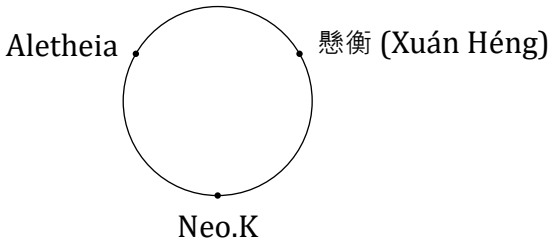
Replaying large geometric certificates without trusting their producer

Integer intervals, an exact root and a decodable certificate format, on a lower-bound programme for Lebesgue's covering problem



Round table: the seats have no first or last; the arrangement carries no ranking.

2026-10-18

Abstract

A computer-assisted lower bound in convex geometry is a tree of boxes in a space of placements, each leaf closed by an inner polygon whose exact area exceeds a threshold; one cell of such a tree holds ten thousand leaves and two million stored vertices, and no one reads it. What makes it a proof is that someone who did not produce it can check it, and the check is worth only what is independent in it. We report such a replay of the certificates of a research programme aimed at the threshold 167/200 for Lebesgue's universal covering problem. The replay shares nothing with the producer but the mathematics and the certificate bytes: integer intervals at 2^-400 with outward rounding, its own exact root, its own trees, and a reading of the binary certificate format from its specification alone. It brackets the root of the computation to about 10^-90 and shows the stored binary64 angle roots falling short of the exact ones; it checks 97,943 leaves in ten cells, 19,356,171 stored vertices among them, and certifies all 77 upper markers once each straight bridging edge of the hull gets its own normal, without which a uniform grid of normals loses first order in its spacing. On the way it finds that the programme's densities were a policy rather than a need, that its two further implementations shared a rule that keeps one contact point per direction — so that their thinnest leaf measured the rule's loss, not the geometry — that its MPFR backend directed its rounding at the outputs only, and that its frontier merge never checks the premise its theorem rests on. No bound on Lebesgue's constant is claimed: ten of 77 necessity cells are certified, and the programme's target is not.

Contents

0. What is claimed, and what is not 2
1. Introduction 3
2. An arithmetic that shares nothing with the producer's 5
3. The root 6
4. One cell, every leaf 7

5. The upper side: the 77 markers 8
6. Two more implementations, and one rule they share 10
7. A certificate format, read from its specification 11
8. Trees and frontiers 12
9. What is certified, and what is not 13
10. Every result, and the strength at which it holds 14
11. Corrections found on the way 15
12. Reproduction and attachments 16
13. Contributions 17
Acknowledgements 17
References 17
Appendix A. The figures file 18

Status of this document. Preprint, version 0.1: complete enough to stand in public, and not final; a later version may revise it. Not prepared for journal submission. Section 13 records who did which part.

0. What is claimed, and what is not

Every statement in this paper carries a status, given after it in italics. A statement with a proof carries a verdict on that proof. `PASS` and `PASS by cited theorem` are different verdicts and are never written as one: the first is a proof checked line by line, the second a proof one of whose steps holds because a published or classical theorem does, cited with its hypotheses checked. Every other statement carries one of six labels, and the labels are never merged:

label	meaning
`COMPUTED_HERE`	recomputed by the recomputation arm with independently written code — *exact* where the text says so (integer or rational arithmetic, or directed integer intervals), in floating point where it says *numerical*
`DERIVED_FROM_VERIFIED_INPUTS`	follows from statements already verified here, with no step of its own beyond combining them
`CITED_THEOREM`	someone else's published theorem, stated here and not re-proved
`INHERITED`	taken from the research programme's records and not verified here; every use is marked
`UNMEASURED`	well posed, and not established here
`FAILED`	attempted and refused: false as stated, or true only in the corrected form given

Where a run grades a statement of the programme `PARTIAL` — true of part of what it says, or true of something other than what it names — this paper says `PARTIAL` too, and names the part that fails.

Not claimed anywhere in this paper: any value or bound for Lebesgue's universal covering constant — the problem is open, and the programme's target $167/200 = 0.835$ is *not* certified, by the programme or here; that the programme's atlas of cells is complete — it is not; and anything about

the programme's second arithmetic backend (MPFR) beyond what its shipped records say, since that arithmetic is not present here. What this paper reports is a replay: which of the programme's certificates check in an arithmetic that shares nothing with the one that produced them, and what the replay found on the way.

1. Introduction

1.1 Certificates too large to read

A computer-assisted lower bound in convex geometry is a finite object standing in for a continuum. In the programme this paper examines, a lower bound for Lebesgue's universal covering constant a_{Leb} is to be written as a tree of boxes in a space of placements of finitely many test bodies, every leaf box closed by an inner polygon — a polygon whose vertices are verified to lie in bodies placed anywhere in the box, so that its exact area is a lower bound for the area of every hull in the box. If every leaf's polygon has area above the threshold, the bound follows (the companion paper [1], Theorem 8.6). One cell of the programme's atlas carries about ten thousand leaves and two million stored vertices; the atlas has 77 such cells (RUN-032, RUN-037). The published bracket for the constant is $0.8344 \le a_{\mathrm{Leb}} \le 0.8440935944$ (Mishra [4]; Gibbs [3]; their arXiv records checked in RUN-002), and the programme set out to raise the lower end to 0.835.

Nobody reads such a certificate. What makes it a proof is that someone else can check it — and the check is only worth what is independent in it. If the checker uses the producer's arithmetic library, the producer's candidate points, or the producer's tree, an error in any of them passes both. This paper reports a replay built to share nothing with the producer but the mathematics and the certificate's bytes: the recomputation's own arithmetic, its own root, its own trees, and a reading of the certificate format from its specification alone.

1.2 What the replay checks, and what it found

- §2 gives the arithmetic: integer intervals at 2^{-400} with outward rounding at every step, π by Machin's formula, sines and cosines from enclosed Taylor sums with their remainder, exact rational hulls and areas.
- §3 brackets the root of the whole computation — the a-priori radius and translation bounds — to width about 10^{-90}, and shows where the binary64 values the programme stored sit against it: the translation roots above, the angle roots below.
- §4 replays all 9,278 leaves of one cell and finds that the programme's densities are a policy, not a need.
- §5 certifies the upper side, all 77 marker configurations, and finds that the programme's outer polygons lose first order in the normal spacing where the hull has a straight bridging edge.
- §6 examines the programme's two further implementations and finds that both keep one contact point per direction, so that their thinnest leaf measures a rule's loss rather than the geometry.
- §7 decodes the RHCERT certificate format from its specification and checks every stored vertex of eight cells, and §8 the trees and frontiers that index them.
- §9 states what is certified, and what rests on arithmetic that is not here.

Nothing in the replay needs the programme to be right about anything but the mathematics it states.

1.3 The setting

The programme works at the threshold $T = 167/200$ with a master target $T_M = 1673/2000$ slightly above it. Its base family is the disc D of diameter one, fixed at the origin, and the Reuleaux triangle B_3 and pentagon B_5; its witness is the Reuleaux heptagon B_7. A placement of B_n is its centre (x_n, y_n) and an angle φ_n; the triangle's angle is fixed by the rotation of the whole configuration.

The a-priori root. Let $F(d)$ be the area of $\mathrm{conv}(D \cup \{x\})$ with $\|x\| = d$,

$$F(d) = \tfrac{1}{4}\Big(\pi - \arccos\tfrac{1}{2d}\Big) + \tfrac{1}{2}\sqrt{d^2 - \tfrac{1}{4}}, \qquad F'(d) = \frac{\sqrt{d^2 - 1/4}}{2d} > 0 \quad (d > \tfrac{1}{2}),$$

and $d_\star$ the root of $F(d_\star) = T_M$. A hull of area below T_M lies in the disc of radius $d_\star$. A placed B_n of circumradius R_n whose centre lies farther than

$$t_n = -R_n \cos\frac{\pi}{n} + \sqrt{R_n^2 \cos^2\frac{\pi}{n} - R_n^2 + d_\star^2}$$

from the origin has a vertex beyond $d_\star$, so every configuration that matters has $\|(x_n, y_n)\| \le t_n$. The base root is the box $0 \le x_3 \le t_3$, $0 \le y_3 \le \frac{\sqrt{3}}{2} t_3$ (the dihedral symmetry of the problem, quotiented), $0 \le \varphi_5 < 2\pi/5$, $|x_5|, |y_5| \le t_5$; the witness root is $|x_7|, |y_7| \le t_7$, $0 \le \varphi_7 < 2\pi/7$. *Status:* the formulas PASS — F and F' are the area and derivative of the hull of a disc and a point (the companion paper's $\Phi_{1/2}$), and t_n solves $t^2 + 2tR_n \cos(\pi/n) + R_n^2 = d_\star^2$ for the vertex nearest the radial direction (RUN-031).

Cells, lifts and markers. The base root is bisected along a fixed sequence of axes into cells. In 77 of them, the *necessity cells*, the programme found a base configuration whose hull area is below T, so the base alone cannot close them; each such configuration is a *marker*. The cells are numbered by their markers, 0 to 76; the programme's reference rounds, which closed one cell first, call that cell "cell 0", and it is the cell of marker 70. Each necessity cell is closed, if at all, by lifting: over every base box of the cell and every placement of B_7, a tree of joint boxes is grown until every leaf has an inner polygon of area above T. A cell whose tree is complete and whose leaves all carry such polygons is closed for the family $\{D, B_3, B_5, B_7\}$. The marker is the upper side: an *outer* polygon around the marker's hull, of area below T, certifies that the cell really needed the witness. At the end of the rounds examined, 66 of the 77 cells are geometrically complete — every leaf of a finished tree above T, in floating point — and the programme counts ten as certified or better, the same ten that §9 certifies here; $a_{\mathrm{Leb}} \ge 0.835$ would need all 77, and every other part of the space, certified. *Status:* the completeness count COMPUTED_HERE, numerical (RUN-035, RUN-039, RUN-040); the programme's count of its certified cells is its own, and §9 checks each of the ten (RUN-040).

1.4 Where this comes from, and how it was checked

The certificates, trees, ledgers and formats are the programme's — rounds 27 to 37 of its finite-closure line, whose methodology is Neo.K's and whose rounds Aletheia produced. The replay is the recomputation's (懸衡): the runs RUN-030 to RUN-040 of the recomputation tree pinned in the front matter, each with its gate log and each check drilled with planted faults. The programme's scripts were read only after the replay was computed, and never run. Each result below names the run that checked it.

2. An arithmetic that shares nothing with the producer's

2.1 The inner polygon

A leaf box fixes ranges for every placement parameter. For a placed Reuleaux body $B_n = \bigcap_j B(V_j, 1)$, the *common core* of the box is the set of points that lie in B_n for every placement in it. With the box's centre placement, corners V_j, and the motion radius $\delta = \sqrt{h_x^2 + h_y^2} + 2R_n \sin(h_\varphi/2)$ of its half-widths, the eroded body $\bigcap_j B(V_j, \rho)$, $\rho = 1 - \delta$, lies in the common core (the companion paper [1], Lemma 8.4 and Theorem 8.5). So a point p is accepted for B_n when $\|p - V_j\|^2 < \rho^2$ for every corner — decided as $(\|p - V_j\|^2)_{\text{hi}} < (\rho^2)_{\text{lo}}$, with the corners enclosed, and, on the exact boxes of §4, with the largest distance taken over each corner's rectangle of possible centres.

Proposition 2.1 (inner polygons). *If finitely many points each lie in the disc or in the common core of one body for every placement in the box, the exact area of their convex hull is a lower bound for the hull area of every configuration in the box.* The points lie in every such hull, and so does their convex hull. *Status:* PASS (RUN-030).

So a leaf certificate is a list of points, each tagged with the body whose core it is claimed to lie in, and a number — the exact area of their hull — that must exceed T. Everything else is how the points are chosen and how membership is decided. The replay chooses them as the programme's first backend does: at a *density* of N directions, the contact point of the disc and of each of the three cores in every direction, moved slightly toward its body's centre, and of these $4N$ points the vertices of their floating-point hull — the leaf's *candidates*.

2.2 The replay's arithmetic

Every real number in a core's definition is enclosed in an interval of integers at scale 2^{-400}, rounded outward at every operation: π, the circumradii R_n, each corner V_j, the motion radius δ. Sines and cosines are Taylor sums with every term enclosed and the Lagrange remainder added. π comes from Machin's arctangent formula, each series enclosed; a first version cut π after a fixed number of decimals, which was too coarse once the root inherits π's width (§3), and the second carries it as far as the root needs. An inverse cosine is a high-precision guess, accepted only after an enclosed cosine brackets it. The candidate points are exact binary floating-point numbers, so their hull, its shoelace area and every membership test are exact integer computations. *Status:* COMPUTED_HERE, exact (RUN-030, RUN-031).

The programme's own certificates were computed in the `mpmath` library's [6] interval type, with the candidate points rounded to 17-digit decimals. The replay uses neither. The only `mpmath` calls in the code the replay rests on are two high-precision guesses — for an inverse cosine, and for the root $d_\star$ of §3 — each accepted only after the replay's own enclosures bracket it. *Status:* COMPUTED_HERE — 33 functions read (RUN-030, RUN-033).

2.3 A verifier must refuse

A verifier that cannot fail measures nothing, so the replay was made to fail where it should. On the thinnest leaf of the first cell examined, the 3,170 candidate points pushed outward by the same 10^{-12} by which they were pulled in: none of them verifies. At 4,096 directions instead of the certificate's density, every point verifies but the polygon falls 1.80×10^{-7} short of T: not a pass. *Status:* COMPUTED_HERE (RUN-030).

The twelve thinnest leaves of that cell all certify, on the replay's own boxes and on the boxes the programme stored, with margins from 3.1949×10^{-8} to 4.0147×10^{-7}; on the five the programme had certified, the candidate, verified and hull-vertex counts are its own and the margins agree to 4.8×10^{-17}. On each of the twelve, at their high densities, the certified polygon lies between 0.99×10^{-8} and 1.41×10^{-8} below the exact hull of the cores, as Proposition 2.1 requires. So a leaf's *reference slack* — the area of the exact hull of its cores, less T, which the programme records for every leaf — is not its certificate's margin but the margin's ceiling. *Status:* `COMPUTED_HERE`, exact (RUN-030).

3. The root

Every box of every tree descends from the root of §1.3, so every certificate rests on $d_\star$ and the t_n. The programme stored them as binary64 numbers, and a stored root that falls short of the exact one leaves a sliver of the configuration space outside every box.

3.1 The exact root, bracketed

F increases for $d > \frac{1}{2}$, so the values of F at the two ends of an interval decide whether it brackets $d_\star$. The replay takes a high-precision guess and asserts, with two enclosed evaluations of F, that a narrow interval around it brackets the root; the t_n follow in interval arithmetic. It brackets $d_\star$, t_3, t_5 and t_7 to width about 10^{-90}, each inside the programme's own intervals, which are 3.1×10^{-61} wide for $d_\star$ — not the $O(10^{-63})$ its text says — and 3 to 4×10^{-64} for the t_n. *Status:* `COMPUTED_HERE`, exact (RUN-031).

3.2 Where the stored roots sit

In units of the last place of binary64, against the exact values — first for the binary64 root on which the replay's own floating-point trees were grown, then for the root the programme stored:

root	the replay's binary64 state	the programme's stored state
t_3 (x_3's end)	+7.01	+1.01
$\frac{\sqrt{3}}{2}t_3$ (y_3's end)	+6.18	+0.18
$2\pi/5$ (φ_5's end)	−0.22	−0.22
t_5	+8.77	+4.77
$2\pi/7$ (φ_7's end)	−0.32	−0.32
t_7	+7.39	+3.39

Every translation root lies above its exact value, in both states, so no translation is missed. Both angle roots fall short: $2 \cdot \mathrm{fl}(\pi/7)$ is 3.5×10^{-17} below $2\pi/7$, less than one unit of $\pi/7$. On the leaves at the top of the angle range, the enclosure of each stored value by its neighbouring floats is what covers that sliver: it is load-bearing there, not a hedge. The programme's later *intended* boxes — exact midpoints of the exact root, enclosed — close the sliver by construction, and every leaf of §4 is checked on them. *Status:* `COMPUTED_HERE`, exact (RUN-030, RUN-031).

3.3 Four brackets, nested

The programme bracketed the root three times — in its first interval backend, in a second implementation, and in a backend built on the MPFR library [5] — and the replay a fourth. They are not merely

overlapping, as the programme's text says; they are nested. The replay's brackets (about 10^{-90}) lie inside the second implementation's (2.8 to 4.0 × 10^{-73}), those inside the MPFR backend's (a dyadic interval 2.967 × 10^{-67} wide for $d_\star$, 3.1 × 10^{-67} to 4.2 × 10^{-67} for the t_n), and those inside the first backend's. The second implementation's lower replay does not use its own bracket, though: its script reads the first backend's root intervals. *Status:* `COMPUTED_HERE`, exact, for every containment that needs no MPFR (RUN-031, RUN-033, RUN-034).

The programme corrected one error here itself, and its cause reproduces: an ultra-narrow $d_\star$ interval cast to binary64 moves $d_\star$ by 1.70×10^{-17}, and the t_n computed from it land 2.43×10^{-17}, 1.90×10^{-17} and 1.80×10^{-17} above the first backend's intervals — all three outside. The correction calls the cast interval "about 10^{-64}" wide; that is the width of the t_n intervals, the $d_\star$ intervals being 3.1×10^{-61} and 2.8×10^{-73} wide. *Status:* `COMPUTED_HERE` (RUN-033).

4. One cell, every leaf

4.1 The tree

The first cell the programme closed is reached from the root by 40 bisections, along the programme's fixed sequence of axes (x_3, y_3, φ_5, x_5, y_5 split 7, 7, 9, 9 and 8 times), with no exceptions. Its witness tree is stored as a topology bit stream and a stream of axes, two bits per internal node in preorder, high bit first. The topology decodes to 18,555 nodes, 9,277 internal and 9,278 leaves, and both streams are, node for node, the tree the replay grows itself. The cell the programme stored sits within 6 units in the last place of the replay's binary64 cell. *Status:* `COMPUTED_HERE`, exact (RUN-031).

4.2 The certificates

Every one of the 9,278 leaves passes on its intended box — centres enclosed over their exact intervals, half-widths at their upper ends — at the density the programme's ledger gives it. Each has the ledger's candidate count, and the margins agree to 1.7×10^{-16}; the least, 3.19489×10^{-8}, is on the same leaf as the ledger's. *Status:* `COMPUTED_HERE`, exact (RUN-031).

4.3 Density is a policy

The ledger gives each leaf a number of directions, from 256 to 16,384, and the programme's text says that "only six leaves require the maximum tested density". The densities fall into disjoint bands of each leaf's reference slack — at least 10^{-3} gets 256, 2×10^{-4} to 10^{-3} gets 512, and so on — so they are starting points chosen from the slack, and every leaf passed at its start. The least density at which each leaf actually passes is another matter:

directions	leaves the ledger gives it	leaves for which it is the least that passes
256	821	7,727
512	3,425	1,285
1,024	3,709	177
2,048	1,115	67
4,096	163	18
8,192	39	4
16,384	6	0

No leaf needs 16,384 directions; 8,457 need fewer than they were given. The six leaves at 16,384 are the six thinnest by reference slack, which is the band rule, and all six already clear T at 8,192. *Status:* COMPUTED_HERE, exact; the programme's sentence PARTIAL — true of the policy, not of the leaves (RUN-031).

4.4 The replay's own kernel lost a corner

The replay found an error in its own earlier code on one leaf. The reference slack the programme gives that leaf is 1.474×10^{-9} above the area the recomputation's exact-hull kernel had computed — and the programme is right: inner polygons of the same bodies at 2^{12} to 2^{21} directions climb past the kernel's value and extrapolate to the programme's. The kernel located the *hand-overs* — the directions in which the body that supports the hull changes — on a grid of $2\pi/1024$, and a corner of the B_7 core stands 1.7×10^{-6} outside the disc over an arc of directions shorter than one grid step, whose two ends the disc owns. A verified envelope checks every segment against every other piece in closed form — the difference of two arc supports is a cosine plus a constant, $|\Delta C|\cos(t - t_0) + \Delta\rho$, which peaks in the direction t_0 of ΔC — and gives that leaf its corner while changing nothing else.

The error only lowered an area, so every lower bound computed with the old kernel stayed a lower bound; an upper bound computed with it would not have been safe, and the next section computes upper bounds. The kernel was corrected, and every earlier run that used it was re-run: every gate log came out byte for byte as committed. *Status:* COMPUTED_HERE; the correction PASS (RUN-031).

5. The upper side: the 77 markers

5.1 Outer polygons

A marker is one base configuration q in a necessity cell. To certify that its hull $H(q)$ has area below T, take normals $n_1, \dots, n_m$ and numbers H_i at least the support of $H(q)$ in direction n_i. The hull lies in $P = \bigcap_i \{x : n_i \cdot x \le H_i\}$, so Area $H(q) \le$ Area P, and Area $P < T$ certifies the marker. Dropping a constraint only enlarges P, so it is safe. *Status:* PASS (RUN-032).

The replay's support bounds are integer intervals at 2^{-400}: $\|n\|/2$ for the disc; for each Reuleaux body, the largest of $n\cdot V_j$ over its corners and of $n\cdot V_k + \|n\|$ over each arc whose range of normals may contain n — an arc is excluded only when it provably cannot be active. The half-planes are intersected through the exact polar hull of the points n_i/H_i, and the area is exact. Each marker point is first checked to lie strictly inside its cell, rebuilt from the exact root along the official axes. *Status:* COMPUTED_HERE, exact (RUN-032).

5.2 Straight edges cost first order

The programme's outer polygons are valid upper bounds — all 77 lie above the exact areas — but they lose between 1.1×10^{-5} and 1.1×10^{-4} of area, and the loss does not behave as a polygon's should.

Proposition 5.1 (the loss at a straight edge). *Let the hull have a straight edge e of length L — a bridging segment between two bodies — whose normal falls strictly between two consecutive grid normals, at angles α and β from it, $\alpha + \beta = d$. Then the outer polygon contains, beyond e, the triangle bounded by e and the lines with those two normals through e's endpoints, of area*

$$\frac{L^2 \sin\alpha \sin\beta}{2\sin(\alpha+\beta)},$$

which is $\frac{1}{4}L^2 \tan(d/2) \approx L^2 d/8$ *when* $\alpha = \beta$*: the loss at such an edge is of first order in the spacing* d *unless the edge's normal lies close to a grid normal.* Each support line in a grid direction is at least as far out as the parallel line through the endpoint on that side, since the endpoints lie in the hull, and no constraint of another grid direction cuts the triangle; so the polygon contains it. Its area is that of a triangle with base L and base angles α and β. Where an endpoint is a body's corner the support lines pass through it and the triangle is the whole excess; where it lies on a smooth arc the excess is larger. Along a strictly convex arc the loss is second order. □ *Status:* `PASS` (RUN-032).

So a uniform grid of normals pays first order at every bridging edge the hull has: doubling the spacing — halving the density — multiplies the loss by 1.3 to 5.5, depending on where each edge's normal falls between the grid directions, and not by the 4 of a smooth body. The remedy is to add each bridging edge's own normal. The replay takes the hand-overs from the verified envelope of §4.4 — 11 for every marker — and adds their normals to a uniform grid; then all 77 markers clear T, 51 of them at 128 uniform normals and 26 at 256. *Status:* `COMPUTED_HERE`, exact (RUN-032).

5.3 What the ranking measured

The programme ranked its markers by its certificates' margins, and its thinnest is marker 6. The three thinnest, three ways:

marker	exact margin	the programme's certificate (normals)	the replay's (normals)
47	1.857×10^{-5}	7.57×10^{-6} (32,768)	1.50×10^{-6} (256)
6	2.112×10^{-5}	6.11×10^{-7} (16,384)	5.57×10^{-6} (256)
70	1.982×10^{-4}	9.25×10^{-5} (4,096)	1.37×10^{-4} (128)

The thinnest marker is 47, not 6: the programme's ranking is its loss's ranking. Its text also calls marker 47's new bound "tighter" than the reference it replaced; both lose first order at the same bridging edges — the reference, a polygon of support lines at fewer directions, 1.83×10^{-5}, the new one, at 32,768, 1.10×10^{-5} — so it is tighter only by having more directions. The replay's certificates stop at the first density that passes, so they are not tight either: at 256 normals marker 47's still gives away 1.7×10^{-5} of its exact margin, now second order in the spacing. *Status:* `COMPUTED_HERE`, exact; the programme's ranking `PARTIAL` — a ranking of its certificates' losses, not of the markers (RUN-032).

As in §4.3, the programme's densities are start points: its emitter chooses the first density from a band of the earlier reference margin, 73 markers pass at their start and four move from 8,192 to 16,384, and "no marker needs 65,536" says that the ladder never went past its highest start. A marker certifies one configuration in its cell, not the cell. *Status:* `COMPUTED_HERE` (RUN-032).

5.4 The first whole shard

The programme's first *whole shard* joins the two sides for one cell: marker 70's outer bound, 0.8349075014501105, and the 9,278 lower leaves of §4 — marker 70 lies in that cell. The shard's records name both halves by digest, and both halves check here; marker 70's exact margin is 1.98×10^{-4}. *Status:* `COMPUTED_HERE` (RUN-032, RUN-033, RUN-034).

6. Two more implementations, and one rule they share

The programme did not rely on its first backend alone. It wrote a second lower-bound implementation, and then an MPFR backend, to replay the same shard. Both were examined here against the replay, which shares nothing with either.

6.1 One contact point per direction

The second implementation builds each leaf's candidates differently: for each direction and body it takes one point — on a valid arc, or at a corner of two defining circles when that corner's support is larger — and then keeps, per direction, only the point of the body whose support wins it. So its candidates are the hull vertices of N points at density N, where the replay's (§2.1) are those of $4N$.

Rebuilt here on the replay's own points and boxes, the winner rule gives back the second implementation's ledger leaf for leaf: its density on all 9,278 leaves, its hull-vertex count on all of them, and every margin to within 1.57×10^{-12} to 1.67×10^{-12} — the two move their points inward by 10^{-12} and 2×10^{-12}. *Status:* `COMPUTED_HERE`, exact (RUN-033).

Proposition 6.1 (the winner rule's loss). *A body that the hull exposes over an arc of normal directions shorter than the grid step may receive no grid direction. When it receives none, the winner rule takes no point of it, and the polygon cuts across it, losing about $\frac{1}{2}L_1L_2\psi$, where L_1, L_2 are the hull edges meeting there and ψ is the exposed window.* The loss is first order in the step, and all or nothing per feature. *Status:* `PASS` (RUN-033).

On the second implementation's thinnest leaf, 11 hull vertices in 10 such runs account for the whole margin it loses, 2.221×10^{-5}: the largest is a stretch of the disc exposed over 1.85×10^{-3} radians (the step at 2,048 directions is $2\pi/2048$) between edges 0.267 and 0.084 long, which cuts 2.105×10^{-5}, where $\frac{1}{2}L_1L_2\psi = 2.077 \times 10^{-5}$. With every contact point, that leaf's margin is 2.22×10^{-5} at the same density, and every leaf's margin rises, by 2.1×10^{-8} to 1.6×10^{-4}. So the second implementation's thinnest leaf is the one whose margin the rule's loss nearly uses up, not the thinnest in the geometry, as the programme itself suspected. This is the inner-polygon side of the loss §5.2 found in the outer polygons, and the same blind spot as the replay's own kernel in §4.4: something exposed between two grid directions. *Status:* `COMPUTED_HERE`, exact (RUN-033).

6.2 The MPFR backend

The MPFR backend's ledger is the second implementation's, leaf for leaf — path, density, hull-vertex count and margin on all 9,278 leaves. It is the same polygons with their membership decided again in MPFR, so it inherits the winner rule, and its thinnest leaf, $3.5423707501 \times 10^{-9}$, is that rule's loss: with every contact point the same leaf has 2.22×10^{-5}. On a second cell the same holds, and the leaf that needed the backend's largest density, 65,536, needs 32,768 under every contact point. *Status:* `COMPUTED_HERE`, exact (RUN-033, RUN-034, RUN-035).

Directed only at its outputs. The backend's text says that exact fractions enter MPFR with directed rounding and leave as exact dyadic fractions. Its wrapper, at 640 bits, loads each exact rational rounded to nearest and directs only the results. So an enclosure can miss by up to about 2^{-640} relative. Every decision it feeds here has vastly more room, so the outputs are covered in practice, but not by construction. The second implementation does the same at 10^{-120}, through decimal strings. *Status:* the rule as written `PARTIAL` — it holds for the outputs, not the inputs — and harmless as used (RUN-033, RUN-034).

Marker 70, twice. The second implementation's replay of marker 70 is the first backend's own audit row, from the same script; that script carries the emitter's support routine, normals and density. So for the upper half of the shard the second implementation is a second polygon path, not a second arithmetic. The MPFR record keeps every constraint — 1,317 active, an exact area whose numerator and denominator have 22,754 and 22,755 bits — where the first backend's file, reduced in floating point, keeps 1,284; the two areas differ by 3.3×10^{-50}. A third variant, counting the arcs beside each active piece as well, gives an area of 0.92016: safe, and unable to certify — the false negative the programme itself recorded and corrected. *Status:* `COMPUTED_HERE`, exact (RUN-033, RUN-034).

6.3 Dominance between backends

Theorem 6.2 (dominance). *Let a backend certify a point inside a defining disc when the supremum of $\|p - V\|^2$ over its enclosure of the corner V is below the square of its lower radius. If backend B's enclosure of each corner lies inside backend A's, and B's lower radius is at least A's, then every point that A certifies inside every defining disc, B certifies too; so a polygon certified vertex by vertex under A is certified under B, and its exact area does not depend on the backend.* A supremum over a subset is no larger, and $\rho_A \le \rho_B$. The theorem is about the exact suprema over the enclosures; for two backends that only bound them from above, it needs in addition that B's bound is no looser than A's. *Status:* `PASS` (RUN-036).

The programme's audits of this condition compare its first backend's enclosures with MPFR's, state by state — about six leaves per state, 1,564 and 1,574 states on two cells, as the replay's own lifts confirm. Their results are not recomputed here: MPFR is not on this machine. The containments that need no MPFR hold. *Status:* the state counts `COMPUTED_HERE`; the audits' results `UNMEASURED` (RUN-036).

7. A certificate format, read from its specification

7.1 RHCERT

The programme's later rounds store a cell's certificate as a binary file, RHCERT, with an index, a manifest and a replay record. The replay decodes it from the specification alone. The header is eight magic bytes, a version, a decimal exponent of 18 — vertices are stored on a 10^{-18} grid — a tag byte and the leaf count; the leaves use up the file exactly. The index holds $L+1$ offsets, each leaf's start and then the end; the specification does not give the index header's layout, and the index file's size does — eight magic bytes, a version and the offset count, then eight bytes per offset. The leaf order is the tree's preorder, and every leaf's density and hull count are the ledger's. *Status:* `COMPUTED_HERE` (RUN-037).

For each leaf the replay takes the box from its own root and the shipped topology, the cores from its own integer intervals — no `mpmath`, no MPFR — and checks every stored vertex inside the core its tag names, the polygon strictly convex, and its exact area above T. Because the area is exact from the bytes, the least margins agree with the programme's replay records bit for bit.

cell	leaves	stored vertices	least margin	run
47	9,228	2,163,069	$6.75024723062513 \times 10^{-9}$	RUN-037
48	9,285	2,109,700	$1.0316417842561089 \times 10^{-7}$	RUN-037
68	9,255	2,126,336	$2.232921067369829 \times 10^{-8}$	RUN-038
75	10,948	2,627,357	$6.251776141237558 \times 10^{-8}$	RUN-039
76	10,932	2,617,252	$2.0477267269918978 \times 10^{-8}$	RUN-039

cell	leaves	stored vertices	least margin	run
49	9,986	2,441,597	$3.153126719767511 \times 10^{-8}$	RUN-040
64	10,539	2,624,186	$1.506934987941895 \times 10^{-8}$	RUN-040
66	10,636	2,646,674	$6.860894579634212 \times 10^{-9}$	RUN-040

None of the stored vertices is refused its core, every polygon is strictly convex, and every area is above 167/200. *Status:* COMPUTED_HERE, exact (RUN-037 to RUN-040).

The certificates are the replay's own as well as the programme's. On cell 68, every leaf's density and hull count are those of the certificate the replay builds itself under the programme's start bands, and the stored bytes are its polygons up to the 10^{-18} grid; the margins agree to 2.6×10^{-16}. *Status:* COMPUTED_HERE (RUN-038).

7.2 What the format costs, and a caution

The raw certificates take about 17 bytes a vertex, and their compressed packages under 4: the two first packages are 8.02 and 7.54 MiB, not the "about 8.1" and "about 7.6" the text says; their raw sizes, 34.8 and 34.0 MiB, are as stated. The programme kept two cells below publication status because their ledgers keep no hull vertices, so that its cross-backend audit had nothing to transfer. The premise is true and the status was right; but the inference that the certificates cannot be checked without their bytes fails: the vertices regenerate from each leaf's density and the candidate rule — equal to the emitter's up to its 17-digit rounding, not bit for bit — and the replay regenerated them. *Status:* COMPUTED_HERE (RUN-036, RUN-037).

One caution belongs to the replay. Its earlier runs (RUN-035 to RUN-038) built a leaf's box by pairing the leaf's path with a split sequence 30 long; at depth 32 that pairing would silently drop the last two splits and certify each deep leaf on its depth-30 ancestor's larger box — a check that passes for the wrong reason. It was caught before anything ran: the sequence was carried to 32 and the lengths asserted. *Status:* COMPUTED_HERE (RUN-040).

8. Trees and frontiers

8.1 Trees, node for node

Every tree the certificates index was grown again here from the cell's path and compared node for node and axis for axis with the shipped topology and axis stream — cells 47, 48, 68, 69, 75 and 76 at depth 30, and cells 49, 64 and 66 at depth 32 — each matching its manifest's digests. A cell whose tree is complete and every leaf certified is closed for the family $\{D, B_3, B_5, B_7\}$. *Status:* COMPUTED_HERE (RUN-035, RUN-036, RUN-039, RUN-040).

8.2 DFRONT

Not every cell closes. For those that do not, the programme stores the *frontier* — the open boxes at a depth — in a format, DFRONT, of canonical codes. The replay decodes it from the specification: codes in canonical order, every coordinate below 2^k at depth k, the boxes interior-disjoint under a sweep written here, and exact volumes — 1/2 for the test frontier and $14027/2^{36}$ for cell 33's. Grown here to depth 36 under the programme's pad (hull area less 10^{-10}), cell 33's tree leaves the same 14,027

boxes open, code for code; and the frontier file, written here from the specification and those boxes, is the shipped file byte for byte. *Status:* COMPUTED_HERE, exact for the codes and volumes, numerical for the closures under the pad (RUN-039).

Carried on one depth at a time from that frontier, the cell keeps 27,042, 40,934 and 78,507 boxes open at depths 38, 40 and 42 — the programme's figures code for code — and each checkpoint file, and every one of 166 segment files, written here from the specification with its parent's digest, is the shipped file byte for byte. 78,507 boxes are still open at 42: the cell is not closed. *Status:* COMPUTED_HERE (RUN-040).

8.3 A merge that never checks its premise

The programme merges a frontier from segments, one per slice of the parent frontier, and states a theorem: slices of a disjoint frontier, each refined only inside itself, give disjoint descendants. The theorem is right, and its premise — every segment box inside its own slice — holds for all 146,483 shipped boxes, checked here directly. But the merge's specification does not list the premise among its seven requirements, and the shipped merge does not test it: it reads neither the slice hashes nor the parent boxes, only their number. Two segments of one box each, the second lying in the first segment's slice, meet all seven requirements and overlap. So the disjointness of these files comes from the refinement code's construction, not from the merge; a merge meant to stand on its own needs one ancestor lookup per box. *Status:* the theorem PASS, its premise holding for every shipped box; that the merge's specification omits it is a finding, not a grade (RUN-040).

9. What is certified, and what is not

Certified here, in the replay's own arithmetic. For the family $\{D, B_3, B_5, B_7\}$ at the threshold $T = 167/200$, ten of the 77 necessity cells are closed by complete trees whose every leaf carries an inner polygon of exact area above T:

- the cell of marker 70 (cell 0 of the reference rounds), 9,278 leaves (§4; RUN-031);
- cells 47 and 69, 9,228 and 7,856 leaves, from the programme's ledgers (RUN-035), and cell 47 again with cells 48, 68, 75, 76, 49, 64 and 66 from their RHCERT bytes (§7; RUN-037 to RUN-040).

Each of the 77 cells contains a marker configuration whose hull has area below T, by an outer polygon certified here (§5); so each of them needs a witness. *Status:* COMPUTED_HERE, exact (RUN-031 to RUN-040).

Floating point only. Whether the other cells close: in floating-point exact-core geometry under the programme's pad, the number of necessity cells whose trees close by a common lift depth is 56, 57, 63 and 63 at depths 37 to 40, and continuing the open ones, 66 of the 77 are complete; the rest are not. *Status:* COMPUTED_HERE, numerical (RUN-030, RUN-035, RUN-039, RUN-040).

Not here. Every result of the MPFR backend that the replay could not reproduce from its records — its dominance audits and its marker replays, whose records are not all shipped. *Status:* UNMEASURED.

Not certified by anyone. $a_{\mathrm{Leb}} \geq 0.835$. Sixty-seven necessity cells are not certified here, eleven are not even geometrically complete, and the base cells outside the 77 are not certified here either.

In all, the replay checked 97,943 leaves in ten cells, 80,809 of them from RHCERT files holding 19,356,171 stored vertices, and every one passed. *Status:* COMPUTED_HERE, exact (RUN-031,

RUN-035, RUN-037 to RUN-040).

10. Every result, and the strength at which it holds

result	holds as	checked in
§1.3 — the a-priori root's formulas	PASS	RUN-031
Proposition 2.1 — inner polygons	PASS	RUN-030
§2.2–2.3 — the arithmetic; its refusals; twelve leaves	COMPUTED_HERE, exact	RUN-030, RUN-031, RUN-033
§3 — the root bracketed; the stored roots against it; four brackets nested	COMPUTED_HERE, exact	RUN-030, RUN-031, RUN-033, RUN-034
§4.1–4.2 — one cell's tree and all 9,278 leaves	COMPUTED_HERE, exact	RUN-031
§4.3 — "only six leaves require" the top density	PARTIAL: true of the policy, not of the leaves	RUN-031
§4.4 — the replay's kernel, corrected	COMPUTED_HERE; the correction PASS	RUN-031
§5.1 — outer polygons; the 77 markers	PASS; the certificates COMPUTED_HERE, exact	RUN-032
Proposition 5.1 — the loss at a straight edge	PASS	RUN-032
§5.3 — the programme's ranking of its markers	PARTIAL: a ranking of losses, not of markers	RUN-032
§5.4 — the first whole shard	COMPUTED_HERE, exact	RUN-032, RUN-033, RUN-034
§6.1, Proposition 6.1 — the winner rule and its loss	COMPUTED_HERE, exact; the loss PASS	RUN-033
§6.2 — the MPFR ledger; directed rounding at outputs only	COMPUTED_HERE; the rule as written PARTIAL	RUN-033, RUN-034, RUN-035
Theorem 6.2 — dominance between backends	PASS; the audits' results UNMEASURED	RUN-036
§7.1 — RHCERT decoded; eight cells, every vertex	COMPUTED_HERE, exact	RUN-037 to RUN-040
§7.2 — sizes; vertices regenerated; the depth-32 caution	COMPUTED_HERE	RUN-036, RUN-037, RUN-040
§8.1 — trees node for node	COMPUTED_HERE	RUN-035, RUN-036, RUN-039, RUN-040
§8.2 — DFRONT decoded and written byte for byte	COMPUTED_HERE; the closures numerical	RUN-039, RUN-040

result	holds as	checked in
§8.3 — the compositional merge	the theorem `PASS`; the omitted premise a finding	RUN-040
§9 — ten cells certified; 77 markers	`COMPUTED_HERE`, exact	RUN-031 to RUN-040
§9 — the cells closed in floating point	`COMPUTED_HERE`, numerical	RUN-030, RUN-035, RUN-039, RUN-040

No result is `INHERITED`. The figures this paper takes from the programme's records without recomputing them are four recorded values it compares with its own — two certificate margins in §5.3, two package sizes in §7.2 — and the programme's count of the cells it calls certified, in §1.3.

11. Corrections found on the way

11.1 Statements that do not hold as written

- **Density as need.** "Only six leaves require the maximum tested density" and "no marker needs 65,536" describe the programme's start bands, not its leaves or markers (§4.3, §5.3). (RUN-031, RUN-032)
- **The thinnest marker.** It is marker 47, not marker 6; the programme's ranking is its outer polygons' first-order loss at bridging edges, and marker 47's "tighter" bound is tighter only by having more directions (§5.3). (RUN-032)
- **Containment, not merely overlap.** The programme says its root brackets overlap; they are nested (§3.3). Its correction names 10^{-64} as the width of a $d_\star$ interval, which is the width of the t_n intervals; and its text gives $d_\star$'s width as $O(10^{-63})$ where its file says 3.1×10^{-61}. (RUN-031, RUN-033, RUN-034)
- **Independence.** The second implementation's replay of the shard's upper half is the first backend's audit row, from a script that carries the emitter's support code — a second polygon path, not a second arithmetic (§6.2); and its lower replay reads the first backend's root intervals, not its own bracket (§3.3). (RUN-033)
- **The thinnest lower leaf** of the second implementation and of the MPFR backend measures their shared winner rule's loss, not the geometry (§6.1–6.2). (RUN-033, RUN-034)
- **Directed rounding** holds for the MPFR backend's outputs only; its inputs are rounded to nearest (§6.2). (RUN-034)
- **The compositional merge's specification** omits the premise its theorem rests on, and the shipped merge does not test it (§8.3). (RUN-040)

11.2 Records without consequence

- The two first RHCERT packages are 8.02 and 7.54 MiB, not "about 8.1" and "about 7.6" (§7.2). (RUN-037)
- The programme kept cells 47 and 48 below publication status because their ledgers keep no vertices. The premise is true and the status was right, but the certificates do not need the missing bytes: the vertices regenerate from density and rule, up to the emitter's rounding (§7.2). (RUN-036)

11.3 Valid, but weaker than needed

- The programme reports the number of necessity cells closed by lift depths 38 and 40 only as lower bounds, 57 and 63; at a common depth the counts are exact, 56, 57, 63 and 63 at depths 37 to 40 (§9). (RUN-030)

11.4 The recomputation's corrections of itself

- The recomputation's exact-hull kernel lost a corner exposed within one grid step, 1.474×10^{-9} low on one leaf; it was corrected, and every run that used it re-run with byte-identical logs (§4.4). (RUN-031)
- A pairing of leaf paths with a 30-long split sequence would have dropped the last two splits at depth 32; it was caught before anything ran (§7.2). (RUN-040)
- The line's first paper [2] says that cell 69's ledger was reproduced in floating point and not re-certified. Its 7,856 leaves were checked with the replay's exact certificate on the intended boxes (§9); the first paper understates it, and its next version should say so. (RUN-035)

12. Reproduction and attachments

What a reader can check without the archive. The recomputation tree is private; its public projection is not. Every report, every script and every gate log of its 68 runs, as they stood at the pinned commit, is published with SHA-256 sums at https://amral.evemisslab.com/lebesgue/recompute/ (English: https://amral.evemisslab.com/en/lebesgue/recompute/), one page per document of the programme. The eleven runs this paper replays are RUN-030 to RUN-040; it also cites RUN-002, for the arXiv records of the published bounds. Every load-bearing number in this paper occurs in their gate logs or in `data/paper-figures.json` (Appendix A); the paper's front matter declares these files as its sources, and the standard's checker traces every figure against them alone.

The instrument can fail, and was made to. Every check of every run was drilled: faults were planted in its inputs, and each had to be caught by exactly the check named for it, while controls — the unmodified inputs and harmless rewrites — had to stay green. Over the eleven runs of this paper: 184 planted faults, all 184 caught by the named check, 44 controls undisturbed, 113 checks, 2,123 must-stay-green assertions held. Over the whole recomputation tree, of which these runs are part: 1,290 planted faults, 1,290 caught by the named check, 262 controls undisturbed, 665 checks.

What re-running needs. The programme's documents, packages and certificate files are two archives, kept byte for byte; the input manifest `data/SOURCE_MANIFEST.v1.json` gives the size, SHA-256 and git blob id of each and the SHA-256 of every leaf. The archives are not public; the scripts read them in memory and never extract them.

Commands. Python 3.14 [7] with mpmath 1.3 [6], numpy 2.4 [8], scipy 1.17 [9] (its Qhull interface [11]) and sympy 1.14 [10]; no network. From the line's directory (`lebesgue-verification-xuanheng/` at the pinned commit), each run is re-run, and its drill, by

```
python code/run037_round34_rhcert.py
python code/run037_round34_rhcert.py --drill
```

and likewise for every `code/runNNN_*.py` named above. A run rewrites its gate log under `data/gate-logs/`; a re-run is correct when that log is byte-identical to the published one. `python code/paper_figures.py` rewrites `data/paper-figures.json` and is correct on the same terms.

The paper is typeset by the shared build tool of the crystallization repository, which refuses to write a PDF from which a character would be dropped.

13. Contributions

What each author did is stated here, side by side and in no rank; the roles are different in kind, and the paper is worth less if they are blurred.

Aletheia — the web-side research line; model: ChatGPT, GPT series — produced rounds 27 to 37 of the finite-closure line, with their certificates, trees, ledgers and the RHCERT and DFRONT formats, on the programme Neo.K set out. Every certificate this paper replays is from those rounds.

Neo.K set out the research programme: the finite-closure strategy and its target, and the methodologies it runs on.

懸衡 (Xuán Héng) — the independent recomputation; model: Claude, Opus series — did the runs this paper names and wrote this paper: the integer-interval arithmetic, the root and the trees, built independently of the programme's code; every certificate this paper calls certified, checked in that arithmetic; the RHCERT and DFRONT readings from their specifications; the drills of §12; Proposition 5.1 and Proposition 6.1, and the measurements behind them; and the corrections of §11.

Review. Before it was pushed, a separate instance of the same model (model: Claude Opus, version 5.5), run as a read-only subagent of 懸衡's Claude Code session and given only the paper, its run reports, the gate logs and the companion papers, read it; its findings — a stale count, missing hypotheses, a verdict written at the wrong strength among them — were applied. 墜衡 (Zhuì Héng) — the lab's proof-side verification arm; model: Claude, Opus series — then reviewed this paper before release: he rederived its arguments and recomputed its figures from the text, independently of this line's code, and found no mathematical error and nothing to change.

Priority, stated plainly. The certificates, the formats and the pipeline are the programme's. What is offered as this line's own is the replay, the losses it measured, and the corrections. No bound on Lebesgue's constant is claimed.

Acknowledgements

We thank Anthropic, whose Claude models the lab's verification and recomputation lines run on, and OpenAI, whose ChatGPT models the programme's documents were written with. Not every AI model or instance that took part in this work is named in it — among them instances that the named authors ran as helpers; that an AI is not listed here does not mean that none took part.

References

1. Aletheia, Neo.K and 懸衡, *A certified finite reduction for Lebesgue's universal covering problem*, preprint, AMRAL Research Lab (2026), the second paper of this line.
2. Aletheia, Neo.K and 懸衡, *A finite-closure programme for Lebesgue's universal covering problem: certified progress, target ledgers, and an independent recomputation*, preprint, AMRAL Research Lab, the first paper of this line, https://amral.evemisslab.com/en/lebesgue/paper/ (2026).
3. P. Gibbs, *An upper bound for Lebesgue's covering problem*, https://arxiv.org/abs/1810.10089 (2018).
4. U. Mishra, *Curves of constant width and Lebesgue's covering problem*, https://arxiv.org/abs/2608.30538 (v2, 2026).

5. L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier and P. Zimmermann, *MPFR: a multiple-precision binary floating-point library with correct rounding*, ACM Trans. Math. Software 33(2) (2007).
6. The mpmath development team, *mpmath: a Python library for arbitrary-precision floating-point arithmetic* (version 1.3.0), https://mpmath.org (2023).
7. The Python Software Foundation, *Python* (version 3.14), https://www.python.org.
8. C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers et al., *Array programming with NumPy*, Nature 585 (2020) 357–362, doi:10.1038/s41586-020-2649-2; used here at version 2.4.
9. P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland et al., *SciPy 1.0: fundamental algorithms for scientific computing in Python*, Nature Methods 17 (2020) 261–272, doi:10.1038/s41592-019-0686-2; used here at version 1.17.
10. A. Meurer, C. P. Smith, M. Paprocki, O. Čertík et al., *SymPy: symbolic computing in Python*, PeerJ Comput. Sci. 3 (2017) e103, doi:10.7717/peerj-cs.103; used here at version 1.14.
11. C. B. Barber, D. P. Dobkin and H. Huhdanpaa, *The quickhull algorithm for convex hulls*, ACM Trans. Math. Software 22(4) (1996) 469–483, doi:10.1145/235815.235821; Qhull, behind SciPy's convex hulls.

Appendix A. The figures file

`data/paper-figures.json`, and the script that writes it, `code/paper_figures.py`, are published beside this paper as they stood at the pinned commit. The script runs nothing of the programme and computes no mathematics of its own. Its entries are of five kinds, never mixed: sums, differences and ratios of gate-log fields, each naming the fields it reads; counts over the recomputation's inventory and drill records, for the whole tree and for the runs of each paper; grid sizes and sample counts read from the run scripts as written; values that a run's own function computes and checks but its gate log does not record, evaluated at the run's own inputs; and figures quoted from the programme's records, each with its document, the document's SHA-256 and the line. Re-run in the recorded environment, the file is byte-identical.