)

`normalized_error_integral` (theorem; proof_source)

`summand` (def; proof_source)

`summandPadic` (def; proof_source)

`summandPadic_coe` (theorem; proof_source)

`truncation` (def; proof_source)

`truncationPadic` (def; proof_source)

`truncationPadic_coe` (theorem; proof_source)

DyadicSeries

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/DyadicSeries.lean`.

SHA-256: `ab56451fee07b38657f9143d81e0e520ce74a31510f706ee9d9f27297849c998`; 15 rows.

Namespace: `KomorebiPaperIX.DyadicSeries`

`beta_original_generating_function` (theorem; proof_source)

`cosh_product_identity` (theorem; proof_source)

`dyadic_reconstruction_series` (theorem; proof_source)

`fullKernel` (def; proof_source)

`inverseSeries` (def; proof_source)

`inverse_even_coefficient` (theorem; proof_source)

`inverse_odd_coefficient` (theorem; proof_source)

`kernel_shift_identity` (theorem; proof_source)

`sinhAt` (def; proof_source)

`sinh_square_identity` (theorem; proof_source)

`sourceSeries` (def; proof_source)

`source_even_coefficient` (theorem; proof_source)

`source_odd_coefficient` (theorem; proof_source)

`source_shift_identity` (theorem; proof_source)

`source_times_cosh` (theorem; proof_source)

DyadicUnits

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/DyadicUnits.lean`.

SHA-256: `8d55fea825056a646ec3ee1523ae5bfcba20cf41416695bc43737d5309645b95`; 14 rows.

Namespace: `KomorebiPaperIX.DyadicUnits`

`decomposition_unique` (theorem; proof_source)

`dyadic_unit_decomposition` (theorem; proof_source)

`four_unit_cases` (theorem; proof_source)

`modFour` (def; proof_source)

`modFourUnits` (def; proof_source)

`principal` (def; proof_source)

`principal_mod_four` (theorem; proof_source)

`sign` (def; proof_source)

`sign_cases` (theorem; proof_source)

`sign_invisible_mod_two` (theorem; proof_source)

`sign_mod_four` (theorem; proof_source)

`sign_square` (theorem; proof_source)

`sign_unique` (theorem; proof_source)

`signs_distinct_mod_four` (theorem; proof_source)

EGFCoordinates

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/EGFCoordinates.lean`.

SHA-256: `cba2da32413923f441e9491a8f6ef5c53eddafcd5c5e797abd4a36516e8ca05e`; 27 rows.

Namespace: `KomorebiPaperIX.EGF`

`coordinateEquiv` (def; proof_source)

`coordinate_forward` (theorem; proof_source)

`coordinate_inverse` (theorem; proof_source)

`decode` (def; proof_source)

`decode_add` (theorem; proof_source)

`decode_encode` (theorem; proof_source)

`decode_zero` (theorem; proof_source)

`egf_product` (theorem; proof_source)

`encode` (def; proof_source)

`encode_add` (theorem; proof_source)

`encode_coeff` (theorem; proof_source)

`encode_decode` (theorem; proof_source)

`encode_zero` (theorem; proof_source)

`inverse_constant_one` (theorem; proof_source)

`matrixEntry` (def; proof_source)

`matrix_above_diagonal` (theorem; proof_source)

`matrix_diagonal` (theorem; proof_source)

`transfer` (def; proof_source)

`transfer_add` (theorem; proof_source)

`transfer_comp` (theorem; proof_source)

`transfer_one` (theorem; proof_source)

`transfer_sum` (theorem; proof_source)

`transfer_zero` (theorem; proof_source)

`transportedPacket` (def; proof_source)

`transported_idempotent` (theorem; proof_source)

`transported_orthogonal` (theorem; proof_source)

`transported_reconstruction` (theorem; proof_source)

EulerCoefficients

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/EulerCoefficients.lean`.

SHA-256: `0456c97fcefd607b7add2abd9ca8e2a43fafa2a7a5c8cd8cbf3186ddad1d245c`; 18 rows.

Namespace: `KomorebiPaperIX.EulerCoefficients`

`clearing_even_one` (theorem; proof_source)

`cosh_coeff` (theorem; proof_source)

`euler` (def; proof_source)

`eulerEven` (def; proof_source)

`eulerEven_cast` (theorem; proof_source)

`euler_egf` (theorem; proof_source)

`euler_even` (theorem; proof_source)

`euler_odd` (theorem; proof_source)

`euler_series_expansion` (theorem; proof_source)

`euler_zero` (theorem; proof_source)

`inverseCoeff` (def; proof_source)

`inverseEven` (def; proof_source)

`inverse_coeff_cast` (theorem; proof_source)

`inverse_even_egf` (theorem; proof_source)

`inverse_odd_zero` (theorem; proof_source)

`inverse_zero` (theorem; proof_source)

`original_even_section` (theorem; proof_source)

`sech_square_coeff` (theorem; proof_source)

EulerDefect

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/EulerDefect.lean`.

SHA-256: `460a5627629e76a6ca2c42513293904b600e2b879d1656b04ebbea1bcc5b6846`; 6 rows.

Namespace: `KomorebiPaperIX.EulerDefect`

`correction` (def; proof_source)

`defect_support` (theorem; proof_source)

`eulerOperator` (def; proof_source)

`euler_pivot_unit` (theorem; proof_source)

`flat_of_euler_flat` (theorem; proof_source)

`iterated_commutator` (theorem; proof_source)

EulerSeriesInverse

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/EulerSeriesInverse.lean`.

SHA-256: `1d7e58820165bbf636774c8a1362209ffdebdf9d8e55e52442f8c72038d571d0`; 25 rows.

Namespace: `KomorebiPaperIX.EulerSeriesInverse`

`Flat` (def; proof_source)

`Supported` (def; proof_source)

`encode_lift` (theorem; proof_source)

`euler` (def; proof_source)

`euler_coeff` (theorem; proof_source)

`euler_lift_encoding` (theorem; proof_source)

`euler_solve` (theorem; proof_source)

`flat_add` (theorem; proof_source)

`flat_constant` (theorem; proof_source)

`flat_mul` (theorem; proof_source)

`flat_sub` (theorem; proof_source)

`flat_zero` (theorem; proof_source)

`lift` (def; proof_source)

`solve` (def; proof_source)

`solve_coeff` (theorem; proof_source)

`solve_constant_zero` (theorem; proof_source)

`solve_euler` (theorem; proof_source)

`solve_flat` (theorem; proof_source)

`solve_integral` (theorem; proof_source)

`solve_supported` (theorem; proof_source)

`supported_add` (theorem; proof_source)

`supported_constant` (theorem; proof_source)

`supported_mul` (theorem; proof_source)

`supported_sub` (theorem; proof_source)

`supported_zero` (theorem; proof_source)

ExponentialSection

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/ExponentialSection.lean`.

SHA-256: `be0887adc23c60fe705ab0ddfe06eec999116de56035245a9f5a4ae82fbfe80e`; 8 rows.

Namespace: `KomorebiPaperIX.ExponentialSection`

`expandedPacket` (def; proof_source)

`expandedSource` (def; proof_source)

`originalSection` (def; proof_source)

`original_section_factorization` (theorem; proof_source)

`original_section_inverse_identity` (theorem; proof_source)

`packet_coefficient_at_address` (theorem; proof_source)

`source_coefficient_at_address` (theorem; proof_source)

`source_coefficient_off_address` (theorem; proof_source)

FibonacciPackets

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/FibonacciPackets.lean`.

SHA-256: `a2663018262ad9cdb601389d8b48de4d9b6f4fdd5329522c7f219eeb64d7f6bc`; 22 rows.

Namespace: `KomorebiPaperIX.FibonacciPackets`

`companion` (def; proof_source)

`companion_power_add_two` (theorem; proof_source)

`companion_square` (theorem; proof_source)

`det_companion` (theorem; proof_source)

`det_power` (theorem; proof_source)

`fibonacci_even_step` (theorem; proof_source)

`fibonacci_odd_step` (theorem; proof_source)

`fibonacci_packet_recurrence` (theorem; proof_source)

`fibonacci_packet_unique` (theorem; proof_source)

`fibonacci_state_packet_recurrence` (theorem; proof_source)

`initial` (def; proof_source)

`jump_charpoly` (theorem; proof_source)

`jump_coeff_one` (theorem; proof_source)

`jump_coeff_zero` (theorem; proof_source)

`lucas` (def; proof_source)

`lucas_is_fibonacci_combination` (theorem; proof_source)

`packet_is_fibonacci` (theorem; proof_source)

`state` (def; proof_source)

`state_is_orbit` (theorem; proof_source)

`state_step` (theorem; proof_source)

`state_zero` (theorem; proof_source)

`trace_power_is_lucas` (theorem; proof_source)

FilteredReconstruction

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/FilteredReconstruction.lean`.

SHA-256: `44d692ff84dbd425ce72b1aee131f119269e99da000d5efe073ad2fb03f9cf85`; 8 rows.

Namespace: `KomorebiPaperIX.Filtered`

`even_weight_tail` (theorem; proof_source)

`filtered_triangular_reconstruction` (theorem; proof_source)

`omitted_tail_dvd` (theorem; proof_source)

`padic_filtered_reconstruction` (theorem; proof_source)

`power_dvd_weightPower` (theorem; proof_source)

`weightPower` (def; proof_source)

`weightPower_nat` (theorem; proof_source)

`weightPower_top` (theorem; proof_source)

FiniteArithmeticExamples

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/FiniteArithmeticExamples.lean`.

SHA-256: `0c3c7d513a811b8493fc3bcdd6ede9a1af5429fb780124ba744d3c0e52d0e2b7`; 15 rows.

Namespace: `KomorebiPaperIX.FiniteArithmeticExamples`

`composite_fermat_failure` (theorem; positive_control)

`dyadic_three_principal` (theorem; positive_control)

`dyadic_three_reconstruction` (theorem; positive_control)

`dyadic_three_sign` (theorem; positive_control)

`noncoprime_components_do_not_separate` (theorem; positive_control)

`ordinary_residue_is_not_lifted_root` (theorem; positive_control)

`outside_window_denominator` (theorem; positive_control)

`outside_window_not_integral` (theorem; positive_control)

`principal_two_mod_nine` (theorem; positive_control)

`reduced_first_row` (theorem; positive_control)

`reduced_second_row` (theorem; positive_control)

`standard_window_example` (theorem; positive_control)

`teichmuller_two_mod_nine` (theorem; positive_control)

`threeModEight` (def; positive_control)

`twoModNine` (def; positive_control)

FiniteMahler

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/FiniteMahler.lean`.

SHA-256: `9fa96a23456f589891bb58cff4f22075375a4d3fe6a00db2e5236ddcbf3facec`; 22 rows.

Namespace: `KomorebiPaperIX.Mahler`

`coefficient` (def; proof_source)

`coefficient_difference` (theorem; proof_source)

`coefficient_eq_of_prefix` (theorem; proof_source)

`coefficient_interpolant` (theorem; proof_source)

`coefficient_zero` (theorem; proof_source)

`difference` (def; proof_source)

`difference_add` (theorem; proof_source)

`difference_eq_of_stencil` (theorem; proof_source)

`difference_explicit` (theorem; proof_source)

`difference_from_coefficients` (theorem; proof_source)

`difference_mul_const` (theorem; proof_source)

`difference_one` (theorem; proof_source)

`difference_succ` (theorem; proof_source)

`difference_sum` (theorem; proof_source)

`difference_zero` (theorem; proof_source)

`finite_mahler_reconstruction` (theorem; proof_source)

`finite_sample_difference` (theorem; proof_source)

`interpolant` (def; proof_source)

`interpolant_coefficients_unique` (theorem; proof_source)

`interpolant_difference` (theorem; proof_source)

`interpolant_high_difference` (theorem; proof_source)

`interpolant_reproduces` (theorem; proof_source)

FiniteUnits

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/FiniteUnits.lean`.

SHA-256: `935bf62202a5c0e0f0af151ac1e3651281e1abb12af2d9307ea60d7547aefb4e`; 17 rows.

Namespace: `KomorebiPaperIX.FiniteUnits`

`decomposition_unique` (theorem; proof_source)

`finite_unit_decomposition` (theorem; proof_source)

`omega` (def; proof_source)

`omega_mul` (theorem; proof_source)

`omega_residue` (theorem; proof_source)

`omega_root` (theorem; proof_source)

`omega_unique` (theorem; proof_source)

`power_lift_congruence` (theorem; proof_source)

`principal` (def; proof_source)

`principal_mul` (theorem; proof_source)

`principal_residue` (theorem; proof_source)

`residue` (def; proof_source)

`residueUnits` (def; proof_source)

`residue_zero_iff` (theorem; proof_source)

`root_fixed_by_prime` (theorem; proof_source)

`root_fixed_by_prime_power` (theorem; proof_source)

`root_unique` (theorem; proof_source)

FrobeniusSquare

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/FrobeniusSquare.lean`.

SHA-256: `50716048ec83db6ceaee9a9f7d99a194aa527673acc6bc7be381cee052bcd370`; 2 rows.

Namespace: `KomorebiPaperIX.FrobeniusSquare`

`normalized_defect_integral` (theorem; proof_source)

`power_defect_divisible` (theorem; proof_source)

GlobalEulerBoundary

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/GlobalEulerBoundary.lean`.

SHA-256: `5fb8c720fbf51e90c2de2476c7b3c5977a25495bb6a5594bd8819949e97f0417`; 8 rows.

Namespace: `KomorebiPaperIX.GlobalEulerBoundary`

`factorial_loss_transfer` (theorem; proof_source)

`lifted_difference_coe` (theorem; proof_source)

`precision` (def; proof_source)

`precision_zero` (theorem; proof_source)

`standard_window_all_orders_mod_p` (theorem; proof_source)

`standard_window_all_orders_order` (theorem; proof_source)

`standard_window_loss_transfer` (theorem; proof_source)

`unnormalized_difference` (theorem; proof_source)

HurwitzConvolution

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/HurwitzConvolution.lean`.

SHA-256: `6047f1dac605fe54aab71b846cd0ac03af8e124d707e28f530dfbdfd987b2eff`; 16 rows.

Namespace: `KomorebiPaperIX.Hurwitz`

`convolution` (def; proof_source)

`convolution_add_left` (theorem; proof_source)

`convolution_add_right` (theorem; proof_source)

`convolution_at_zero` (theorem; proof_source)

`convolution_left_eq_zero` (theorem; proof_source)

`convolution_sub_left` (theorem; proof_source)

`convolution_sub_right` (theorem; proof_source)

`convolution_zero_left` (theorem; proof_source)

`convolution_zero_right` (theorem; proof_source)

`oneSeries` (def; proof_source)

`prime_shift_convolution` (theorem; proof_source)

`shift` (def; proof_source)

`shift_add` (theorem; proof_source)

`shift_convolution` (theorem; proof_source)

`shift_one_convolution` (theorem; proof_source)

`shift_zero` (theorem; proof_source)

HurwitzDworkClosure

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/HurwitzDworkClosure.lean`.

SHA-256: `b225c928c4c25c9f870a5b38d7f3fb3c7b2eaa619d88df4b208923f7314585d6`; 17 rows.

Namespace: `KomorebiPaperIX.HurwitzDworkClosure`

`decode_B` (theorem; proof_source)

`decode_B_iterate` (theorem; proof_source)

`decode_deriv` (theorem; proof_source)

`decode_derivative` (theorem; proof_source)

`decode_encodeIntegral` (theorem; proof_source)

`decode_one` (theorem; proof_source)

`defectIter_coe` (theorem; proof_source)

`encodeIntegral` (def; proof_source)

`flat_encode_iff` (theorem; proof_source)

`flat_product` (theorem; proof_source)

`flat_reciprocal` (theorem; proof_source)

`integralSeries` (def; proof_source)

`integral_derivative` (theorem; proof_source)

`integral_encode` (theorem; proof_source)

`integral_inverse` (theorem; proof_source)

`inverse_decode_step` (theorem; proof_source)

`normalized_mem_iff` (theorem; proof_source)

HurwitzInverse

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/HurwitzInverse.lean`.

SHA-256: `ef1bc0132dd3082f7bd5e60533178e7ea3ef33c83e61ec31637cfc4c07260e06`; 7 rows.

Namespace: `KomorebiPaperIX.HurwitzInverse`

`inverse` (def; proof_source)

`inverse_convolution` (theorem; proof_source)

`inverse_prime_difference` (theorem; proof_source)

`inverse_step` (theorem; proof_source)

`inverse_support` (theorem; proof_source)

`inverse_zero` (theorem; proof_source)

`sum_on_multiples` (theorem; proof_source)

KummerSequences

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/KummerSequences.lean`.

SHA-256: `602180b22312175b93fd3faef6b91299dddb27fd64de2af40a899a4858d8c946`; 7 rows.

Namespace: `KomorebiPaperIX.KummerSequences`

`Kummer` (def; proof_source)

`affine_kummer` (theorem; proof_source)

`constant_kummer` (theorem; proof_source)

`difference_constant` (theorem; proof_source)

`difference_product` (theorem; proof_source)

`kummer_inverse` (theorem; proof_source)

`kummer_mul` (theorem; proof_source)

MahlerExamples

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/MahlerExamples.lean`.

SHA-256: `e2e4fcfd857b57a888698d3f0d3199d9f1f9ed2d2c3129c7ef9027d7e5f650b2`; 15 rows.

Namespace: `KomorebiPaperIX.MahlerExamples`

`constant_mode_infinite_valuation` (theorem; positive_control)

`constant_mode_zero` (theorem; positive_control)

`exampleWeight` (def; positive_control)

`finite_sample_boundary` (theorem; positive_control)

`nonunit_mode_counterexample` (theorem; positive_control)

`square_coefficients` (theorem; positive_control)

`square_reconstructed` (theorem; positive_control)

`unit_mode_sharp_integer_example` (theorem; positive_control)

`unrestricted_interpolant_difference_false` (theorem; positive_control)

`weightedCoefficients` (def; positive_control)

`weighted_example_bound` (theorem; positive_control)

`weighted_example_difference` (theorem; positive_control)

`weighted_example_padic_difference` (theorem; positive_control)

`weighted_example_padic_valuation` (theorem; positive_control)

`weighted_example_sharp` (theorem; positive_control)

NishibuchiCompletion

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/NishibuchiCompletion.lean`.

SHA-256: `2abaf31d61dd1fbf3db8998f6f48b854ed3ecc497ed1d58b2da8aaa21188c99e`; 13 rows.

Namespace: `KomorebiPaperIX.NishibuchiCompletion`

`allowed_parameters` (theorem; proof_source)

`conjecture_3_9` (theorem; proof_source)

`displacementFactor` (def; proof_source)

`divisor_period` (theorem; proof_source)

`full_integrality` (theorem; proof_source)

`full_period_degree` (theorem; proof_source)

`full_period_packet` (theorem; proof_source)

`generalizedEuler` (def; proof_source)

`generalizedEuler_at_packet` (theorem; proof_source)

`modulus_divides_displacement` (theorem; proof_source)

`original_degree_shift` (theorem; proof_source)

`original_generating_identity` (theorem; proof_source)

`packet_step_factor` (theorem; proof_source)

NormalizationBoundary

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/NormalizationBoundary.lean`.

SHA-256: `4881988d3339abce4b1077f05fcef68a9e65853008dc9316c6f2df0e95fda768`; 12 rows.

Namespace: `KomorebiPaperIX.Normalization`

`counterexample_parameters` (theorem; proof_source)

`difference_nonzero` (theorem; proof_source)

`difference_valuation` (theorem; proof_source)

`factorial_loss_is_sharp` (theorem; proof_source)

`factorial_valuation` (theorem; proof_source)

`false_unrestricted_bound` (theorem; proof_source)

`normalized_difference` (theorem; proof_source)

`normalized_first` (theorem; proof_source)

`normalized_second` (theorem; proof_source)

`universal_no_loss_refuted` (theorem; proof_source)

`unnormalized_difference` (theorem; proof_source)

`unnormalized_difference_valuation` (theorem; proof_source)

OddEulerExamples

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/OddEulerExamples.lean`.

SHA-256: `ab3b62d33aa2edfacac0745ee9c2bf0177bc18b3c2140ef73a102b155f1ad7af`; 11 rows.

Namespace: `KomorebiPaperIX.OddEulerExamples`

`coefficient_four` (theorem; positive_control)

`coefficient_one` (theorem; positive_control)

`coefficient_three` (theorem; positive_control)

`coefficient_two` (theorem; positive_control)

`euler_anchor_values` (theorem; positive_control)

`even_length_reconstruction_fails` (theorem; positive_control)

`inverse_anchor_values` (theorem; positive_control)

`manuscript_error` (theorem; positive_control)

`manuscript_error_sharp` (theorem; positive_control)

`manuscript_truncation` (theorem; positive_control)

`source_anchor_values` (theorem; positive_control)

OddEulerReconstruction

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/OddEulerReconstruction.lean`.

SHA-256: `a462465cedc9ed789ce10a89d0a6f53e2bf6059b215f066ad2dfc6059637b699`; 18 rows.

Namespace: `KomorebiPaperIX.OddEuler`

`decode_even_source` (theorem; proof_source)

`decode_rescale` (theorem; proof_source)

`filter_range_le` (theorem; proof_source)

`full_depth_exact` (theorem; proof_source)

`odd_euler_reconstruction` (theorem; proof_source)

`odd_euler_reconstruction_zigzag` (theorem; proof_source)

`odd_euler_supercongruence` (theorem; proof_source)

`odd_prime_supercongruence` (theorem; proof_source)

`odd_reconstruction_series` (theorem; proof_source)

`reconstruction_all_indices` (theorem; proof_source)

`rowEntry` (def; proof_source)

`row_above_diagonal` (theorem; proof_source)

`row_action` (theorem; proof_source)

`row_diagonal` (theorem; proof_source)

`row_weight` (theorem; proof_source)

`scaled_sech_cosh` (theorem; proof_source)

`sum_even_terms` (theorem; proof_source)

`truncation` (def; proof_source)

OddPowerSeries

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/OddPowerSeries.lean`.

SHA-256: `4b1c80671d1c701ffa6f8f28ef6cdff16fa05c0e24766df5fe61cf73a35efb1f`; 24 rows.

Namespace: `KomorebiPaperIX.OddPower`

`coshAt` (def; proof_source)

`cosh_constant` (theorem; proof_source)

`cosh_mul_sech` (theorem; proof_source)

`cosh_neg` (theorem; proof_source)

`cosh_square` (theorem; proof_source)

`evenPart` (def; proof_source)

`evenPart_coeff` (theorem; proof_source)

`expAt` (def; proof_source)

`expAt_coeff` (theorem; proof_source)

`expAt_constant` (theorem; proof_source)

`expAt_mul` (theorem; proof_source)

`expAt_zero` (theorem; proof_source)

`odd_source_egf` (theorem; proof_source)

`odd_source_even_egf` (theorem; proof_source)

`rescale_constant` (theorem; proof_source)

`rescale_cosh` (theorem; proof_source)

`rescale_expAt` (theorem; proof_source)

`rescale_sech_neg` (theorem; proof_source)

`sech` (def; proof_source)

`sech_mul_cosh` (theorem; proof_source)

`source` (def; proof_source)

`sourceSeries` (def; proof_source)

`source_as_exponentials` (theorem; proof_source)

`source_times_cosh` (theorem; proof_source)

PadicDifferenceBounds

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/PadicDifferenceBounds.lean`.

SHA-256: `aaf74728af8a0e97badebe3365af4545093c76d9d5af560ee5187f2e803ec9ba`; 14 rows.

Namespace: `KomorebiPaperIX.PadicBounds`

`coefficient_tail_bound` (theorem; proof_source)

`difference_dvd_of_coefficients` (theorem; proof_source)

`interpolant_weighted_bound` (theorem; proof_source)

`nat_le_order_iff` (theorem; proof_source)

`order` (def; proof_source)

`order_add_lower` (theorem; proof_source)

`order_mul` (theorem; proof_source)

`order_natCast` (theorem; proof_source)

`order_nonzero` (theorem; proof_source)

`order_sum_lower` (theorem; proof_source)

`order_zero` (theorem; proof_source)

`weightedBound` (def; proof_source)

`weighted_difference_bound` (theorem; proof_source)

`weighted_difference_divisibility` (theorem; proof_source)

PeriodExamples

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/PeriodExamples.lean`.

SHA-256: `70655437a706347237eb9e8002f0859f6f1498af13479eb2b3c05b22fc97e3c3`; 12 rows.

Namespace: `KomorebiPaperIX.PeriodExamples`

`c60_difference` (theorem; positive_control)

`c60_first` (theorem; positive_control)

`c60_second` (theorem; positive_control)

`first_difference_precision_is_one` (theorem; positive_control)

`nondivisor_period_fails` (theorem; positive_control)

`nondivisor_reduced_third` (theorem; positive_control)

`p3_m2_positive_tail` (theorem; positive_control)

`p5_m2_positive_tail` (theorem; positive_control)

`period_can_fail_at_zero` (theorem; positive_control)

`reduced_one` (theorem; positive_control)

`reduced_two` (theorem; positive_control)

`zero_has_every_precision` (theorem; positive_control)

PeriodFromDifferences

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/PeriodFromDifferences.lean`.

SHA-256: `7434bbb6d142befe3708a7678248ccc48ced0b25d8beb8c6bef95dd460f468c4`; 4 rows.

Namespace: `KomorebiPaperIX.PeriodFromDifferences`

`binomial_precision` (theorem; proof_source)

`displacement_expansion` (theorem; proof_source)

`prime_power_period` (theorem; proof_source)

`valuation_lt` (theorem; proof_source)

PrimeBlockPolynomial

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/PrimeBlockPolynomial.lean`.

SHA-256: `4fd07e046a0bc4c8d0d0eef91a1780477115d4c957902139343abf8018c9ca73`; 10 rows.

Namespace: `KomorebiPaperIX.PrimeBlockPolynomial`

`block_mem` (theorem; proof_source)

`falling` (def; proof_source)

`fallingAt` (def; proof_source)

`fallingAt_add` (theorem; proof_source)

`falling_blocks_mem` (theorem; proof_source)

`falling_defect_divisible` (theorem; proof_source)

`falling_eval` (theorem; proof_source)

`falling_factorial_divisibility` (theorem; proof_source)

`falling_mem` (theorem; proof_source)

`prime_field_falling` (theorem; proof_source)

RatioRiccati

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/RatioRiccati.lean`.

SHA-256: `feaa57b9c8c751f1dba6f72f7a09d17013a5f0cf6052a4516ae1a504a5db5e88`; 9 rows.

Namespace: `KomorebiPaperIX.RatioRiccati`

`centered_equation` (theorem; proof_source)

`next_ratio_flat` (theorem; proof_source)

`original_derivative` (theorem; proof_source)

`original_nonzero` (theorem; proof_source)

`ratio` (def; proof_source)

`ratio_p_support` (theorem; proof_source)

`ratio_riccati` (theorem; proof_source)

`ratio_riccati_padic` (theorem; proof_source)

`ratio_zero` (theorem; proof_source)

RatioSeriesSupport

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/RatioSeriesSupport.lean`.

SHA-256: `dd94ae985ab933d532ad1df434c6457c8a02811a1bc19800792c51556b160b56`; 7 rows.

Namespace: `KomorebiPaperIX.RatioSeriesSupport`

`expanded_inverse` (theorem; proof_source)

`ratioCoefficient` (def; proof_source)

`ratioCoefficient_zero` (theorem; proof_source)

`ratio_expansion` (theorem; proof_source)

`ratio_support` (theorem; proof_source)

`singular_ratio_integral` (theorem; proof_source)

`singular_ratio_precision` (theorem; proof_source)

RecurrenceExamples

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/RecurrenceExamples.lean`.

SHA-256: `8282676bd72fea6471c1b76b114c7c76d6100cd94db01355235bd0ebfa1d936c`; 15 rows.

Namespace: `KomorebiPaperIX.RecurrenceExamples`

`empty_state_recurrence` (theorem; positive_control)

`even_step_wrong_sign` (theorem; positive_control)

`identity_packet_constant` (theorem; positive_control)

`identity_packet_first_order` (theorem; positive_control)

`lucas_anchor_values` (theorem; positive_control)

`manuscript_sequence` (theorem; positive_control)

`manuscript_step_three` (theorem; positive_control)

`nilpotent` (def; positive_control)

`nilpotent_charpoly` (theorem; positive_control)

`nilpotent_packet_zero` (theorem; positive_control)

`nilpotent_square` (theorem; positive_control)

`step_one` (theorem; positive_control)

`step_two` (theorem; positive_control)

`wrong_jump_matrix` (theorem; positive_control)

`zero_step` (theorem; positive_control)

ResiduePackets

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/ResiduePackets.lean`.

SHA-256: `337ff6013b30e09336b0e0e8cadb092864a5f5a3b6ce18c801130c5b7afc0b73`; 10 rows.

Namespace: `KomorebiPaperIX.ResiduePackets`

`packet` (def; proof_source)

`packet_apply` (theorem; proof_source)

`packet_idempotent` (theorem; proof_source)

`packet_orthogonal` (theorem; proof_source)

`packet_reconstruction` (theorem; proof_source)

`packet_refinement` (theorem; proof_source)

`seriesPacket` (def; proof_source)

`seriesPacket_coeff` (theorem; proof_source)

`series_packet_reconstruction` (theorem; proof_source)

`series_packet_refinement` (theorem; proof_source)

ResidueZeroDwork

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/ResidueZeroDwork.lean`.

SHA-256: `bb1209803a3619553c0b17013332ce8e167491bba730335bfe9dec96680439e0`; 11 rows.

Namespace: `KomorebiPaperIX.ResidueZeroDwork`

`encode_convolution` (theorem; proof_source)

`sectionCoefficients` (def; proof_source)

`section_defect_zero` (theorem; proof_source)

`section_flat` (theorem; proof_source)

`section_zero` (theorem; proof_source)

`zeroPacket` (def; proof_source)

`zero_packet_coe` (theorem; proof_source)

`zero_packet_flat` (theorem; proof_source)

`zero_packet_global_bound` (theorem; proof_source)

`zero_packet_is_reciprocal` (theorem; proof_source)

`zero_packet_section_product` (theorem; proof_source)

RiccatiFlatPropagation

Source: baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/RiccatiFlatPropagation.lean.

SHA-256: bdb9adf5c152b1d2eafeb3e741c6ab1ba007d844d7e1266e08a930dc91f4c5fb; 10 rows.

Namespace: KomorebiPaperIX.RiccatiFlatPropagation

B_iterate_prefix (theorem; proof_source)

forcing (def; proof_source)

forcing_difference (theorem; proof_source)

picard (def; proof_source)

picard_constant (theorem; proof_source)

picard_flat (theorem; proof_source)

picard_matches (theorem; proof_source)

picard_supported (theorem; proof_source)

product_vanishing_prefix (theorem; proof_source)

riccati_solution_flat (theorem; proof_source)

SectionLogBridge

Source: baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/SectionLogBridge.lean.

SHA-256: 3b519fbb417aabac764fb101a31a5012a08faa64892c305978b9cf652c9477ff; 11 rows.

Namespace: KomorebiPaperIX.SectionLogBridge

coefficient_section (theorem; proof_source)

factorial_coeff_section (theorem; proof_source)

numerator_succ (theorem; proof_source)

numerator_zero (theorem; proof_source)

pullback_equation (theorem; proof_source)

pullback_recurrence (theorem; proof_source)

pullback_zero (theorem; proof_source)

sectionPullback (def; proof_source)

sectionSeries (def; proof_source)

section_derivative (theorem; proof_source)

section_weighted_integral (theorem; proof_source)

SeriesDifferenceBridge

Source: baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/SeriesDifferenceBridge.lean.

SHA-256: 8c48a5db9bda683b7885b98f04ebd5757621ae73a9153ea4255afae4775dfbac; 7 rows.

Namespace: KomorebiPaperIX.SeriesDifferenceBridge

decode_rationalMap (theorem; proof_source)

`packetLift` (def; proof_source)

`packetLift_coe` (theorem; proof_source)

`packet_divisibility` (theorem; proof_source)

`packet_encoding` (theorem; proof_source)

`packet_precision` (theorem; proof_source)

`rationalMap` (def; proof_source)

SingularAmiceExamples

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/SingularAmiceExamples.lean`.

SHA-256: `fa578ab72fd4b39a74866b6ffb6159b5b6a427c26ecfd5453cfde5dc77a146f3`; 10 rows.

Namespace: `KomorebiPaperIX.SingularAmiceExamples`

`normalized_section_isUnit` (theorem; positive_control)

`p3_singular_ratio_constant` (theorem; positive_control)

`p3_singular_ratio_weighted` (theorem; positive_control)

`positive_section_zero_constant` (theorem; positive_control)

`section3_bound_sharp` (theorem; positive_control)

`section3_coefficient_valuation` (theorem; positive_control)

`section3_first` (theorem; positive_control)

`section3_first_nonconstant` (theorem; positive_control)

`section3_second_two` (theorem; positive_control)

`section3_second_zero` (theorem; positive_control)

SingularNormalizedSection

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/SingularNormalizedSection.lean`.

SHA-256: `d3f900e7689f612e6acd0a6c78457a9232b420e26dd8657f563ff11913473e14`; 15 rows.

Namespace: `KomorebiPaperIX.SingularNormalizedSection`

`kernelPadic` (def; proof_source)

`kernel_constant` (theorem; proof_source)

`logUnit` (def; proof_source)

`logUnit_constant` (theorem; proof_source)

`log_factorization` (theorem; proof_source)

`mapped_original_factorization` (theorem; proof_source)

`normalizedUnit` (def; proof_source)

`normalized_coefficient` (theorem; proof_source)

`normalized_constant` (theorem; proof_source)

`prime_power_dvd_singular_factorial` (theorem; proof_source)

`sectionSeries_is_original` (theorem; proof_source)

section_factorization (theorem; proof_source)

weighted_normalized_singular (theorem; proof_source)

weighted_prime_logUnit (theorem; proof_source)

weighted_singular_leading (theorem; proof_source)

SingularRatioAmice

Source: baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/SingularRatioAmice.lean.

SHA-256: ddf3691149adc29c17f2a98e177a5d649c47e12c78dc45c2491f075fd4a143d0; 19 rows.

Namespace: KomorebiPaperIX.SingularRatioAmice

X_derivative_power (theorem; proof_source)

derivative_map_rat (theorem; proof_source)

normalized_log_derivative (theorem; proof_source)

paper_singular_amice_estimate (theorem; proof_source)

ratioPullback (def; proof_source)

ratioPullback_is_rational_map (theorem; proof_source)

ratioSeries (def; proof_source)

ratio_cleared (theorem; proof_source)

ratio_constant (theorem; proof_source)

ratio_pullback_cleared (theorem; proof_source)

ratio_pullback_formula (theorem; proof_source)

rationalRatioPullback (def; proof_source)

rational_pullback_constant (theorem; proof_source)

sectionPadic_as_subst (theorem; proof_source)

section_nonzero (theorem; proof_source)

section_theta (theorem; proof_source)

singular_ratio_weighted (theorem; proof_source)

singular_rational_coefficient_bound (theorem; proof_source)

valuation_of_scaled_integral (theorem; proof_source)

StandardWindow

Source: baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/StandardWindow.lean.

SHA-256: 5d9bba7b3ad262849f5a718d5d0916674a7620c45ec7c7a3b1e60a769159af1f; 6 rows.

Namespace: KomorebiPaperIX.StandardWindow

factorial_unit_window (theorem; proof_source)

factorial_valuation_zero (theorem; proof_source)

lucas_row_reduction (theorem; proof_source)

standard_diagonal_mod_one (theorem; proof_source)

`standard_diagonal_not_divisible` (theorem; proof_source)

`standard_diagonal_unit` (theorem; proof_source)

StandardWindowDwork

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/StandardWindowDwork.lean`.

SHA-256: `ddcef4f0da1cc65bb5e17fd752b5610acea5c0817eefdc4ca609634a092d28a4`; 15 rows.

Namespace: `KomorebiPaperIX.StandardWindowDwork`

`defectIter_mul_const` (theorem; proof_source)

`flat_mul_const` (theorem; proof_source)

`kernelCoefficients` (def; proof_source)

`kernel_coe` (theorem; proof_source)

`kernel_constant` (theorem; proof_source)

`kernel_encoding` (theorem; proof_source)

`kernel_euler_step` (theorem; proof_source)

`kernel_flat` (theorem; proof_source)

`kernel_support` (theorem; proof_source)

`kernel_zero_address` (theorem; proof_source)

`packet_encoding` (theorem; proof_source)

`standard_window_flat` (theorem; proof_source)

`standard_window_global_bound` (theorem; proof_source)

`standard_window_global_order` (theorem; proof_source)

`window_is_reciprocal` (theorem; proof_source)

StandardWindowPeriod

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/StandardWindowPeriod.lean`.

SHA-256: `29cedfb69eb21401800028d2939a7fef30e39086ace7bbe86080e0d48da78ea1`; 9 rows.

Namespace: `KomorebiPaperIX.WindowPeriod`

`inverse_off_support` (theorem; proof_source)

`inverse_on_support` (theorem; proof_source)

`multiple_period` (theorem; proof_source)

`sectionCoefficients` (def; proof_source)

`section_off_support` (theorem; proof_source)

`section_prime_shift` (theorem; proof_source)

`section_zero` (theorem; proof_source)

`standard_window_period_divisibility` (theorem; proof_source)

`uniform_standard_window_period` (theorem; proof_source)

StandardWindowPrecision

Source: baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/StandardWindowPrecision.lean.

SHA-256: 5c90b1298e01e43085409cd1476565e16cb71c8a0253e9cd44e6d61342e8c243; 5 rows.

Namespace: KomorebiPaperIX.StandardWindowPrecision

modulus_p_complete_family (theorem; proof_source)

normalized_window_precision (theorem; proof_source)

rational_precision_of_divisibility (theorem; proof_source)

residue_zero_complete_bound (theorem; proof_source)

unnormalized_window_precision (theorem; proof_source)

Theorem75

Source: baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/Theorem75.lean.

SHA-256: 4f18d608ec38c38f9773e2a58ed058595d523e3860944c65a78cc9e35da06a0a; 2 rows.

Namespace: KomorebiPaperIX.Theorem75

global_finite_difference (theorem; proof_source)

precision_at_threshold (theorem; proof_source)

ThetaWeighted

Source: baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/ThetaWeighted.lean.

SHA-256: 715a740572c34e784cbe239b83da0b26f6e80aafda4ed51b906255f68f760dd9; 30 rows.

Namespace: KomorebiPaperIX.ThetaWeighted

B_constant (theorem; proof_source)

B_iter_mem (theorem; proof_source)

B_mem (theorem; proof_source)

B_prefix (theorem; proof_source)

B_y_power (theorem; proof_source)

Good (def; proof_source)

X_binomial (theorem; proof_source)

deriv_mem (theorem; proof_source)

deriv_prefix (theorem; proof_source)

deriv_y_power (theorem; proof_source)

dividedGenerator (def; proof_source)

dividedGenerator_mem (theorem; proof_source)

divided_y_defect_mem (theorem; proof_source)

good_X (theorem; proof_source)

`good_add` (theorem; proof_source)

`good_const` (theorem; proof_source)

`good_dividedGenerator` (theorem; proof_source)

`good_monomial` (theorem; proof_source)

`good_mul` (theorem; proof_source)

`good_neg` (theorem; proof_source)

`good_pow` (theorem; proof_source)

`good_sum` (theorem; proof_source)

`good_y` (theorem; proof_source)

`middle_mem` (theorem; proof_source)

`monomial_factorization` (theorem; proof_source)

`theta` (def; proof_source)

`theta_apply` (theorem; proof_source)

`theta_coeff` (theorem; proof_source)

`theta_mem` (theorem; proof_source)

`theta_y_power` (theorem; proof_source)

Triangular

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/Triangular.lean`.

SHA-256: `296b4aed921d061cf5579b0a7c532da2ac324fb7229f3c24988cb6e8b1d96ac1`; 17 rows.

Namespace: `KomorebiPaperIX.Triangular`

`Integral` (def; proof_source)

`denominator` (def; proof_source)

`denominator_dvd` (theorem; proof_source)

`denominator_succ` (theorem; proof_source)

`denominator_zero` (theorem; proof_source)

`integral_add` (theorem; proof_source)

`integral_int` (theorem; proof_source)

`integral_mul` (theorem; proof_source)

`integral_promote` (theorem; proof_source)

`integral_sub` (theorem; proof_source)

`integral_sum` (theorem; proof_source)

`solve` (def; proof_source)

`solve_denominator_integral` (theorem; proof_source)

`solve_row` (theorem; proof_source)

`solve_step` (theorem; proof_source)

`solve_unique` (theorem; proof_source)

`solve_zero` (theorem; proof_source)

UnitModeDifferences

Source: baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/UnitModeDifferences.lean.

SHA-256: e1619f4ab356ac746fbf942ec6fedb4b196809290233d10e3437347d84c3b82f; 9 rows.

Namespace: KomorebiPaperIX.UnitModes

finite_mode_formula (theorem; proof_source)

finite_unit_modes_divisibility (theorem; proof_source)

finite_unit_modes_valuation_gain (theorem; proof_source)

geometric_difference (theorem; proof_source)

geometric_difference_divisibility (theorem; proof_source)

unit_fermat_divisibility (theorem; proof_source)

unit_mode_divisibility (theorem; proof_source)

unit_mode_valuation_gain (theorem; proof_source)

unit_multiple_step_divisibility (theorem; proof_source)

UnitRefinement

Source: baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/UnitRefinement.lean.

SHA-256: 4bb96f1682473774c23e51955cc4d7e730f9aab3e4a985a6baa4fd859e066709; 6 rows.

Namespace: KomorebiPaperIX.UnitRefinement

base_precision_principal (theorem; proof_source)

omega_compatible (theorem; proof_source)

principal_compatible (theorem; proof_source)

principal_power (theorem; proof_source)

reduce (def; proof_source)

reduceUnits (def; proof_source)

UnitTriangular

Source: baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/UnitTriangular.lean.

SHA-256: ab7423623dd1c26b41b65077960aa5d056b4c29125e1947815867b5cc0ecd26b; 6 rows.

Namespace: KomorebiPaperIX.UnitTriangular

field_solve_map (theorem; proof_source)

solve (def; proof_source)

solve_map_to_field (theorem; proof_source)

solve_row (theorem; proof_source)

solve_step (theorem; proof_source)

solve_unique (theorem; proof_source)

WeightedAmice

Source: baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/WeightedAmice.lean.

SHA-256: 7eac9c8d5d14f3f259821dc737e451b8fcb61dc04eab31c0a40d3174407aedb2; 23 rows.

Namespace: KomorebiPaperIX.WeightedAmice

Integral (def; proof_source)

Weighted (def; proof_source)

integral_add (theorem; proof_source)

integral_int (theorem; proof_source)

integral_mul (theorem; proof_source)

integral_nat (theorem; proof_source)

integral_neg (theorem; proof_source)

integral_one (theorem; proof_source)

integral_pow (theorem; proof_source)

integral_sum (theorem; proof_source)

integral_zero (theorem; proof_source)

scaled_mono (theorem; proof_source)

scaled_product (theorem; proof_source)

sectionPadic (def; proof_source)

weighted_add (theorem; proof_source)

weighted_constant (theorem; proof_source)

weighted_inverse (theorem; proof_source)

weighted_mul (theorem; proof_source)

weighted_neg (theorem; proof_source)

weighted_one (theorem; proof_source)

weighted_pow (theorem; proof_source)

weighted_section (theorem; proof_source)

weighted_zero (theorem; proof_source)

WeightedAmiceDifferential

Source: baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/WeightedAmiceDifferential.lean.

SHA-256: 6d6e1ce93e29fb6baff35f764f98c60fa89a2e9abd1d7fe8e360d69f40da22e2; 8 rows.

Namespace: KomorebiPaperIX.WeightedAmice

eulerLog (def; proof_source)

inverse_nat_integral (theorem; proof_source)

linear_le_power (theorem; proof_source)

valuation_le_floor (theorem; proof_source)

`weighted_X` (theorem; proof_source)

`weighted_derivative` (theorem; proof_source)

`weighted_eulerLog` (theorem; proof_source)

`weighted_log` (theorem; proof_source)

WeightedAmiceRing

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/WeightedAmiceRing.lean`.

SHA-256: `ab28188695852c09425cb6b80798ad8cc95f1163dedd1e0b65729dfa0eb685f7`; 15 rows.

Namespace: `KomorebiPaperIX.WeightedAmice`

`IsWeighted` (def; proof_source)

`X_mem` (theorem; proof_source)

`constant_mem` (theorem; proof_source)

`derivative_mem` (theorem; proof_source)

`inverse_coefficient` (theorem; proof_source)

`inverse_mem` (theorem; proof_source)

`mem_ring` (theorem; proof_source)

`monomial_mem` (theorem; proof_source)

`ordinary_mem` (theorem; proof_source)

`raise_weight` (theorem; proof_source)

`ring` (def; proof_source)

`shifted_weight` (theorem; proof_source)

`tail` (def; proof_source)

`tail_coeff` (theorem; proof_source)

`weighted_product_term` (theorem; proof_source)

WeightedDworkTransfer

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/WeightedDworkTransfer.lean`.

SHA-256: `d31afdf839ff4072edd9715cc2621c76d08716acbc4b8920b26d0b4ba10b1c7e`; 12 rows.

Namespace: `KomorebiPaperIX.WeightedDworkTransfer`

`B_expPull` (theorem; proof_source)

`constant_expPull` (theorem; proof_source)

`decode_expPull` (theorem; proof_source)

`deriv_expPull` (theorem; proof_source)

`derivative_expPull` (theorem; proof_source)

`expPull` (def; proof_source)

`expPull_flat` (theorem; proof_source)

`expPull_integral` (theorem; proof_source)

`expPull_log` (theorem; proof_source)

`flat_of_log_weighted` (theorem; proof_source)

`iterate_B_expPull` (theorem; proof_source)

`singular_ratio_flat` (theorem; proof_source)

WindowIndependence

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/WindowIndependence.lean`.

SHA-256: `247f6420080b7c26ffa8c1a001b2d85e1f03ca024918927655eb02708d804b6b`; 14 rows.

Namespace: `KomorebiPaperIX.WindowIndependence`

`coeffPadic` (def; proof_source)

`coeffPadic_coe` (theorem; proof_source)

`coeffPadic_row` (theorem; proof_source)

`diagonalUnit` (def; proof_source)

`diagonalUnit_coe` (theorem; proof_source)

`diagonal_padic_isUnit` (theorem; proof_source)

`reduced` (def; proof_source)

`reduced_row` (theorem; proof_source)

`reduced_unique` (theorem; proof_source)

`reduced_zero` (theorem; proof_source)

`residue_independence` (theorem; proof_source)

`standard_window_integral` (theorem; proof_source)

`two_window_addresses_agree` (theorem; proof_source)

`two_window_addresses_congruent` (theorem; proof_source)

ZigzagCoefficients

Source: `baseline/PRSC_IX_v0.11.0_Nishibuchi_Completion.zip!PRSC_IX_v0.11.0_Nishibuchi_Completion/formalization/ZigzagCoefficients.lean`.

SHA-256: `8fe52750980c04ce9dec3605facfcfbf9a9e98acf29948fa589e148d0f7b9d19`; 18 rows.

Namespace: `KomorebiPaperIX.Zigzag`

`cos_constant` (theorem; proof_source)

`cosine_sign_expansion` (theorem; proof_source)

`derivative_cos` (theorem; proof_source)

`derivative_sec` (theorem; proof_source)

`derivative_sin` (theorem; proof_source)

`derivative_tan` (theorem; proof_source)

`inverse_is_signed_zigzag` (theorem; proof_source)

`odd_zigzag_coefficient` (theorem; proof_source)

`secSeries` (def; proof_source)

`sec_cos` (theorem; proof_source)

`secant_odd_coeff` (theorem; proof_source)

`secant_sign_expansion` (theorem; proof_source)

`secant_square_even_coeff` (theorem; proof_source)

`sech_expansion` (theorem; proof_source)

`sin_constant` (theorem; proof_source)

`sin_square_add_cos_square` (theorem; proof_source)

`tanSeries` (def; proof_source)

`zigzag` (def; proof_source)

AffineObstructionControls

Source: `native/lean/replay_02/sources/AffineObstructionControls.lean`.

SHA-256: `625313196b0e109d793513ecd4f48581bf41493f83f1e65c41e0d709a52f7779`; 4 rows.

Namespace: `Komorebi.NativeAffine.Controls`

`identity_unobstructed` (theorem; positive_control)

`singular_lower_solution` (theorem; positive_control)

`singular_no_lift` (theorem; positive_control)

`zero_map_residual_nonzero` (theorem; positive_control)

FiniteAffineObstruction

Source: `native/lean/replay_02/sources/FiniteAffineObstruction.lean`.

SHA-256: `8f7ffb9f8387cbb23949f65d8d79ef601dd677ed567208f0bdeff475609ba056`; 13 rows.

Namespace: `Komorebi.NativeAffine`

`Obstruction` (abbrev; proof_source ; 2 source versions)

`affine_lift_iff_obstruction_zero` (theorem; proof_source ; 2 source versions)

`affine_residual_transport` (theorem; proof_source ; 2 source versions)

`digitMap` (def; proof_source ; 2 source versions)

`finite_affine_obstruction` (theorem; proof_source ; 2 source versions)

`obstruction` (def; proof_source ; 2 source versions)

`obstruction_finite` (theorem; proof_source ; 2 source versions)

`obstruction_representative_independent` (theorem; proof_source ; 2 source versions)

`obstruction_zero_iff_correction` (theorem; proof_source ; 2 source versions)

`reduction` (def; proof_source ; 2 source versions)

`reduction_zero_iff` (theorem; proof_source ; 2 source versions)

`residual_exists` (theorem; proof_source ; 2 source versions)

`same_residue_iff` (theorem; proof_source ; 2 source versions)

FiniteFiberCountsContract

Source: polynomial_counts/contracts/FiniteFiberCountsContract.lean.

SHA-256: 587f3ee59982c40bc9ad04da94d5fd67aee2b4d7eced7ce9f3327a5f27c5f71d; 2 rows.

Namespace: Komorebi.FiniteFiberCountsContract

explicit_fiber_and_obstruction_counts (theorem; statement_contract)

explicit_free_transitive (theorem; statement_contract)

PolynomialLiftCountContract

Source: polynomial_counts/contracts/PolynomialLiftCountContract.lean.

SHA-256: 11bf03c901c3d1353e269c240d58668340e159af1e8dab5e0ac450d624b2c84e; 2 rows.

Namespace: Komorebi.PolynomialLiftCountContract

explicit_actual_polynomial_count (theorem; statement_contract)

explicit_jacobian_formula (theorem; statement_contract)

ResidueDigitsContract

Source: polynomial_counts/contracts/ResidueDigitsContract.lean.

SHA-256: 96c06bf4928003510b7d7f0028a49cf1edf84dc64485737c920cc7e30eb00668; 2 rows.

Namespace: Komorebi.ResidueDigitsContract

explicit_residue_digit_equivalence (def; statement_contract)

explicit_residue_digit_formula (theorem; statement_contract)

PolynomialCountControls

Source: polynomial_counts/formalization/PolynomialCountControls.lean.

SHA-256: a4ddb1282ee57643fa17837bbc7fff9c0409bd7d092776b52e8f52e1fe23deea; 13 rows.

Namespace: Komorebi.PolynomialCountControls

linearSystem (def; positive_control)

linear_digit_count (theorem; positive_control)

linear_lift_count (theorem; positive_control)

obstructed_square_digit_count (theorem; positive_control)

obstructed_square_lift_count (theorem; positive_control)

singular_square_digit_count (theorem; positive_control)

singular_square_lift_count (theorem; positive_control)

squareMinusTwo (def; positive_control)

squareSystem (def; positive_control)

unrestricted_zero_root_count (theorem; positive_control)

zeroSystem (def; positive_control)

zero_digit_count (theorem; positive_control)

zero_lift_count (theorem; positive_control)

PolynomialObstructionContract

Source: `polynomial_lifting/contracts/PolynomialObstructionContract.lean`.

SHA-256: `b5f7d5ea68aa85097939d3913cfa0a0699ee080efd389a62746e3e2a33116d16`; 1 rows.

Namespace: `Komorebi.PolynomialObstructionContract`

`explicit_finite_polynomial_contract` (theorem; statement_contract)

PolynomialRepresentativeContract

Source: `polynomial_lifting/contracts/PolynomialRepresentativeContract.lean`.

SHA-256: `8e1ab12346c822e070d52f4482d36ea5d947a2ac6e2082e2f31365b673e0c5db`; 1 rows.

Namespace: `Komorebi.PolynomialRepresentativeContract`

`explicit_representative_contract` (theorem; statement_contract)

PolynomialSharedContract

Source: `polynomial_lifting/contracts/PolynomialSharedContract.lean`.

SHA-256: `dc013485992ecdcb51602ad758bbe2b52981f98417c951c74447dc81c2489c64`; 1 rows.

Namespace: `Komorebi.PolynomialSharedContract`

`explicit_shared_polynomial_contract` (theorem; statement_contract)

PolynomialTaylorContract

Source: `polynomial_lifting/contracts/PolynomialTaylorContract.lean`.

SHA-256: `d6bba92070ef41f5fe3da87b040eb0caf78f73120cfb322302981d158ce1fc69`; 1 rows.

Namespace: `Komorebi.PolynomialTaylorContract`

`explicit_integral_taylor_contract` (theorem; statement_contract)

AffinePairTransport

Source: `polynomial_lifting/formalization/AffinePairTransport.lean`.

SHA-256: `d6359e8bd04afc212633e2f902e1ea39d32b6922d0cff77e1d021634b5ca6082`; 4 rows.

Namespace: `Komorebi.AffinePair`

`HasJointLift` (def; proof_source)

`digit_correction_iff_same_image` (theorem; proof_source)

`joint_lift_iff_digit_correction` (theorem; proof_source)

`next_level_iff_digit` (theorem; proof_source)

AffineSharedCorrection

Source: `polynomial_lifting/formalization/AffineSharedCorrection.lean`.

SHA-256: `b4e412dbf811a08e0fb673dbf7776e653d1d16c946306fc81e8ddee7bf1687a2`; 7 rows.

Namespace: `Komorebi.AffinePair`

`SharedObstruction` (abbrev; proof_source)

`finite_affine_pair_compatibility` (theorem; proof_source)

`joint_lift_iff_class_eq` (theorem; proof_source)

`joint_lift_iff_compatible_corrections` (theorem; proof_source)

`left_class_independent` (theorem; proof_source)

`right_class_independent` (theorem; proof_source)

`shared_obstruction_finite` (theorem; proof_source)

PolynomialControls

Source: `polynomial_lifting/formalization/PolynomialControls.lean`.

SHA-256: `edda90f9dcd62b96e293e0e201676ceae12aea672706cf0f518528e3af7d5b89`; 9 rows.

Namespace: `Komorebi.PolynomialControls`

`even_square_cannot_be_two_mod_four` (theorem; positive_control)

`linearTwo` (def; positive_control)

`linearZero` (def; positive_control)

`no_common_polynomial_lift` (theorem; positive_control)

`separate_polynomial_lifts` (theorem; positive_control)

`singular_root_without_lift` (theorem; positive_control)

`squareMinusTwo` (def; positive_control)

`squarePolynomial` (def; positive_control)

`zero_depth_taylor_failure` (theorem; positive_control)

PolynomialRepresentative

Source: `polynomial_lifting/formalization/PolynomialRepresentative.lean`.

SHA-256: `68818904b747cd7b845877777458138ab0bed5938cf938a69b8e27acb5150fb2`; 5 rows.

Namespace: `Komorebi.PolynomialRepresentative`

`digit_jacobian_eq` (theorem; proof_source)

`digit_jacobian_shift_eq` (theorem; proof_source)

`representative_independence` (theorem; proof_source)

`residual_class_independent` (theorem; proof_source)

`residual_digit_transport` (theorem; proof_source)

PolynomialShared

Source: `polynomial_lifting/formalization/PolynomialShared.lean`.

SHA-256: `8df73989caf432acdf0b7278f95d0e3f0afa1e8804b77e56ff9b09e019ff0825`; 9 rows.

Namespace: `Komorebi.PolynomialShared`

`HasJointLift` (def; proof_source)

`SharedObstruction` (abbrev; proof_source)

`finite_polynomial_pair_compatibility` (theorem; proof_source)

`joint_lift_iff_class_eq` (theorem; proof_source)

`joint_lift_iff_compatible_corrections` (theorem; proof_source)

`joint_lift_iff_digit_correction` (theorem; proof_source)

`left_class_independent` (theorem; proof_source)

`right_class_independent` (theorem; proof_source)

`shared_obstruction_finite` (theorem; proof_source)

SharedCorrection

Source: `polynomial_lifting/formalization/SharedCorrection.lean`.

SHA-256: `77c9ad37f9f56bc107396050edc9399af3f4b8f3194fee3dd8a7a6a1428d7d76`; 19 rows.

Namespace: `Komorebi.SharedCorrection`

`Common` (abbrev; proof_source)

`commonKernel` (def; proof_source)

`common_finite_of_left` (theorem; proof_source)

`common_finite_of_right` (theorem; proof_source)

`common_paired_universal` (theorem; proof_source)

`common_universal` (theorem; proof_source)

`fiber` (def; proof_source)

`joint` (def; proof_source)

`jointImageEquivFiber` (def; proof_source)

`joint_range_eq_fiber` (theorem; proof_source)

`labelJ` (def; proof_source)

`labelJ_image` (theorem; proof_source)

`labelJ_surjective` (theorem; proof_source)

`labelK` (def; proof_source)

`labelK_image` (theorem; proof_source)

`labelK_surjective` (theorem; proof_source)

`labels_eq_iff_common` (theorem; proof_source)

`mem_fiber` (theorem; proof_source)

`quotient_mk_eq_iff_common` (theorem; proof_source)

PolynomialCoefficientPerturbationContract

Source: `polynomial_regularity/contracts/PolynomialCoefficientPerturbationContract.lean`.

SHA-256: `bf8c6423f4b764b7adaf51ab6a52faeca32404cd7ebda30685f442f1bfefd115`; 2 rows.

Namespace: `Komorebi.PolynomialCoefficientPerturbationContract`

`explicit_perturbed_jacobian` (theorem; statement_contract)

`explicit_power_perturbation_root` (theorem; statement_contract)

PolynomialGeneralChartContract

Source: `polynomial_regularity/contracts/PolynomialGeneralChartContract.lean`.

SHA-256: `42a57317eb813cd5f29a9a58008153ad374bb606c573bd707d057bb36a5a8859`; 1 rows.

Namespace: `Komorebi.PolynomialGeneralChartContract`

`full_rectangular_chart` (theorem; statement_contract)

PolynomialHomeomorphContract

Source: `polynomial_regularity/contracts/PolynomialHomeomorphContract.lean`.

SHA-256: `93a559a213702a9470b150b6e074862049ce976c92764f67d40c0813103588d9`; 1 rows.

Namespace: `Komorebi.PolynomialHomeomorphContract`

`square_homeomorphism` (theorem; statement_contract)

PolynomialJointMotionContracts

Source: `polynomial_regularity/contracts/PolynomialJointMotionContracts.lean`.

SHA-256: `ef407d17d713a64f01ef69eaf1f1a798768cf3b543004d43f56f00b692248eef`; 2 rows.

Namespace: `Komorebi.PolynomialJointMotionContracts`

`actual_joint_certificate` (theorem; statement_contract)

`actual_joint_rational_motion` (theorem; statement_contract)

PolynomialLocalMapContract

Source: `polynomial_regularity/contracts/PolynomialLocalMapContract.lean`.

SHA-256: `6203a1d89bbf7e079bbbac8167ae7a5ab0ada098e573500abec6e34900136648`; 2 rows.

Namespace: `Komorebi.PolynomialLocalMapContract`

`all_depths_reflected` (theorem; statement_contract)

`arbitrary_padic_target` (theorem; statement_contract)

PolynomialPadicUniqueContract

Source: `polynomial_regularity/contracts/PolynomialPadicUniqueContract.lean`.

SHA-256: `62d3afb540bb019bc3d705683d1ec4e9fd6d279c61a29ec60c13dd19598ad6f1`; 2 rows.

Namespace: `Komorebi.PolynomialPadicUniqueContract`

`explicit_arbitrary_root_uniqueness` (theorem; statement_contract)

`explicit_unique_padic_root` (theorem; statement_contract)

PolynomialParameterTowerContracts

Source: `polynomial_regularity/contracts/PolynomialParameterTowerContracts.lean`.

SHA-256: `40a450b93c3d686cfd7ea84081a0cb061e00730cc96ff542015e4e8751d48ae6`; 5 rows.

Namespace: `Komorebi.PolynomialParameterTowerContracts`

`actual_integer_path_tower` (theorem; statement_contract)

`actual_square_parameter_uniqueness` (theorem; statement_contract)

`arbitrary_padic_parameter_path` (theorem; statement_contract)

`arbitrary_padic_target_tower` (theorem; statement_contract)

`supplied_path_actual_endpoint` (theorem; statement_contract)

PolynomialPrescribedContract

Source: `polynomial_regularity/contracts/PolynomialPrescribedContract.lean`.

SHA-256: `6598a12ba513de9a23f1745a50d241ae5ca5a5dade6771c312747171b166c6cc`; 5 rows.

Namespace: `Komorebi.PolynomialPrescribedContract`

`ActualResidues` (abbrev; statement_contract)

`actualResidueEquiv` (def; statement_contract)

`actualSolvedMap` (def; statement_contract)

`actual_lower_root_obstruction` (theorem; statement_contract)

`actual_lower_root_prime_count` (theorem; statement_contract)

PolynomialRegularityContracts

Source: `polynomial_regularity/contracts/PolynomialRegularityContracts.lean`.

SHA-256: `4b3ef7ae03c57942325aaafc278848ac1c339c6131235c8e053c7cf7ffa5eff6`; 5 rows.

Namespace: `Komorebi.PolynomialRegularityContracts`

`actual_integral_analytic_root_graph` (theorem; statement_contract)

`actual_padic_analytic_root_graph` (theorem; statement_contract)

`actual_polynomial_evaluation` (theorem; statement_contract)

`actual_selection_analytic_on_open` (theorem; statement_contract)

`actual_selection_germ` (theorem; statement_contract)

PolynomialRootMotionContract

Source: `polynomial_regularity/contracts/PolynomialRootMotionContract.lean`.

SHA-256: `c1f671c373ed386790176b517c19c65974c43fc0c952907adbd72c955bc3db5b`; 2 rows.

Namespace: `Komorebi.PolynomialRootMotionContract`

`original_jacobian_integral_error` (theorem; statement_contract)

`original_jacobian_rational_motion` (theorem; statement_contract)

PolynomialRootStabilityContract

Source: `polynomial_regularity/contracts/PolynomialRootStabilityContract.lean`.

SHA-256: `cb5dddb6def6a35ff96eccd9096c42a11476a4e0cbbb371dd8a552324f583c62`; 3 rows.

Namespace: `Komorebi.PolynomialRootStabilityContract`

`explicit_lower_root_and_unique_perturbation` (theorem; statement_contract)

`explicit_rectangular_nearby_root` (theorem; statement_contract)

`explicit_square_root_displacement` (theorem; statement_contract)

PolynomialWeightedAuxiliaryContract

Source: `polynomial_regularity/contracts/PolynomialWeightedAuxiliaryContract.lean`.

SHA-256: `a734bcfd612bfbbd4ffe694d6795d5198ee9dc9124d499ac0076474a37f4c057`; 3 rows.

Namespace: `Komorebi.PolynomialWeightedAuxiliaryContract`

`actual_polynomial_contraction` (theorem; statement_contract)

`exact_inverse_denominator_certificate` (theorem; statement_contract)

`uniform_polynomial_taylor` (theorem; statement_contract)

PolynomialWeightedLiftContract

Source: `polynomial_regularity/contracts/PolynomialWeightedLiftContract.lean`.

SHA-256: `f74ab51190f4e7e6a4fa503ec2216bebd5496c6bbd160054196b4ba0e17679ef`; 2 rows.

Namespace: `Komorebi.PolynomialWeightedLiftContract`

`actual_jacobian_bounded_inverse` (theorem; statement_contract)

`actual_jacobian_integral_certificate` (theorem; statement_contract)

PolynomialDeformationControls

Source: `polynomial_regularity/controls/PolynomialDeformationControls.lean`.

SHA-256: `72809a109644db6ce09ffe1b1f7244e2759d4ccbca060f0357be82a6c8a80439`; 6 rows.

Namespace: `Komorebi.PolynomialDeformationControls`

`linear_shift_coefficient_perturbation` (theorem; positive_control)

`linear_shift_no_extra_padic_precision` (theorem; positive_control)

`squareMinusOne` (def; positive_control)

`square_minus_one_jacobian_bijective` (theorem; positive_control)

`square_minus_one_not_global_unique` (theorem; positive_control)

`square_minus_one_two_padic_roots` (theorem; positive_control)

PolynomialJointMotionControls

Source: `polynomial_regularity/controls/PolynomialJointMotionControls.lean`.

SHA-256: `7f564ab0c011cae38264f3d7f149864649fdd329bb28412bdb94e5e2e3cdb362`; 16 rows.

Namespace: `Komorebi.PolynomialJointMotionControls`

`exists_sharp_parameter_only_root` (theorem; positive_control)

`join_one_one` (theorem; positive_control)

`join_one_zero` (theorem; positive_control)

`linearPerturbation` (def; positive_control)

`linearSystem` (def; positive_control)

`linear_actual_jacobians` (theorem; positive_control)

`linear_augmented_root_depth_two` (theorem; positive_control)

`linear_base_data` (theorem; positive_control)

`linear_joint_forcing_and_error` (theorem; positive_control)

`linear_joint_root` (theorem; positive_control)

`linear_joint_unique` (theorem; positive_control)

`no_uniform_extra_parameter_error_digit` (theorem; positive_control)

`omitted_parameter_term_fails` (theorem; positive_control)

`quadraticParameterSystem` (def; positive_control)

`quadratic_parameter_base_data` (theorem; positive_control)

`reversed_parameter_sign_fails` (theorem; positive_control)

PolynomialMotionControls

Source: `polynomial_regularity/controls/PolynomialMotionControls.lean`.

SHA-256: `4c90b8177833dc905f9cd6996805682dce59e9084dcaf1e4e333d5370a14d883`; 6 rows.

Namespace: `Komorebi.PolynomialMotionControls`

`exists_sharp_motion_root` (theorem; positive_control)

`no_uniform_extra_error_digit` (theorem; positive_control)

`sharpQuadratic` (def; positive_control)

`sharpQuadratic_base` (theorem; positive_control)

`sharp_error_of_even_root` (theorem; positive_control)

`sharp_mod_eight` (theorem; positive_control)

PolynomialParameterControls

Source: `polynomial_regularity/controls/PolynomialParameterControls.lean`.

SHA-256: `43a88e22897c64c0403330d0189827c18eec60bf45c0a85e9ac0eb45d38b7a83`; 6 rows.

Namespace: `Komorebi.PolynomialParameterControls`

`full_jacobian_surjective` (theorem; positive_control)

`no_root_at_parameter_two` (theorem; positive_control)

`parameter_two_near_origin` (theorem; positive_control)

`solved_jacobian_not_surjective` (theorem; positive_control)

`solved_jacobian_zero` (theorem; positive_control)

`squareMinusParameter` (def; positive_control)

PolynomialParameterTowerControls

Source: `polynomial_regularity/controls/PolynomialParameterTowerControls.lean`.

SHA-256: `41037b0fbd33d89ee7c4601a05455441d4d618ae341bea9a2789b8730bdee5bb`; 3 rows.

Namespace: `Komorebi.PolynomialParameterTowerControls`

`firstSolved` (def; positive_control)

`firstSolved_surjective` (theorem; positive_control)

`rectangular_same_parameter_nonunique` (theorem; positive_control)

PolynomialPrescribedControls

Source: `polynomial_regularity/controls/PolynomialPrescribedControls.lean`.

SHA-256: `5cce4a5e5295ba98e65ea52efe72741b92e2607bfbe4844e8d5ad647c8be8a5a`; 15 rows.

Namespace: `Komorebi.PolynomialPrescribedControls`

`digit_one_empty` (theorem; positive_control)

`digit_one_for_three_empty` (theorem; positive_control)

`digit_two_for_three_exists` (theorem; positive_control)

`full_jacobian_surjective` (theorem; positive_control)

`parameterOnly` (def; positive_control)

`parameterPlusThree` (def; positive_control)

`prescribed_one_for_three_no_actual_lift` (theorem; positive_control)

`prescribed_one_no_actual_lift` (theorem; positive_control)

`prescribed_two_for_three_actual_lift` (theorem; positive_control)

`prescribed_zero_actual_card` (theorem; positive_control)

`solved_jacobian_zero` (theorem; positive_control)

`three_residual` (theorem; positive_control)

`zeroDigitEquiv` (def; positive_control)

`zero_digit_card` (theorem; positive_control)

`zero_residual` (theorem; positive_control)

PolynomialRegularityControls

Source: `polynomial_regularity/controls/PolynomialRegularityControls.lean`.

SHA-256: `a84a69f9c32efb43b4bfbccf86b7879002f4327ad8be06da3bf03c75793605e2`; 12 rows.

Namespace: `Komorebi.PolynomialRegularityControls`

`linearSystem` (def; positive_control)

`linear_analytic` (theorem; positive_control)

`linear_derivative_not_positive` (theorem; positive_control)

`linear_parameterJacobian` (theorem; positive_control)

`linear_root_graph` (theorem; positive_control)

`linear_solvedJacobian` (theorem; positive_control)

`linear_strict_derivative` (theorem; positive_control)

`quadraticSystem` (def; positive_control)

`quadratic_analytic_branch` (theorem; positive_control)

`quadratic_parameterJacobian` (theorem; positive_control)

`quadratic_root` (theorem; positive_control)

`quadratic_solvedJacobian` (theorem; positive_control)

PolynomialWeightedControls

Source: `polynomial_regularity/controls/PolynomialWeightedControls.lean`.

SHA-256: `88fea6bc068832ba84339af00cd98b0f9b458219dd3df71c1578f445cb7fdf04`; 11 rows.

Namespace: `Komorebi.PolynomialWeightedControls`

`boundary_exact_residual_depth` (theorem; positive_control)

`boundary_no_mod_eight` (theorem; positive_control)

`boundary_no_padic_root` (theorem; positive_control)

`boundary_no_root` (theorem; positive_control)

`boundary_residual_derivative` (theorem; positive_control)

`boundary_scaled_inverse` (theorem; positive_control)

`linear_displacement_sharp` (theorem; positive_control)

`linear_perturbation_root` (theorem; positive_control)

`prime_power_exact_depth` (theorem; positive_control)

`squarePlusThree` (def; positive_control)

`weightedLinear` (def; positive_control)

AffineObstructionCore

Source: `polynomial_regularity/formalization/AffineObstructionCore.lean`.

SHA-256: `8af3cd815f87519dfa90aec72244a7924f19d4291d29f116a97fcce31af9f80f`; 13 rows.

Namespace: `Komorebi.NativeAffine`

`Obstruction` (abbrev; proof_source ; 2 source versions)

`affine_lift_iff_obstruction_zero` (theorem; proof_source ; 2 source versions)

`affine_residual_transport` (theorem; proof_source ; 2 source versions)

`digitMap` (def; proof_source ; 2 source versions)

`finite_affine_obstruction` (theorem; proof_source ; 2 source versions)

`obstruction` (def; proof_source ; 2 source versions)

`obstruction_finite` (theorem; proof_source ; 2 source versions)

`obstruction_representative_independent` (theorem; proof_source ; 2 source versions)

`obstruction_zero_iff_correction` (theorem; proof_source ; 2 source versions)

`reduction` (def; proof_source ; 2 source versions)

`reduction_zero_iff` (theorem; proof_source ; 2 source versions)

`residual_exists` (theorem; proof_source ; 2 source versions)

`same_residue_iff` (theorem; proof_source ; 2 source versions)

FiniteFiberCounts

Source: `polynomial_regularity/formalization/FiniteFiberCounts.lean`.

SHA-256: `4e7e7fcc6350236114543d1db3c3b290da45b1bcd0b92b630cb9581ec08c6528`; 20 rows.

Namespace: `Komorebi.FiniteFiberCounts`

`Fiber` (def; proof_source)

`cokernel_card` (theorem; proof_source)

`cokernel_finrank` (theorem; proof_source)

`difference` (def; proof_source)

`difference_translate` (theorem; proof_source)

`fiberAddTorsor` (def; proof_source)

`fiberEquivKernel` (def; proof_source)

`fiber_card` (theorem; proof_source)

`fiber_card_eq_kernel` (theorem; proof_source)

`fiber_card_of_empty` (theorem; proof_source)

`fiber_card_of_nonempty` (theorem; proof_source)

`fiber_nonempty_iff_class_zero` (theorem; proof_source)

`kernel_card` (theorem; proof_source)

`kernel_finrank` (theorem; proof_source)

`rank` (def; proof_source)

`translate` (def; proof_source)

`translate_add` (theorem; proof_source)

`translate_difference` (theorem; proof_source)

`translate_exists_unique` (theorem; proof_source)

`translate_zero` (theorem; proof_source)

IntegerFiniteCokernel

Source: `polynomial_regularity/formalization/IntegerFiniteCokernel.lean`.

SHA-256: `ca844b9373b49e64974717ab73eddf56ddfa6df45b11e9c4334c9ac2c1bae8e3`; 6 rows.

Namespace: `Komorebi.IntegerFiniteCokernel`

`cokernelEquiv` (def; proof_source)

`cokernelEquiv_mk` (theorem; proof_source)

`cokernelEquiv_obstruction` (theorem; proof_source)

`cokernel_card_eq` (theorem; proof_source)

`digit_range_eq_restrict` (theorem; proof_source)

`reduction_surjective` (theorem; proof_source)

PadicTowerLimit

Source: `polynomial_regularity/formalization/PadicTowerLimit.lean`.

SHA-256: `43d4cab78912d49bb0c23645d3b1af4d4148d188781c31957e611902ff603528`; 6 rows.

Namespace: `Komorebi.PadicTowerLimit`

`compatible_int_tower_cauchy` (theorem; proof_source)

`compatible_int_tower_limit` (theorem; proof_source)

`compatible_int_tower_unique_limit` (theorem; proof_source)

`compatible_vector_tower_limit` (theorem; proof_source)

`divisible_int_tendsto_zero` (theorem; proof_source)

`tower_pair_dvd` (theorem; proof_source)

PolynomialAnalyticBlocks

Source: `polynomial_regularity/formalization/PolynomialAnalyticBlocks.lean`.

SHA-256: `19fa21ab3877afe7925003f2c142267b214159d9c217f94204e5765e8e1d5955`; 6 rows.

Namespace: `Komorebi.PolynomialAnalyticBlocks`

`inverse_free_zero` (theorem; proof_source)

`inverse_solved_zero` (theorem; proof_source)

`upperEquiv` (def; proof_source)

`upperEquiv_apply` (theorem; proof_source)

`upperEquiv_symm_apply` (theorem; proof_source)

`upperEquiv_toCLM` (theorem; proof_source)

PolynomialAnalyticEvaluation

Source: `polynomial_regularity/formalization/PolynomialAnalyticEvaluation.lean`.

SHA-256: `f8d75e73ac2df1e336c55889da4e81184f54ff6bffe94eee0112d90e905ff57f`; 9 rows.

Namespace: `Komorebi.PolynomialAnalyticEvaluation`

`analyticAt_evalSystem` (theorem; proof_source)

`analyticOnNhd_evalSystem` (theorem; proof_source)

`evalSystem` (def; proof_source)

`hasStrictFDerivAt_eval` (theorem; proof_source)

`hasStrictFDerivAt_evalSystem` (theorem; proof_source)

`matrixCLM` (def; proof_source)

`matrixCLM_apply` (theorem; proof_source)

`polynomialDeriv` (def; proof_source)

`polynomialDeriv_apply` (theorem; proof_source)

PolynomialAnalyticImplicit

Source: `polynomial_regularity/formalization/PolynomialAnalyticImplicit.lean`.

SHA-256: `4fb68eb1f5417a3ecc46a281d5eddb23afa551ac3cb4c65354b8b907c681bb7a`; 7 rows.

Namespace: `Komorebi.PolynomialAnalyticImplicit`

`differentiableOn_of_analytic_graph` (theorem; proof_source)

`exists_analytic_graph` (theorem; proof_source)

`exists_analytic_implicit` (theorem; proof_source)

`parameterInjectionCLM` (def; proof_source)

`parameterInjectionCLM_apply` (theorem; proof_source)

`solvedPartCLM` (def; proof_source)

`solvedPartCLM_apply` (theorem; proof_source)

PolynomialAnalyticInverse

Source: `polynomial_regularity/formalization/PolynomialAnalyticInverse.lean`.

SHA-256: `4139bc9fe166b3f9aa926fa1f5d4b83d5155c2a31fa0826dac8e48fad422b3c7`; 2 rows.

Namespace: `Komorebi.PolynomialAnalyticInverse`

`analyticAt_inverse` (theorem; proof_source)

`exists_open_analytic_inverse` (theorem; proof_source)

PolynomialAnalyticSelection

Source: `polynomial_regularity/formalization/PolynomialAnalyticSelection.lean`.

SHA-256: `ed23cb0c55b662bfb8270fa17e2af27c4dff8c5605c6a1e65c4fa35bbca0ba82`; 3 rows.

Namespace: `Komorebi.PolynomialAnalyticSelection`

`continuous_selection_analyticOnNhd` (theorem; proof_source)

`continuous_selection_eventually_eq` (theorem; proof_source)

`continuous_selection_regularity` (theorem; proof_source)

PolynomialBlockInverse

Source: `polynomial_regularity/formalization/PolynomialBlockInverse.lean`.

SHA-256: `398f1123b76a2d458bac4eb6d671d15ad41c36d329c70cac8a3fdc0bca3a945a`; 12 rows.

Namespace: `Komorebi.PolynomialBlockInverse`

`blockCertificate` (def; proof_source)

`blockCertificate_mul` (theorem; proof_source)

`blockCertificate_mulVec_join_neg` (theorem; proof_source)

`blockCertificate_two_sided` (theorem; proof_source)

`bounded_inverse_upperBlock` (theorem; proof_source)

`mul_blockCertificate` (theorem; proof_source)

`upperBlock` (def; proof_source)

`upperBlock_free_free` (theorem; proof_source)

`upperBlock_free_solved` (theorem; proof_source)

`upperBlock_mulVec_join` (theorem; proof_source)

`upperBlock_solved_free` (theorem; proof_source)

`upperBlock_solved_solved` (theorem; proof_source)

PolynomialCoefficientPerturbation

Source: `polynomial_regularity/formalization/PolynomialCoefficientPerturbation.lean`.

SHA-256: `fbb6841c84bf08cd074cd3ecdb40ef9fec4cc65708272abd0fd034f7ccbc5198`; 9 rows.

Namespace: `Komorebi.PolynomialCoefficientPerturbation`

`digitJacobian_perturb_eq` (theorem; proof_source)

`digitJacobian_perturb_point_eq` (theorem; proof_source)

`evalSystem_perturb` (theorem; proof_source)

`exists_padic_root_perturb` (theorem; proof_source)

`exists_padic_root_power_perturb` (theorem; proof_source)

`lower_root_perturb` (theorem; proof_source)

`padicEval_perturb` (theorem; proof_source)

`pderiv_perturb` (theorem; proof_source)

`perturb` (def; proof_source)

PolynomialCoordinateReorder

Source: `polynomial_regularity/formalization/PolynomialCoordinateReorder.lean`.

SHA-256: `d8a5e2e71e5c56103816a083a914f8ff01e24b881ecae3e9c04a97a98a7f2652`; 14 rows.

Namespace: `Komorebi.PolynomialCoordinateReorder`

`continuous_reorderPoint` (theorem; proof_source)

`digitJacobian_single` (theorem; proof_source)

`evalSystem_reorder` (theorem; proof_source)

`evalSystem_reorder_at` (theorem; proof_source)

`exists_reorder_solvedJacobian_surjective` (theorem; proof_source)

`padicEval_reorder` (theorem; proof_source)

`padicEval_reorder_at` (theorem; proof_source)

`pderiv_eval_reorder` (theorem; proof_source)

`pderiv_reorder` (theorem; proof_source)

`reorderPoint` (def; proof_source)

`reorderPoint_inverse` (theorem; proof_source)

`reorderPoint_inverse'` (theorem; proof_source)

`reorderSystem` (def; proof_source)

`solvedJacobian_reorder_apply` (theorem; proof_source)

PolynomialDigitBridge

Source: `polynomial_regularity/formalization/PolynomialDigitBridge.lean`.

SHA-256: `6a985c2c6308910d148283297e71b0cd30b1abdc86056898aa2b5294fedc27e2`; 7 rows.

Namespace: `Komorebi.PolynomialDigitBridge`

`DigitSolutions` (abbrev; proof_source)

`digitJacobian` (def; proof_source)

`digit_solution_iff` (theorem; proof_source)

`digit_solutions_nonempty_iff` (theorem; proof_source)

`integerLift` (def; proof_source)

`reduction_integerLift` (theorem; proof_source)

`reduction_jacobian` (theorem; proof_source)

PolynomialGeneralChart

Source: `polynomial_regularity/formalization/PolynomialGeneralChart.lean`.

SHA-256: `492bccc855312c5b262c31bbfe1b7521dc03f30ef15afc3739d05874ff6062c8`; 2 rows.

Namespace: `Komorebi.PolynomialGeneralChart`

`exists_parameter_homeomorph` (theorem; proof_source)

`reorderRootHomeomorph` (def; proof_source)

PolynomialIntegerTower

Source: `polynomial_regularity/formalization/PolynomialIntegerTower.lean`.

SHA-256: `1b36300b65e2771cb785fce035044c2b80e4210412133077be7788785a9c16a8`; 5 rows.

Namespace: `Komorebi.PolynomialIntegerTower`

`exists_compatible_tower` (theorem; proof_source)

`exists_compatible_tower_strong` (theorem; proof_source)

`tower_digitJacobian_eq` (theorem; proof_source)

`tower_initial_congruent` (theorem; proof_source)

`tower_pairwise_congruent` (theorem; proof_source)

PolynomialInverseCongruence

Source: `polynomial_regularity/formalization/PolynomialInverseCongruence.lean`.

SHA-256: `298bba9f501f5dc5a983a3eb7a34fa6790d7ec16279b220101fae23f5bf32ebf`; 6 rows.

Namespace: `Komorebi.PolynomialInverseCongruence`

`integer_inverse_congruence` (theorem; proof_source)

`integer_inverse_congruence_step` (theorem; proof_source)

`padicEval_congruent` (theorem; proof_source)

`padic_congruence_iff` (theorem; proof_source)

padic_inverse_congruence (theorem; proof_source)

padic_map_injective (theorem; proof_source)

PolynomialJointDeformation

Source: polynomial_regularity/formalization/PolynomialJointDeformation.lean.

SHA-256: 5cef4d16ad05e035326f23b1ec62468b4e1e5708e29218513d9fb22528450345; 3 rows.

Namespace: Komorebi.PolynomialJointDeformation

exists_joint_deformation (theorem; proof_source)

exists_unique_joint_deformation (theorem; proof_source)

join_deformed_ball (theorem; proof_source)

PolynomialJointMotion

Source: polynomial_regularity/formalization/PolynomialJointMotion.lean.

SHA-256: 946ca8e3e4f17c71e66db136a1ab70b1012c60dbc1f87ded33f2d7defb1b84e4; 2 rows.

Namespace: Komorebi.PolynomialJointMotion

exists_joint_root_certificate (theorem; proof_source)

exists_joint_root_motion (theorem; proof_source)

PolynomialJointSystem

Source: polynomial_regularity/formalization/PolynomialJointSystem.lean.

SHA-256: 01aec0f34cdcf5a6f9288f4af342bb804678ecb1713b1aef40cf6fa227e767df; 17 rows.

Namespace: Komorebi.PolynomialJointSystem

augmented (def; proof_source)

augmentedPerturbation (def; proof_source)

augmentedPerturbation_free (theorem; proof_source)

augmentedPerturbation_solved (theorem; proof_source)

augmented_free (theorem; proof_source)

augmented_root (theorem; proof_source)

augmented_solved (theorem; proof_source)

eval_augmented (theorem; proof_source)

eval_augmentedPerturbation (theorem; proof_source)

jacobian_augmented (theorem; proof_source)

jacobian_augmented_free_free (theorem; proof_source)

jacobian_augmented_free_solved (theorem; proof_source)

jacobian_augmented_solved_free (theorem; proof_source)

jacobian_augmented_solved_solved (theorem; proof_source)

parameterJacobian (def; proof_source)

perturb_augmented_root_iff (theorem; proof_source)

`solvedJacobian` (def; proof_source)

PolynomialLiftCounts

Source: `polynomial_regularity/formalization/PolynomialLiftCounts.lean`.

SHA-256: `10c01bfa8a892f0b6aacb7f36e1c0c86cef6ddfa51917ae5c0a84be32f963931`; 10 rows.

Namespace: `Komorebi.PolynomialLiftCounts`

`ActualLifts` (abbrev; proof_source)

`digitAddTorsor` (def; proof_source)

`digitEquivLifts` (def; proof_source)

`digit_residue_solution_iff` (theorem; proof_source)

`finite_polynomial_lift_count` (theorem; proof_source)

`liftEquivKernel` (def; proof_source)

`lift_card` (theorem; proof_source)

`obstructionEquiv` (def; proof_source)

`obstructionEquiv_class` (theorem; proof_source)

`obstruction_card` (theorem; proof_source)

PolynomialLocalInverse

Source: `polynomial_regularity/formalization/PolynomialLocalInverse.lean`.

SHA-256: `11d6dda31c7932a5175a1ecfb99f18e0a9d287098997bfc5a43789af8385f635`; 7 rows.

Namespace: `Komorebi.PolynomialLocalInverse`

`ballMap` (def; proof_source)

`ballMap_bijective` (theorem; proof_source)

`continuous_ballMap` (theorem; proof_source)

`localHomeomorph` (def; proof_source)

`localHomeomorph_apply` (theorem; proof_source)

`localHomeomorph_symm_congruent` (theorem; proof_source)

`localHomeomorph_symm_eval` (theorem; proof_source)

PolynomialLocalSurjective

Source: `polynomial_regularity/formalization/PolynomialLocalSurjective.lean`.

SHA-256: `c4ddf4985f7739b5e8555d103c85e131524d7c04d855bbbe94826ba58824c937`; 7 rows.

Namespace: `Komorebi.PolynomialLocalSurjective`

`digitJacobian_sub_C` (theorem; proof_source)

`exists_integer_target_preimage` (theorem; proof_source)

`exists_padic_preimage` (theorem; proof_source)

`isClosed_residueBall` (theorem; proof_source)

`isCompact_residueBall` (theorem; proof_source)

`residueBall` (def; proof_source)

`truncInt_tendsto` (theorem; proof_source)

PolynomialMinorSelection

Source: `polynomial_regularity/formalization/PolynomialMinorSelection.lean`.

SHA-256: `00ca2ec49dcaf069c90d882e7edf4b088c68f7d4a37f42fcd728e39f351e4c3e`; 3 rows.

Namespace: `Komorebi.PolynomialMinorSelection`

`exists_permuted_columns` (theorem; proof_source)

`exists_prime_permuted_columns` (theorem; proof_source)

`exists_selected_columns` (theorem; proof_source)

PolynomialMotionEstimate

Source: `polynomial_regularity/formalization/PolynomialMotionEstimate.lean`.

SHA-256: `b805b3ce934dc0eeda54be631e7a1210c1919d4c231b9794cdf6eeb64e6e2f4b`; 1 rows.

Namespace: `Komorebi.PolynomialMotionEstimate`

`first_order_error_dvd` (theorem; proof_source)

PolynomialMotionNormalization

Source: `polynomial_regularity/formalization/PolynomialMotionNormalization.lean`.

SHA-256: `488c028384e45fe3e9a2f3365227bf4f2583beddc73b7ad76faf8e5b605249c2`; 5 rows.

Namespace: `Komorebi.PolynomialMotionNormalization`

`eval_perturbSystem` (theorem; proof_source)

`exists_eval_scaled_difference` (theorem; proof_source)

`exists_motion_normalization` (theorem; proof_source)

`perturbSystem` (def; proof_source)

`precondition_perturbSystem` (theorem; proof_source)

PolynomialMotionRational

Source: `polynomial_regularity/formalization/PolynomialMotionRational.lean`.

SHA-256: `17d45ac3501b93146f907f59702e96fabee320fb15449756685fe3e9fb480beb`; 3 rows.

Namespace: `Komorebi.PolynomialMotionRational`

`coe_mulVec` (theorem; proof_source)

`exists_rational_error` (theorem; proof_source)

`scaled_inverse_mulVec` (theorem; proof_source)

PolynomialObstruction

Source: `polynomial_regularity/formalization/PolynomialObstruction.lean`.

SHA-256: `ab0670c8d9c84aaf36e0e83f396c26f0514b8279f694d7b3653c3ab43410c653`; 12 rows.

Namespace: `Komorebi.PolynomialObstruction`

`dvd_iff_of_dvd_sub` (theorem; proof_source)

`evalSystem` (def; proof_source)

`evalSystem_apply` (theorem; proof_source)

`finite_polynomial_obstruction` (theorem; proof_source)

`jacobian` (def; proof_source)

`jacobian_apply` (theorem; proof_source)

`next_level_iff_digit` (theorem; proof_source)

`polynomial_lift_iff_obstruction_zero` (theorem; proof_source)

`polynomial_same_residue_lift_iff` (theorem; proof_source)

`residual_exists` (theorem; proof_source)

`residual_unique` (theorem; proof_source)

`taylor_residual_dvd` (theorem; proof_source)

PolynomialPadicContraction

Source: `polynomial_regularity/formalization/PolynomialPadicContraction.lean`.

SHA-256: `e667bd03d6577c8a7180659ef5fa95f068ccc172a7cf43d0f7f2508156f8e7ce`; 9 rows.

Namespace: `Komorebi.PolynomialPadicContraction`

`contracting_contractionMap` (theorem; proof_source)

`contractionMap` (def; proof_source)

`evalSystem` (def; proof_source)

`eval_dist_le` (theorem; proof_source)

`eval_dvd_of_dvd` (theorem; proof_source)

`exists_unique_fixedPoint` (theorem; proof_source)

`exists_unique_fixedPoint_divisible` (theorem; proof_source)

`fixedPoint_divisible` (theorem; proof_source)

`lipschitz_evalSystem` (theorem; proof_source)

PolynomialPadicLift

Source: `polynomial_regularity/formalization/PolynomialPadicLift.lean`.

SHA-256: `072b6ad7c76a858d63d674a9eabd6d0604a9d2903ce7f0d81bf5c7ab3eb6cda6`; 6 rows.

Namespace: `Komorebi.PolynomialPadicLift`

`continuous_padicEval` (theorem; proof_source)

`exists_padic_root` (theorem; proof_source)

`exists_padic_root_with_tower` (theorem; proof_source)

`padicEval` (def; proof_source)

`padicEval_intCast` (theorem; proof_source)

`polynomial_tower_limit_root` (theorem; proof_source)

PolynomialPadicUnique

Source: `polynomial_regularity/formalization/PolynomialPadicUnique.lean`.

SHA-256: `51a734f436ec118fd39cf20ad228eedaffe5225eeb989b5a1040deb701c0f4cd`; 10 rows.

Namespace: `Komorebi.PolynomialPadicUnique`

`exists_unique_padic_root` (theorem; proof_source)

`int_pow_dvd_iff` (theorem; proof_source)

`padic_root_unique` (theorem; proof_source)

`rootResidueLift` (def; proof_source)

`root_next_residue_eq` (theorem; proof_source)

`toZModPow_eq_iff_dvd_sub` (theorem; proof_source)

`truncInt` (def; proof_source)

`truncInt_eval_residue` (theorem; proof_source)

`truncInt_precision` (theorem; proof_source)

`truncInt_root` (theorem; proof_source)

PolynomialParameterChart

Source: `polynomial_regularity/formalization/PolynomialParameterChart.lean`.

SHA-256: `77fa77bfce3b270fe3388ffc70e3567caac9d9bea62baf70fe4cc1d604c01559`; 15 rows.

Namespace: `Komorebi.PolynomialParameterChart`

`chartGraph_iff` (theorem; proof_source)

`chartPhi` (def; proof_source)

`chartPoint` (def; proof_source)

`chartPoint_congruent` (theorem; proof_source)

`chartPoint_congruent_iff` (theorem; proof_source)

`chartPoint_free` (theorem; proof_source)

`chartPoint_graph` (theorem; proof_source)

`chartPoint_root` (theorem; proof_source)

`chartPoint_spec` (theorem; proof_source)

`chartPoint_unique` (theorem; proof_source)

`chartTarget` (def; proof_source)

`continuous_chartPhi` (theorem; proof_source)

`continuous_chartPoint` (theorem; proof_source)

`continuous_chartTarget` (theorem; proof_source)

`parameterHomeomorph` (def; proof_source)

PolynomialParameterPath

Source: `polynomial_regularity/formalization/PolynomialParameterPath.lean`.

SHA-256: `8ce7df0bc47d4207bd6a6ad1f19afb82343efeb9573de8c415d16f6d92ec39ff`; 6 rows.

Namespace: `Komorebi.PolynomialParameterPath`

`exists_parameter_path` (theorem; proof_source)

`parameterPath` (def; proof_source)

`parameterPath_compatible` (theorem; proof_source)

`parameterPath_precision` (theorem; proof_source)

`parameterPath_tendsto` (theorem; proof_source)

`parameterPath_zero` (theorem; proof_source)

PolynomialParameterSystem

Source: `polynomial_regularity/formalization/PolynomialParameterSystem.lean`.

SHA-256: `ec8fc723a7fd5a6c1e945c7dab27a20c4a5e42e6db5ab7fe0fb981b2f2e4ee70`; 21 rows.

Namespace: `Komorebi.PolynomialParameterSystem`

`augmented` (def; proof_source)

`augmented_free` (theorem; proof_source)

`augmented_jacobian_surjective` (theorem; proof_source)

`augmented_solved` (theorem; proof_source)

`continuous_freePart` (theorem; proof_source)

`continuous_join` (theorem; proof_source)

`continuous_solvedPart` (theorem; proof_source)

`digitJacobian_augmented` (theorem; proof_source)

`digitJacobian_join_zero` (theorem; proof_source)

`evalSystem_augmented` (theorem; proof_source)

`freePart` (def; proof_source)

`freePart_join` (theorem; proof_source)

`join` (def; proof_source)

`join_free` (theorem; proof_source)

`join_parts` (theorem; proof_source)

`join_solved` (theorem; proof_source)

`padicEval_augmented` (theorem; proof_source)

`padicEval_augmented_iff` (theorem; proof_source)

`solvedJacobian` (def; proof_source)

`solvedPart` (def; proof_source)

`solvedPart_join` (theorem; proof_source)

PolynomialPrescribedCounts

Source: `polynomial_regularity/formalization/PolynomialPrescribedCounts.lean`.

SHA-256: a7db903a0ba00e05d84059639c78df0625635a946b603775e23768fafbad903c; 8 rows.

Namespace: Komorebi.PolynomialPrescribedCounts

digitAddTorsor (def; proof_source)

digit_card (theorem; proof_source)

liftAddTorsor (def; proof_source)

liftEquivKernel (def; proof_source)

lift_card (theorem; proof_source)

lift_card_eq_kernel (theorem; proof_source)

obstruction_card (theorem; proof_source)

obstruction_finrank (theorem; proof_source)

PolynomialPrescribedIntegerTower

Source: polynomial_regularity/formalization/PolynomialPrescribedIntegerTower.lean.

SHA-256: d0a11598d31484d7e5ca7a34b26b7bdfd7b71de52691b2dafbc213f52f1bdbe5; 3 rows.

Namespace: Komorebi.PolynomialPrescribedIntegerTower

exists_prescribed_tower (theorem; proof_source)

exists_prescribed_tower_strong (theorem; proof_source)

tower_solvedJacobian_eq (theorem; proof_source)

PolynomialPrescribedLifts

Source: polynomial_regularity/formalization/PolynomialPrescribedLifts.lean.

SHA-256: af964eafecd3d84148aba6baa5c7ade302051f97bff40b09645ddd9b2309dc6b; 6 rows.

Namespace: Komorebi.PolynomialPrescribedLifts

PrescribedLifts (abbrev; proof_source)

freePart_digitResidue (theorem; proof_source)

prescribedDigitEquivLifts (def; proof_source)

prescribedDigitEquivLifts_apply (theorem; proof_source)

prescribedDigitEquivRestricted (def; proof_source)

restrictedDigitEquivLifts (def; proof_source)

PolynomialPrescribedObstruction

Source: polynomial_regularity/formalization/PolynomialPrescribedObstruction.lean.

SHA-256: 4e1a7312fe5dbfd49762bf54022568e3321224111b0314b09d6f754eaabde140; 5 rows.

Namespace: Komorebi.PolynomialPrescribedObstruction

finite_prescribed_lift_count (theorem; proof_source)

finite_prescribed_obstruction (theorem; proof_source)

prescribed_lift_iff_obstruction_zero (theorem; proof_source)

prescribed_obstruction_finite (theorem; proof_source)

`prescribed_residue_exists_iff` (theorem; proof_source)

PolynomialPrescribedPadicTower

Source: `polynomial_regularity/formalization/PolynomialPrescribedPadicTower.lean`.

SHA-256: `ccab254952dd2648be03304011fec384c8eb479b468da8c36bebcc9a16e6882f`; 4 rows.

Namespace: `Komorebi.PolynomialPrescribedPadicTower`

`exists_prescribed_padic_root` (theorem; proof_source)

`exists_prescribed_padic_root_with_tower` (theorem; proof_source)

`exists_root_with_prescribed_path` (theorem; proof_source)

`exists_unique_prescribed_padic_root` (theorem; proof_source)

PolynomialPrescribedStep

Source: `polynomial_regularity/formalization/PolynomialPrescribedStep.lean`.

SHA-256: `248587ef2bb1663331fc0163385df5144400bae44c274fa0f3d6b10c730ed34d`; 7 rows.

Namespace: `Komorebi.PolynomialPrescribedStep`

`obstruction_zero` (theorem; proof_source)

`one_step_lift` (theorem; proof_source)

`prescribed_lift_card` (theorem; proof_source)

`solvedJacobian_eq` (theorem; proof_source)

`solvedJacobian_shift_eq` (theorem; proof_source)

`solved_dimension` (theorem; proof_source)

`solved_rank` (theorem; proof_source)

PolynomialPrescribedSystem

Source: `polynomial_regularity/formalization/PolynomialPrescribedSystem.lean`.

SHA-256: `87d8d3542994ae04a705ea6ba282ce00adda6d64fc1c61573c0b7d3592fb9978`; 10 rows.

Namespace: `Komorebi.PolynomialPrescribedSystem`

`PrescribedDigits` (abbrev; proof_source)

`PrescribedObstruction` (abbrev; proof_source)

`digitJacobian_join` (theorem; proof_source)

`obstruction` (def; proof_source)

`parameterJacobian` (def; proof_source)

`parameterResidual` (def; proof_source)

`prescribed_digits_finite` (theorem; proof_source)

`prescribed_digits_nonempty_iff` (theorem; proof_source)

`prescribed_equation_iff` (theorem; proof_source)

`solvedJacobian` (def; proof_source)

PolynomialPrescribedTowerLimit

Source: `polynomial_regularity/formalization/PolynomialPrescribedTowerLimit.lean`.

SHA-256: `4559f09410a4d76900bf4670747b83c8dbd5648a7f080856b8014d885c5e81d2`; 1 rows.

Namespace: `Komorebi.PolynomialPrescribedTowerLimit`

`prescribed_tower_limit` (theorem; proof_source)

PolynomialPrescribedTowerUnique

Source: `polynomial_regularity/formalization/PolynomialPrescribedTowerUnique.lean`.

SHA-256: `0c060ec7a92f7ca84bb1970fc1e85340fc64f438813fde33a054d215a98399c1`; 4 rows.

Namespace: `Komorebi.PolynomialPrescribedTowerUnique`

`prescribed_root_eq_chartPoint` (theorem; proof_source)

`prescribed_root_unique` (theorem; proof_source)

`prescribed_tower_endpoints_unique` (theorem; proof_source)

`solvedJacobian_square` (theorem; proof_source)

PolynomialRegularityIntegral

Source: `polynomial_regularity/formalization/PolynomialRegularityIntegral.lean`.

SHA-256: `ef886787a4cbb17a992c554f1ce6258abdc17ac045f45c02cfb0fe66f4d35c6d`; 11 rows.

Namespace: `Komorebi.PolynomialRegularityIntegral`

`eval_pderiv_rationalSystem` (theorem; proof_source)

`eval_rationalSystem` (theorem; proof_source)

`exists_analytic_integral_implicit` (theorem; proof_source)

`parameterJacobian_rationalSystem` (theorem; proof_source)

`rationalPoint` (def; proof_source)

`rationalPoint_freePart` (theorem; proof_source)

`rationalPoint_join` (theorem; proof_source)

`rationalPoint_solvedPart` (theorem; proof_source)

`rationalSystem` (def; proof_source)

`rationalSystem_root` (theorem; proof_source)

`solvedJacobian_rationalSystem` (theorem; proof_source)

PolynomialRootMotion

Source: `polynomial_regularity/formalization/PolynomialRootMotion.lean`.

SHA-256: `76a6b1a6cdeab1306809881a58eccb6885911c7f2dbb4dd10b92e874da4b367a`; 4 rows.

Namespace: `Komorebi.PolynomialRootMotion`

`exists_perturbed_root_certificate` (theorem; proof_source)

`exists_root_from_original_preconditioner` (theorem; proof_source)

`exists_root_motion` (theorem; proof_source)

`exists_unique_root_motion` (theorem; proof_source)

PolynomialRootStability

Source: `polynomial_regularity/formalization/PolynomialRootStability.lean`.

SHA-256: `77eb7db2cc0e93d3f57f0248cf7e19f30932416441caa369f625f15a2869a2db`; 5 rows.

Namespace: `Komorebi.PolynomialRootStability`

`exists_nearby_perturbed_root` (theorem; proof_source)

`exists_root_and_unique_perturbation` (theorem; proof_source)

`exists_unique_nearby_perturbed_root` (theorem; proof_source)

`integer_congruent_of_common_padic` (theorem; proof_source)

`perturbed_roots_congruent` (theorem; proof_source)

PolynomialSmoothCounts

Source: `polynomial_regularity/formalization/PolynomialSmoothCounts.lean`.

SHA-256: `bb4d79e92006e03f8bff12b3faa7eb5e87d9b0a25b5a7cf5265f16cfd670fd35`; 5 rows.

Namespace: `Komorebi.PolynomialSmoothCounts`

`exists_regular_tower` (theorem; proof_source)

`smooth_dimension` (theorem; proof_source)

`smooth_lift_card` (theorem; proof_source)

`smooth_obstruction_card` (theorem; proof_source)

`smooth_rank` (theorem; proof_source)

PolynomialSurjectiveStep

Source: `polynomial_regularity/formalization/PolynomialSurjectiveStep.lean`.

SHA-256: `d66d6635f63ca82be65e2a86aa55c2c4dc486796e6a3a1222fa039be162e6417`; 3 rows.

Namespace: `Komorebi.PolynomialSurjectiveStep`

`digitJacobian_eq` (theorem; proof_source)

`digitJacobian_shift_eq` (theorem; proof_source)

`one_step_lift` (theorem; proof_source)

PolynomialTaylor

Source: `polynomial_regularity/formalization/PolynomialTaylor.lean`.

SHA-256: `192667cea6c00131792d164ef6d802a689d4a34f8c465235a42d9319c57e699b`; 9 rows.

Namespace: `Komorebi.PolynomialTaylor`

`directional` (def; proof_source)

`directional_C` (theorem; proof_source)

`directional_X` (theorem; proof_source)

`directional_add` (theorem; proof_source)

`directional_mul` (theorem; proof_source)

`eval_dvd_of_dvd` (theorem; proof_source)

`pderiv_eval_dvd_of_dvd` (theorem; proof_source)

`taylor_dvd` (theorem; proof_source)

`taylor_step_dvd` (theorem; proof_source)

PolynomialTaylorDisplacement

Source: `polynomial_regularity/formalization/PolynomialTaylorDisplacement.lean`.

SHA-256: `a283ae6bbcfb681a386165e562130765d5601d6c02a200010040174711e01e9d`; 1 rows.

Namespace: `Komorebi.PolynomialTaylorDisplacement`

`taylor_remainder_dvd` (theorem; proof_source)

PolynomialTaylorRing

Source: `polynomial_regularity/formalization/PolynomialTaylorRing.lean`.

SHA-256: `344502b0ee6d50e602ae1a23a2b6685c3992a08a6366c029b4e64d2c1f9d1c32`; 11 rows.

Namespace: `Komorebi.PolynomialTaylorRing`

`eval_dvd_of_dvd` (theorem; proof_source)

`eval_linearPart` (theorem; proof_source)

`eval_scaled_bind` (theorem; proof_source)

`eval_zero_linearPart` (theorem; proof_source)

`exists_eval_scaled_remainder` (theorem; proof_source)

`exists_scaled_remainder` (theorem; proof_source)

`linearPart` (def; proof_source)

`linearPart_C` (theorem; proof_source)

`linearPart_X` (theorem; proof_source)

`linearPart_add` (theorem; proof_source)

`linearPart_mul` (theorem; proof_source)

PolynomialTowerControls

Source: `polynomial_regularity/formalization/PolynomialTowerControls.lean`.

SHA-256: `369da958792083c935d91b72f781b3a1ee624c50bfddb7f2500404248aedbb2b`; 10 rows.

Namespace: `Komorebi.PolynomialTowerControls`

`linearMinusTwo` (def; positive_control)

`linear_shift_no_extra_precision` (theorem; positive_control)

`linear_shift_nonsingular_lower_root` (theorem; positive_control)

`projectionSystem` (def; positive_control)

`projection_jacobian_surjective` (theorem; positive_control)

`projection_two_padic_roots` (theorem; positive_control)

`squareMinusFive` (def; positive_control)

`square_minus_five_lower_steps` (theorem; positive_control)

`square_minus_five_no_mod_eight` (theorem; positive_control)

`square_minus_five_no_tower` (theorem; positive_control)

PolynomialWeightedInverse

Source: `polynomial_regularity/formalization/PolynomialWeightedInverse.lean`.

SHA-256: `f809cd360810e51a7179d27080643bf3827066cb9ddc8a34deaaed74e814c1b8`; 17 rows.

Namespace: `Komorebi.PolynomialWeightedInverse`

`bounded_inverse_iff_certificate` (theorem; proof_source)

`certificate_of_integral_inverse` (theorem; proof_source)

`eval_precondition` (theorem; proof_source)

`integral_inverse_iff_certificate` (theorem; proof_source)

`integral_inverse_of_certificate` (theorem; proof_source)

`jacobian` (def; proof_source)

`jacobian_precondition` (theorem; proof_source)

`mulVec_injective_of_scalar_certificate` (theorem; proof_source)

`mulVec_preserves_dvd` (theorem; proof_source)

`padic_mulVec_injective` (theorem; proof_source)

`precondition` (def; proof_source)

`precondition_root_iff` (theorem; proof_source)

`rationalMatrix` (def; proof_source)

`rationalMatrix_injective` (theorem; proof_source)

`rationalMatrix_isUnit_of_certificate` (theorem; proof_source)

`rationalMatrix_mul` (theorem; proof_source)

`rationalMatrix_scalar` (theorem; proof_source)

PolynomialWeightedLift

Source: `polynomial_regularity/formalization/PolynomialWeightedLift.lean`.

SHA-256: `8b30f4b2015e6f12cabdb123972ef5b40378034c5c3b8f7947336a2307148845`; 5 rows.

Namespace: `Komorebi.PolynomialWeightedLift`

`exists_root_of_normalization` (theorem; proof_source)

`exists_unique_weighted_root` (theorem; proof_source)

`exists_weighted_root` (theorem; proof_source)

`exists_weighted_root_certificate` (theorem; proof_source)

`shiftedPoint` (def; proof_source)

PolynomialWeightedNormalization

Source: polynomial_regularity/formalization/PolynomialWeightedNormalization.lean.

SHA-256: 3b7ac03ae8c15ff0150f4cc98fc5ced67f285e16ec659400dcddec5585614411; 2 rows.

Namespace: Komorebi.PolynomialWeightedNormalization

exists_normalization (theorem; proof_source)

precondition_directional (theorem; proof_source)

ResidueLiftDigits

Source: polynomial_regularity/formalization/ResidueLiftDigits.lean.

SHA-256: 378dadff08814a05565baea228042017e58753f5a490d398655d4c95df8fb6d8; 10 rows.

Namespace: Komorebi.ResidueLiftDigits

ResidueFiber (abbrev; proof_source)

digitEquivResidue (def; proof_source)

digitEquivResidue_apply (theorem; proof_source)

digitResidue (def; proof_source)

digitResidue_injective (theorem; proof_source)

digitResidue_mem (theorem; proof_source)

digitResidue_surjective (theorem; proof_source)

next_modulus_dvd_val_sub_shift (theorem; proof_source)

residueFiber_card (theorem; proof_source)

shift_cast_eq_iff (theorem; proof_source)

PadicTowerLimitContract

Source: polynomial_towers/contracts/PadicTowerLimitContract.lean.

SHA-256: 6d81199904dec30c4e6d0ce263e7a69ae9a2bae7ffa70b4ab86ef5a6ad5a5c1e; 1 rows.

Namespace: Komorebi.PadicTowerLimitContract

explicit_vector_limit (theorem; statement_contract)

PolynomialPadicLiftContract

Source: polynomial_towers/contracts/PolynomialPadicLiftContract.lean.

SHA-256: acb5f14247e13111819e76a510950c6b5ab122cd4b01c570e662c6ec64f0133d; 2 rows.

Namespace: Komorebi.PolynomialPadicLiftContract

explicit_padic_root (theorem; statement_contract)

explicit_padic_root_with_tower (theorem; statement_contract)

PolynomialTowerContract

Source: polynomial_towers/contracts/PolynomialTowerContract.lean.

SHA-256: `b53f89c3151e95d920e2ad8e06c2e4bb11313f6ad113e61b905097980517421e`; 2 rows.

Namespace: `Komorebi.PolynomialTowerContract`

`explicit_jacobian` (theorem; statement_contract)

`explicit_polynomial_tower` (theorem; statement_contract)

SharedCorrectionContract

Source: `shared_correction/controls/SharedCorrectionContract.lean`.

SHA-256: `4aec4fc0ee27d21e0c98de6ebf66e2c331a278347a95b60c7988ec3e75eca89f`; 1 rows.

Namespace: `Komorebi.SharedCorrectionContract`

`explicit_quotient_contract` (theorem; statement_contract)

AffinePairControls

Source: `shared_correction/formalization/AffinePairControls.lean`.

SHA-256: `ef7312826db247cc709fbf0fde10b49fc014b6cbec97addf9421789b8787533e`; 5 rows.

Namespace: `Komorebi.AffinePairControls`

`composite_level_zero` (theorem; positive_control)

`incompatible_mod_four` (theorem; positive_control)

`modulus_one` (theorem; positive_control)

`oneId` (abbrev; positive_control)

`two_individual_lifts` (theorem; positive_control)

SharedCorrectionControls

Source: `shared_correction/formalization/SharedCorrectionControls.lean`.

SHA-256: `6c08654acf2c8b51863630c97618d779dd953084db61e3d9cf05b7cce9b2caac`; 11 rows.

Namespace: `Komorebi.SharedCorrectionControls`

`Two` (abbrev; positive_control)

`evaluateCoordinate` (def; positive_control)

`identical_maps_distinguish_targets` (theorem; positive_control)

`identity` (abbrev; positive_control)

`independent_joint_range_top` (theorem; positive_control)

`independent_targets_have_equal_labels` (theorem; positive_control)

`leftCoordinate` (abbrev; positive_control)

`rightCoordinate` (abbrev; positive_control)

`separately_solvable_but_incompatible` (theorem; positive_control)

`sequence_domain_common_finite` (theorem; positive_control)

`zero_left_identifies_all` (theorem; positive_control)

AllResidueStepAmplification

Source: strengthening/formalization/AllResidueStepAmplification.lean.

SHA-256: a16c419ab66d69068d54e3e47751c4684e21febd90232601e1c72c37e4177e2b; 3 rows.

Namespace: KomorebiPaperIX.AllResidueStepAmplification

normalized_precision (theorem; proof_source)

unnormalized_bound (theorem; proof_source)

unnormalized_precision (theorem; proof_source)

AllRowsControls

Source: strengthening/formalization/AllRowsControls.lean.

SHA-256: 5c0b80c3b45af5cd396e18b76235aa069f5ce7aaf4fa638fd428636955fe2031; 3 rows.

Namespace: KomorebiPaperIX.AllRowsControls

zero_step_every_order (theorem; positive_control)

zeroth_row_every_order (theorem; positive_control)

zeroth_row_every_step (theorem; positive_control)

ChargedDworkClosure

Source: strengthening/formalization/ChargedDworkClosure.lean.

SHA-256: d1db5c67f6ac3e73ca1914e7b6b394082aa01c2575a9ab3714b7fd7cc997831e; 24 rows.

Namespace: KomorebiPaperIX.ChargedDworkClosure

B (def; proof_source)

B_charge_shift (theorem; proof_source)

B_deriv (theorem; proof_source)

B_mul (theorem; proof_source)

B_zero_one (theorem; proof_source)

bounded (def; proof_source)

bounded_add (theorem; proof_source)

bounded_charge_shift (theorem; proof_source)

bounded_deriv (theorem; proof_source)

bounded_inverse (theorem; proof_source)

bounded_middle (theorem; proof_source)

bounded_mono (theorem; proof_source)

bounded_mul (theorem; proof_source)

bounded_nat_mul (theorem; proof_source)

bounded_neg (theorem; proof_source)

bounded_sub (theorem; proof_source)

bounded_succ (theorem; proof_source)

bounded_sum (theorem; proof_source)

bounded_zero (theorem; proof_source)

`flat` (def; proof_source)

`flat_charge_shift` (theorem; proof_source)

`flat_inverse` (theorem; proof_source)

`inverse_defect` (theorem; proof_source)

`iterate_B_deriv` (theorem; proof_source)

CompactSelfIsometry

Source: `strengthening/formalization/CompactSelfIsometry.lean`.

SHA-256: `3a81b435e899be582f1bcec59c561de1c57b7d7a7c6150a3eeafbe2ca8723c44`; 3 rows.

Namespace: `KomorebiPaperIX.CompactSelfIsometry`

`denseRange` (theorem; proof_source)

`iterate_isometry` (theorem; proof_source)

`surjective` (theorem; proof_source)

DensePadicIsometry

Source: `strengthening/formalization/DensePadicIsometry.lean`.

SHA-256: `5ff66895f2ef98dad15abe1af104aa00285c05466d87ba384f7f5af795fbc136`; 15 rows.

Namespace: `KomorebiPaperIX.DensePadicIsometry`

`existsUnique_extensionEquiv` (theorem; proof_source)

`extension` (def; proof_source)

`extensionEquiv` (def; proof_source)

`extensionEquiv_apply` (theorem; proof_source)

`extensionEquiv_nat` (theorem; proof_source)

`extensionEquiv_unique` (theorem; proof_source)

`extensionEquiv_unique_continuous` (theorem; proof_source)

`extension_continuous` (theorem; proof_source)

`extension_isometry` (theorem; proof_source)

`extension_nat` (theorem; proof_source)

`extension_surjective` (theorem; proof_source)

`extension_unique` (theorem; proof_source)

`onNaturalRange` (def; proof_source)

`onNaturalRange_isometry` (theorem; proof_source)

`onNaturalRange_nat` (theorem; proof_source)

DyadicFourAllRows

Source: `strengthening/formalization/DyadicFourAllRows.lean`.

SHA-256: `e90316dce8f7976fed0333f72fbea9043542c0b6ea188f9acfa8a6084cffa3a4`; 18 rows.

Namespace: `KomorebiPaperIX.DyadicFourAllRows`

`B` (abbrev; proof_source)

`CoeffFlat` (def; proof_source)

`Flat` (def; proof_source)

`actual_packet_flat` (theorem; proof_source)

`base_step_bound` (theorem; proof_source)

`decode_B` (theorem; proof_source)

`decode_B_iter` (theorem; proof_source)

`difference_coe` (theorem; proof_source)

`flat_encode_iff` (theorem; proof_source)

`flat_reciprocal` (theorem; proof_source)

`lifted_flat` (theorem; proof_source)

`normalized_bound` (theorem; proof_source)

`normalized_precision` (theorem; proof_source)

`packed_difference_support` (theorem; proof_source)

`packed_flat_of_kummer` (theorem; proof_source)

`source_flat` (theorem; proof_source)

`unnormalized_bound` (theorem; proof_source)

`unnormalized_precision` (theorem; proof_source)

DyadicFourFinalStrengthening

Source: `strengthening/formalization/DyadicFourFinalStrengthening.lean`.

SHA-256: `23e7b814ac843f882fc99c0429bdac6b4542f041a43cb0ba8e92b601a2dcb179`; 11 rows.

Namespace: `KomorebiPaperIX.DyadicFourFinalStrengthening`

`all_rows_half_period` (theorem; proof_source)

`all_rows_half_period_degree` (theorem; proof_source)

`all_rows_half_period_precision` (theorem; proof_source)

`all_rows_period` (theorem; proof_source)

`all_rows_period_degree` (theorem; proof_source)

`all_rows_period_precision` (theorem; proof_source)

`normalized_period` (theorem; proof_source)

`normalized_step_bound` (theorem; proof_source)

`normalized_step_precision` (theorem; proof_source)

`unnormalized_step_bound` (theorem; proof_source)

`unnormalized_step_precision` (theorem; proof_source)

DyadicFourIsometry

Source: `strengthening/formalization/DyadicFourIsometry.lean`.

SHA-256: `22ee30fdfe665602b491ef9ff43b3aa8fe2bacfc503724e19e6623b227ef4da8`; 19 rows.

Namespace: `KomorebiPaperIX.DyadicFourIsometry`

`existsUnique_isometry` (theorem; proof_source)

`isometry` (def; proof_source)

`isometry_continuous` (theorem; proof_source)

`isometry_dist` (theorem; proof_source)

`isometry_nat` (theorem; proof_source)

`isometry_surjective` (theorem; proof_source)

`isometry_zero` (theorem; proof_source)

`originalExtension` (def; proof_source)

`originalExtension_continuous` (theorem; proof_source)

`originalExtension_dist` (theorem; proof_source)

`originalExtension_injective` (theorem; proof_source)

`originalExtension_nat` (theorem; proof_source)

`originalExtension_range` (theorem; proof_source)

`samples` (def; proof_source)

`samples_coe` (theorem; proof_source)

`samples_dist` (theorem; proof_source)

`unique_continuous` (theorem; proof_source)

`unique_original_preimage` (theorem; proof_source)

`unique_preimage` (theorem; proof_source)

DyadicFourLeastPeriods

Source: `strengthening/formalization/DyadicFourLeastPeriods.lean`.

SHA-256: `1073c2debcd4f11ac2478b89549c18682cc05683c078d9378d86991da23d7286`; 8 rows.

Namespace: `KomorebiPaperIX.DyadicFourLeastPeriods`

`exact_displacement` (theorem; proof_source)

`least_packet_period` (theorem; proof_source)

`normalized_exact_displacement` (theorem; proof_source)

`originalBonus` (def; proof_source)

`originalBonus_values` (theorem; proof_source)

`original_degree_exact_displacement` (theorem; proof_source)

`period_iff_dvd` (theorem; proof_source)

`precision_iff_dvd` (theorem; proof_source)

DyadicFourPrefix

Source: `strengthening/formalization/DyadicFourPrefix.lean`.

SHA-256: `3fc459e037e333e6c3680753ac591502061ef0b2e2a83073f5f7892335ef5ca2`; 49 rows.

Namespace: `KomorebiPaperIX.DyadicFourPrefix`

`bonus` (def; proof_source)

`coeff_0_1` (theorem; proof_source)

`coeff_0_2` (theorem; proof_source)

`coeff_1_1` (theorem; proof_source)

`coeff_1_2` (theorem; proof_source)

`coeff_2_1` (theorem; proof_source)

`coeff_2_2` (theorem; proof_source)

`coeff_2_3` (theorem; proof_source)

`coeff_2_4` (theorem; proof_source)

`coeff_2_5` (theorem; proof_source)

`coeff_2_6` (theorem; proof_source)

`coeff_3_1` (theorem; proof_source)

`coeff_3_2` (theorem; proof_source)

`coeff_3_3` (theorem; proof_source)

`coeff_3_4` (theorem; proof_source)

`first_nonzero` (theorem; proof_source)

`first_valuation` (theorem; proof_source)

`mahlerCoeff` (def; proof_source)

`mahler_0_1` (theorem; proof_source)

`mahler_0_2` (theorem; proof_source)

`mahler_1_1` (theorem; proof_source)

`mahler_1_2` (theorem; proof_source)

`mahler_2_1` (theorem; proof_source)

`mahler_2_2` (theorem; proof_source)

`mahler_2_3` (theorem; proof_source)

`mahler_2_4` (theorem; proof_source)

`mahler_2_5` (theorem; proof_source)

`mahler_2_6` (theorem; proof_source)

`mahler_3_1` (theorem; proof_source)

`mahler_3_2` (theorem; proof_source)

`mahler_3_3` (theorem; proof_source)

`mahler_3_4` (theorem; proof_source)

`mahler_valuation_0_1` (theorem; proof_source)

`mahler_valuation_0_2` (theorem; proof_source)

`mahler_valuation_1_1` (theorem; proof_source)

`mahler_valuation_1_2` (theorem; proof_source)

`mahler_valuation_2_1` (theorem; proof_source)

`mahler_valuation_2_2` (theorem; proof_source)

`mahler_valuation_2_3` (theorem; proof_source)

`mahler_valuation_2_4` (theorem; proof_source)

`mahler_valuation_2_5` (theorem; proof_source)

`mahler_valuation_2_6` (theorem; proof_source)

`mahler_valuation_3_1` (theorem; proof_source)

`mahler_valuation_3_2` (theorem; proof_source)

`mahler_valuation_3_3` (theorem; proof_source)

`mahler_valuation_3_4` (theorem; proof_source)

`precision_of_order_le` (theorem; proof_source)

`tail_precision` (theorem; proof_source)

`valuation_two_power_odd` (theorem; proof_source)

DyadicFourSharpening

Source: `strengthening/formalization/DyadicFourSharpening.lean`.

SHA-256: `eda98f64f8525d210569db34177a5795f9745939dc59d9cd98887eac52df57ac`; 9 rows.

Namespace: `KomorebiPaperIX.DyadicFourSharpening`

`all_rows_even` (theorem; proof_source)

`factorial_difference_bound` (theorem; proof_source)

`factorial_difference_precision` (theorem; proof_source)

`factorial_even` (theorem; proof_source)

`half_period` (theorem; proof_source)

`half_period_degree` (theorem; proof_source)

`half_period_precision` (theorem; proof_source)

`normalized_period` (theorem; proof_source)

`unnormalized_difference_factorial` (theorem; proof_source)

DyadicIsometryControls

Source: `strengthening/formalization/DyadicIsometryControls.lean`.

SHA-256: `b4f813406c69b1d9c37766c388be658d82e900f2efdccc864a5d41458b6625b3`; 6 rows.

Namespace: `KomorebiPaperIX.DyadicIsometryControls`

`address_zero_at_one` (theorem; positive_control)

`address_zero_at_two` (theorem; positive_control)

`address_zero_inverse_at_thirty_four` (theorem; positive_control)

`address_zero_not_additive` (theorem; positive_control)

`original_address_zero_ne_zero` (theorem; positive_control)

`original_address_zero_not_surjective` (theorem; positive_control)

DyadicLogThresholds

Source: `strengthening/formalization/DyadicLogThresholds.lean`.

SHA-256: `3c5230cbad55877095ab8a5b45cb99082ed724c1b0720f90d77fc5f9d4917eae`; 5 rows.

Namespace: `KomorebiPaperIX.DyadicLogThresholds`

`five_add_log_le` (theorem; proof_source)

`log_two_succ_le` (theorem; proof_source)

`offset_log_bound_of_le` (theorem; proof_source)

`three_add_log_le` (theorem; proof_source)

`two_add_log_le` (theorem; proof_source)

DyadicPacketRational

Source: `strengthening/formalization/DyadicPacketRational.lean`.

SHA-256: `7a29429898faaa4d9d0d8d9ee931be6ba38644b061e967d6c36a69a3d49dc9e1`; 7 rows.

Namespace: `KomorebiPaperIX.DyadicPacketRational`

`exact_displacement` (theorem; proof_source)

`norm_displacement` (theorem; proof_source)

`norm_le_one` (theorem; proof_source)

`norm_sub` (theorem; proof_source)

`packet` (def; proof_source)

`packet_sub` (theorem; proof_source)

`packet_zero` (theorem; proof_source)

DyadicSharpTower

Source: `strengthening/formalization/DyadicSharpTower.lean`.

SHA-256: `70f43e0621368af3bb3871e102deb238e68fb610fd73d81cae1486d7a95c3a2b`; 27 rows.

Namespace: `KomorebiPaperIX.DyadicSharpTower`

`bernoulli_dyadic_valuation` (theorem; proof_source)

`bernoulli_even_nonzero` (theorem; proof_source)

`beta_bernoulli` (theorem; proof_source)

`cosh_derivative` (theorem; proof_source)

`cothNormalized` (def; proof_source)

`coth_times_sinh` (theorem; proof_source)

`dyadic_supercongruence_row_weight` (theorem; proof_source)

`dyadic_supercongruence_sharp` (theorem; proof_source)

`expAt_derivative` (theorem; proof_source)

`first_omitted_error` (theorem; proof_source)

`first_omitted_error_valuation` (theorem; proof_source)

`hyperbolic_identity` (theorem; proof_source)

`inverse_eq_coth_derivative` (theorem; proof_source)

`kappa` (def; proof_source)

kappaPadic (def; proof_source)

kappaPadic_coe (theorem; proof_source)

kappaPadic_divisible (theorem; proof_source)

kappaPadic_norm (theorem; proof_source)

kappa_bernoulli (theorem; proof_source)

kappa_nonzero (theorem; proof_source)

kappa_valuation (theorem; proof_source)

odd_factor_valuation (theorem; proof_source)

sinh_derivative (theorem; proof_source)

summandPadic_divisible (theorem; proof_source)

summandPadic_kappa (theorem; proof_source)

summand_kappa (theorem; proof_source)

summand_valuation (theorem; proof_source)

DyadicStepAmplification

Source: strengthening/formalization/DyadicStepAmplification.lean.

SHA-256: e5d7937d20e2664578d066e4df1c843190da3018a2677ccd5dd9f72e117f1eab; 5 rows.

Namespace: KomorebiPaperIX.DyadicStepAmplification

normalized_bound (theorem; proof_source)

normalized_precision (theorem; proof_source)

prime_power_step_bound (theorem; proof_source)

unnormalized_factorial_bound (theorem; proof_source)

unnormalized_factorial_precision (theorem; proof_source)

ExactPeriodClassification

Source: strengthening/formalization/ExactPeriodClassification.lean.

SHA-256: 86fb5cc83b77e3889d004503916aaa12c981d9dc6e9566c544a0136664856eb5; 4 rows.

Namespace: KomorebiPaperIX.ExactPeriodClassification

IsPeriod (def; proof_source)

isPeriod_iff_dvd (theorem; proof_source)

least_period (theorem; proof_source)

precision_iff_dvd (theorem; proof_source)

ExactPeriodControls

Source: strengthening/formalization/ExactPeriodControls.lean.

SHA-256: 13237d0c0b681c160c1c1dcf4e1158e156ad3f28378d800dc68196f54ca6f3f4; 6 rows.

Namespace: KomorebiPaperIX.ExactPeriodControls

address_three_mod1024_least (theorem; positive_control)

`address_two_mod1024_period96` (theorem; positive_control)

`address_two_mod32` (theorem; positive_control)

`address_two_mod64_least` (theorem; positive_control)

`address_two_period_one_fails_mod64` (theorem; positive_control)

`positive_shift_nonzero` (theorem; positive_control)

NewtonDominance

Source: `strengthening/formalization/NewtonDominance.lean`.

SHA-256: `57973ae0db7d02891a2d2048d5b68530252d47a354cb7569edb4922f7b73da05`; 11 rows.

Namespace: `KomorebiPaperIX.NewtonDominance`

`choose_precision` (lemma; proof_source)

`displacement_expansion` (lemma; proof_source)

`exact_displacement` (theorem; proof_source)

`first_difference_tail` (lemma; proof_source)

`precision_add` (lemma; proof_source)

`precision_mono` (lemma; proof_source)

`precision_mul` (lemma; proof_source)

`precision_mul_nat` (lemma; proof_source)

`precision_sum` (lemma; proof_source)

`strict_dominance` (lemma; proof_source)

`translated_tail` (lemma; proof_source)

RationalPadicEmbedding

Source: `strengthening/formalization/RationalPadicEmbedding.lean`.

SHA-256: `234880a6a053a143265ab9b474f489ff7f4107356135b7038518b757d617f6dc`; 4 rows.

Namespace: `KomorebiPaperIX.RationalPadicEmbedding`

`norm_le_one` (theorem; proof_source)

`samples` (def; proof_source)

`samples_coe` (theorem; proof_source)

`samples_dist` (theorem; proof_source)

StepAmplification

Source: `strengthening/formalization/StepAmplification.lean`.

SHA-256: `c4f9ecbcf2cffa821a25db744f8b52e2cb88c98cbafd89014f266f5286b47900`; 5 rows.

Namespace: `KomorebiPaperIX.StepAmplification`

`binomial_step_precision` (theorem; proof_source)

`difference_compose` (theorem; proof_source)

`offset_displacement` (theorem; proof_source)

`repeated_offset` (theorem; proof_source)

`repeated_precision` (theorem; proof_source)

StepAmplificationExamples

Source: `strengthening/formalization/StepAmplificationExamples.lean`.

SHA-256: `286d99f1bbe2876a6339f0672daf49dd9d20b3f2d3cd57ac19def50c3bad68ac`; 4 rows.

Namespace: `KomorebiPaperIX.StepAmplificationExamples`

`actual_first_difference` (theorem; positive_control)

`actual_first_difference_valuation` (theorem; positive_control)

`actual_second_coefficient` (theorem; positive_control)

`no_uniform_extra_digit` (theorem; positive_control)