A Width–2 Boundary Program for Excluding Off–Axis Quartets with a Low–Anchor Tail Certificate and a Finite–Height Front–End (v31)

Dylan Anthony Dupont

v31 compiled: 2026-01-17

Abstract

This manuscript develops a width–2 "boundary program" designed to exclude off–axis quartets of nontrivial zeros of the Riemann zeta function. The analytic core reduces the large–height "tail" to a single explicit inequality (Equation (21)) at one anchor height, together with worst–α and monotonicity lemmas that propagate the check to all larger heights.

v31 re-engineering (what changed from v30). In v30 the tail inequality was anchored at a large height $m_{\text{band}} = 6 \cdot 10^{12}$ to match a published "verified band" input. In v31 we instead anchor the tail inequality at the minimal admissible height $m_\star = 10$ (as required by the residual envelope statement), and treat the remaining region $0 < m < 10$ as a small finite–height front–end in the classical s-plane ($0 < \mathrm{Im}(s) < 5$).

Claim hygiene (v31 posture). The interval certificates included here prove the tail inequality at $m_\star = 10$ *given* the constants ledger intervals $(C_1, C_2, C_{\text{up}}, C_h'')$. However, these certificates do *not* by themselves certify that the ledger constants are valid bounds for the analytic and geometric quantities asserted in the lemmas. Accordingly, v31 remains a *certified criterion*: *RH follows provided the constants ledger is independently certified* by explicit quantitative analysis or by separate certified computation.

Contents

Executive Proof Status

1. **Analytic spine (paper).** Part II develops the local boundary mechanism: if an off–axis quartet exists at width–2 height m, then on an aligned box $B(\alpha, m, \delta)$ one must simultaneously have (i) a forcing phase lower bound of size $\pi/2$ on a short side from the local pair factor, and (ii) an upper bound on the remaining boundary variation in terms of a residual envelope $\sup_{\partial B} |F'/F|$ plus shape–only constants. This yields a concrete tail inequality (Theorem 11.1).

2. **Tail certificates actually executed (this version).** Appendix A contains a deterministic interval-arithmetic generator/verifier. In this build, the verifier was executed for:

   - the historical v29 anchor $m_{\text{band}} = 6 \cdot 10^{12}$ (Appendix A.4), and
   - the v31 low anchor $m_\star = 10$ (Appendix A.5).

   In both cases the verifier prints the strict check `CHECK: lhs.hi < rhs.lo : True` and returns `PASS`.

3. **What the tail certificate does and does not prove.** It proves: *given the ledger intervals*, Equation (21) holds with strict interval separation at the anchor height, hence for all larger heights by monotonicity. It does *not* prove that the ledger intervals are correct.

4. **Finite-height front-end after lowering the anchor.** With $m_\star = 10$, the only remaining heights not covered by the tail are $0 < m < 10$, i.e. $0 < \mathrm{Im}(s) < 5$. A clean global closure requires only a small front-end statement (Section 4). In v31 this is discharged by citation to Platt–Trudgian's rigorous verification of RH up to height $3 \cdot 10^{12}$, which trivially implies RH up to height 5.

5. **Hard gate remaining for an unconditional claim.** The constants ledger $(C_1, C_2, C_{\mathrm{up}}, C_h'')$ is currently provided only as a JSON interval file (Appendix A.6). Unconditional closure requires an independent certification of each ledger constant (Section 13).

One-shot audit path. From the accompanying folder `v31_repro_pack/`, run:

1. `sha256sum -c SHA256SUMS.txt`
2. `python3 v31_verify_tail_certificate.py --constants v29_constants.json --certificate v29_tail_certificate.json`
3. `python3 v31_verify_tail_certificate.py --constants v31_constants_m10.json --certificate v31_tail_certificate_m10.json`

Part I
Reader's Guide / Definitions and Reduction

1 Width–2 normalization

Define the width–2 objects

$$u := 2s, \qquad \zeta_2(u) := \zeta\big(\tfrac{u}{2}\big), \qquad \Lambda_2(u) := \pi^{-u/4}\,\Gamma\big(\tfrac{u}{4}\big)\;\zeta\big(\tfrac{u}{2}\big). \tag{1}$$

Then Λ_2 is entire and satisfies the functional equation

$$\Lambda_2(u) = \Lambda_2(2-u). \tag{2}$$

We recenter at $u = 1$:

$$v := u - 1, \qquad E(v) := \Lambda_2(1+v). \tag{3}$$

The functional equation becomes the evenness relation

$$E(v) = E(-v), \tag{4}$$

and complex conjugation gives $E(\overline{v}) = \overline{E(v)}$.

2 Heights and horizontal displacement (RH–free)

Let $\rho = \beta + i\gamma$ be any nontrivial zero of $\zeta(s)$ (no assumption on β). In width–2 we write

$$u_\rho := 2\rho = (1+a) + im, \qquad a := 2\beta - 1 \in (-1,1), \qquad m := 2\gamma > 0. \tag{5}$$

Thus RH is equivalent to $a = 0$ for every nontrivial zero.

3 Quartet symmetry in width–2

The functional equation and conjugation imply that any off–axis zero with parameters (a, m) produces a quartet

$$\{\, 1 \pm a \pm im \,\} \subset \{u \in \mathbb{C} : \Lambda_2(u) = 0\}. \tag{6}$$

In the centered v–coordinate this becomes $\{\pm a \pm im\} \subset \{v \in \mathbb{C} : E(v) = 0\}$.

4 Finite-height front-end after lowering the tail anchor

Once the tail anchor is lowered to $m_\star$, the analytic tail argument covers all $m \geq m_\star$. The remaining region corresponds to classical heights

$$0 < \mathrm{Im}(s) < H_0 := m_\star/2. \tag{7}$$

In v31 we take $m_\star = 10$, hence $H_0 = 5$.

Definition 4.1 (Front-end statement)**.** We say that *RH holds up to height* H_0 if every nontrivial zero $\rho = \beta + i\gamma$ with $0 < \gamma \leq H_0$ satisfies $\beta = 1/2$.

Remark 4.2 (How v31 discharges the front-end)*.* The required statement for v31 is RH up to height $H_0 = 5$. This is a tiny special case of published rigorous verifications of RH to enormous heights. For example, Platt–Trudgian prove RH for all zeros with $0 < \gamma \leq 3 \cdot 10^{12}$ using interval arithmetic, which immediately implies RH up to $H_0 = 5$. Appendix B records this discharge in a pinned JSON file.

Part II
Self-Contained Boundary Program and Tail Closure

5 Aligned boxes and the $\delta(m)$ scale

Let $m > 0$ and $\alpha \in (0, 1]$. Fix a parameter $\eta \in (0, 1)$ and set

$$\delta = \delta(m, \alpha) := \frac{\eta\alpha}{(\log m)^2}. \tag{8}$$

Define the (width–2) box centered at $\alpha + im$ by

$$B(\alpha, m, \delta) := \{\, v \in \mathbb{C} : |\mathrm{Re}\, v - \alpha| \leq \delta,\ |\mathrm{Im}\, v - m| \leq \delta \,\}. \tag{9}$$

We will also use the symmetric dial centers $v_\pm := \pm\alpha + im$.

6 Local factor and finiteness

For a fixed $m > 0$, let

$$Z(m) := \{\, \rho : E(\rho) = 0,\ |\mathrm{Im}\, \rho - m| \leq 1 \,\} \tag{10}$$

(zeros counted with multiplicity). Define the local zero factor and residual:

$$Z_{\mathrm{loc}}(v) := \prod_{\rho \in Z(m)} (v-\rho)^{m_\rho}, \qquad F(v) := \frac{E(v)}{Z_{\mathrm{loc}}(v)}. \tag{11}$$

Lemma 6.1 (Finiteness of Z_{loc})**.** *For each fixed $m > 0$ the set $Z(m)$ is finite; hence Z_{loc} is a finite product and F is meromorphic globally and analytic in any neighborhood of $\partial B(\alpha, m, \delta)$ that contains no zeros of E.*

Proof. Nontrivial zeros of ζ satisfy $0 < \beta < 1$, hence in the v–coordinate one has $\mathrm{Re}\, v \in (-1, 1)$ for all nontrivial zeros. Therefore the set $\{|\mathrm{Im}\, v - m| \le 1\} \cap \{|\mathrm{Re}\, v| \le 1\}$ is compact. Since E is entire and its zeros are discrete, only finitely many zeros can lie in this compact set. □

7 Residual envelope bound and the constants ledger

Lemma 7.1 (Residual envelope inequality)**.** *There exist absolute constants $C_1, C_2 > 0$ such that for all $m \ge 10$, all $\alpha \in (0, 1]$, and $\delta = \eta\alpha/(\log m)^2$, one has*

$$\sup_{v \in \partial B(\alpha, m, \delta)} \left| \frac{F'(v)}{F(v)} \right| \le C_1 \log m + C_2. \tag{12}$$

Remark 7.2 (Hard gate). The tail certificates in Appendix A use explicit numerical interval enclosures for C_1 and C_2 (stored in `v31_repro_pack/v29_constants.json`). The certificates verify the tail inequality *conditional on* these enclosures being correct. An unconditional claim requires an independent certification of C_1, C_2 and the shape-only ledger constants (Section 13).

8 Short-side forcing

Assume an off-axis pair at height m with displacement $a > 0$ exists. On an aligned box with $\alpha = a$, the two upper zeros in the centered v–plane are at $v = \pm a + im$. The pair factor

$$Z_{\mathrm{pair}}(v) := (v - (a + im))(v - (-a + im)) \tag{13}$$

produces a large phase rotation on the near vertical side.

Lemma 8.1 (Short-side forcing lower bound)**.** *Let $I_+ := \{\alpha + iy : |y - m| \le \delta\}$ with $|\alpha - a| \le \delta$. Then*

$$\Delta_{I_+} \arg Z_{\mathrm{pair}} = 2 \arctan\left(\frac{\delta}{|\alpha - a|}\right) + 2 \arctan\left(\frac{\delta}{\alpha + a}\right) \ge \frac{\pi}{2}. \tag{14}$$

9 Outer factorization and the inner quotient (Bridge 1)

Let $B = B(\alpha, m, \delta)$ and assume E has no zeros on ∂B. Let U be the harmonic solution to the Dirichlet problem on B with boundary data $\log |E|$. Let V be a harmonic conjugate on B (chosen so that $U + iV$ is analytic). Define the outer function

$$G_{\mathrm{out}}(v) := \exp(U(v) + iV(v)). \tag{15}$$

Then G_{out} is analytic and zero-free on B and satisfies $|G_{\mathrm{out}}| = |E|$ on ∂B. Define the inner quotient

$$W(v) := \frac{E(v)}{G_{\mathrm{out}}(v)}. \tag{16}$$

Then W is analytic on B and satisfies $|W| = 1$ on ∂B.

Proposition 9.1 (Bridge 1: boundary modulus 1 forces constancy if zero-free)**.** *Assume W is analytic and zero-free on B, continuous on $\overline{B}$, and satisfies $|W| = 1$ on ∂B. Then W is constant on B.*

Proof. Since W is continuous on $\overline{B}$ and analytic on B, the maximum modulus principle gives $|W| \le 1$ on B. Since W is zero-free, $1/W$ is analytic on B and continuous on $\overline{B}$, with $|1/W| = 1$ on ∂B. Applying the maximum modulus principle to $1/W$ yields $|1/W| \le 1$ on B, i.e. $|W| \ge 1$ on B. Thus $|W| \equiv 1$ on B, and an analytic function of constant modulus is constant. □

10 Shape-only invariance and the envelope constants

Let $T(v) := (v - (\alpha + im))/\delta$, mapping ∂B affinely onto the fixed square boundary ∂Q with $Q = [-1, 1]^2$.

Lemma 10.1 (Shape-only invariance)**.** *Any constant arising solely from geometric or boundary-operator estimates on ∂B that are invariant under affine rescaling depends only on ∂Q and is independent of (α, m, δ).*

Proof. Under T, arclength scales by δ and tangential derivatives by $1/\delta$. After normalization, all purely geometric quantities and operator norms reduce to fixed quantities on ∂Q. □

Lemma 10.2 (Upper envelope bound (residual form))**.** *There exists a shape-only constant $C_{\mathrm{up}} > 0$ such that on aligned boxes $\alpha = \pm a$ one has*

$$\sum_{\pm} \left| W(v_{\pm}) - e^{i\varphi_0^{\pm}} \right| \le 2C_{\mathrm{up}}\, \delta^{3/2} \sup_{v \in \partial B} \left| \frac{F'(v)}{F(v)} \right|, \tag{17}$$

where $e^{i\varphi_0^{\pm}}$ are fixed boundary phase anchors for the two dial centers.

Lemma 10.3 (Horizontal budget)**.** *There exists a shape-only constant $C_h'' > 0$ such that, after removing the residual factor F, the remaining non-forcing boundary phase contribution satisfies*

$$\Delta_{\mathrm{nonforce}} \le C_h''\, \delta\, (\log m + 1) \tag{18}$$

on aligned boxes.

11 The explicit tail inequality

Define

$$L(m) := C_1 \log m + C_2. \tag{19}$$

Fix the numerical constants

$$c_0 := \frac{3 \log 2}{8\pi}, \qquad c := \frac{3 \log 2}{16}, \qquad K_{\mathrm{alloc}} := 3 + 8\sqrt{3}. \tag{20}$$

Theorem 11.1 (Tail inequality (certified form))**.** *Fix $\eta \in (0, 1)$ and set $\delta = \eta\alpha/(\log m)^2$. Let $C_{\mathrm{up}}, C_h'' > 0$ be the shape-only constants from Lemma 10.2 and Lemma 10.3, and let $C_1, C_2 > 0$ be residual constants from Lemma 7.1. If the inequality*

$$2C_{\mathrm{up}}\, \delta^{3/2}\, L(m) < c - \delta \left(K_{\mathrm{alloc}}\, c_0\, L(m) + C_h''\, (\log m + 1) \right) \tag{21}$$

holds for a given $m \ge 10$ and all $\alpha \in (0, 1]$, then there is no off–axis quartet at width–2 height m.

Lemma 11.2 (Worst–α reduction)**.** *For fixed m and fixed admissible constants, the left-hand side of* (21) *is increasing in $\alpha \in (0,1]$ and the right-hand side is decreasing in $\alpha \in (0,1]$. Therefore it suffices to verify* (21) *at $\alpha = 1$.*

Proof. With $\delta(\alpha) = \eta\alpha/(\log m)^2$, the left-hand side is proportional to $\delta(\alpha)^{3/2} \propto \alpha^{3/2}$. The right-hand side equals $c - \delta(\alpha) \cdot (\cdots)$, hence is affine decreasing in α. □

Lemma 11.3 (Monotonicity for one-height verification)**.** *Fix $\eta \in (0,1)$ and any admissible constants. For all $m > 1$ and fixed $\alpha \in (0,1]$, the left-hand side of* (21) *is non-increasing in m, and the right-hand side is non-decreasing in m. Consequently, verifying* (21) *at one height $m = m_\star$ implies it for all $m \geq m_\star$.*

Proof. Write $x = \log m > 0$. Since $\delta(m) = \eta\alpha/x^2$, we have $\delta(m)^{3/2}L(m) = \eta^{3/2}\alpha^{3/2}(C_1x^{-2} + C_2x^{-3})$, which is strictly decreasing in x because its derivative is $\eta^{3/2}\alpha^{3/2}(-2C_1x^{-3} - 3C_2x^{-4}) < 0$. Thus the left-hand side decreases in m.

For the right-hand side, the subtractive term equals $\eta\alpha(Ax^{-1} + Bx^{-2})$ with $A := K_{\text{alloc}}c_0C_1 + C_h'' > 0$ and $B := K_{\text{alloc}}c_0C_2 + C_h'' > 0$. Its derivative in x is negative: $-\eta\alpha(Ax^{-2} + 2Bx^{-3}) < 0$, hence the subtractive term decreases in m and the right-hand side increases. □

Theorem 11.4 (Tail closure from one certified check)**.** *Fix $\eta \in (0,1)$ and suppose* (21) *holds at one height $m = m_\star$ for $\alpha = 1$. Then* (21) *holds for all $m \geq m_\star$ and all $\alpha \in (0,1]$. In particular, there are no off–axis quartets at any width–2 height $m \geq m_\star$.*

Proof. Combine Lemma 11.2 and Lemma 11.3. □

12 Global RH from a small front-end + a low-anchor tail (conditional on the ledger)

Theorem 12.1 (Global RH from front-end + tail + certified ledger)**.** *Let $m_\star = 10$. Assume:*

(A) ***Front-end.*** *RH holds up to classical height $H_0 = m_\star/2 = 5$ (Definition 4.1).*

(B) ***Ledger constants.*** *The constants ledger $(C_1, C_2, C_{\text{up}}, C_h'')$ is independently certified to satisfy the interval enclosures in `v31_repro_pack/v29_constants.json`.*

(C) ***One-height tail check.*** *The tail inequality* (21) *is verified at $m = m_\star$ and $\alpha = 1$.*

Then RH holds for all nontrivial zeros of $\zeta(s)$.

Proof. By (C) and Theorem 11.4, there are no off-axis quartets for all width–2 heights $m \geq m_\star$, i.e. no off-axis zeros for $\text{Im}(s) \geq m_\star/2 = 5$. By (A), there are no off-axis zeros for $0 < \text{Im}(s) \leq 5$. Therefore there are no off-axis zeros at any height. □

Remark 12.2 (What blocks "unconditional proof" in v31). Appendix A demonstrates (C) as a reproducible interval inequality check. Appendix B records a standard literature discharge of (A). However, (B) is not discharged inside this version set: no independent derivation or certified computation is provided for the ledger constants. The most conservative correct posture is therefore: Theorem 12.1 is a certified criterion.

13 Ledger certification requirements (hard gate)

To promote Theorem 12.1 to an unconditional (computer-assisted) proof, one must certify *each* ledger constant. At minimum, for each constant one needs:

1. an unambiguous mathematical specification (what quantity is being bounded, on what domain),
2. a certification method (explicit analytic bound with quantified constants, or a certified computation using ball/interval arithmetic), and
3. a pinned, reproducible artifact bundle (code, inputs, outputs, hashes).

Appendix C provides an explicit certification plan and a checklist of artifacts.

A Tail certificate bundle and reproducibility

A.1 What the tail certificates prove (and what they do not)

Each tail certificate proves the statement:

> Given a constants file that provides interval enclosures for $(C_1, C_2, C_{\mathrm{up}}, C_h'')$ and the chosen parameters (m, η, α), the generated interval bounds satisfy LHS $<$ RHS with strict separation in the sense $\mathrm{LHS}_{\mathrm{hi}} < \mathrm{RHS}_{\mathrm{lo}}$.

It does *not* certify that the constants file is correct.

A.2 SHA–256 table (exact artifacts)

The file `v31_repro_pack/SHA256SUMS.txt` is the canonical hash list.

```
5c48b19813580ef3dd8e623d85bff2ec256740513c3e66f248cf600a7288853d  ./ENVIRONMENT.txt
93848d85999588b095c508a4e82f47e05526c42773f6727b8f90cad65cf51296  ./README.md
91fa5b4b0fc9b1af800eba8735c79dffb3d7f49f5fc5b9b8887ff4cd83f5762b  ./v29_constants.json
91fa5b4b0fc9b1af800eba8735c79dffb3d7f49f5fc5b9b8887ff4cd83f5762b  ./v29_constants.json.txt
d2f40a3fdeff871a6990625597fd464bb4e4020930416aeae73b2bf73839f034  ./v29_generate_tail_certificate.
    py
b3e10cdf9d797b0b7ef9d3b2c4c8f0c47068b24be4979a612264ea7a8388ae50  ./v29_tail_certificate.json
b3e10cdf9d797b0b7ef9d3b2c4c8f0c47068b24be4979a612264ea7a8388ae50  ./v29_tail_certificate.json.txt
1670a46567adbb1b69679dc8bad242d25786c4d949a55aed96e6b9e68dc991ed  ./v29_tick_generator_audit.py
666b66ca1dca13d7c759edcaef4a354d22efd9b8a377a8d1f2aeb4d88f1af328  ./v29_verify_tail_certificate.py
929812a3397edf22119200f85d911fcd1727dea3319a862c0e6f098e24381e91  ./v31_check_ledger_format.py
950cc4f59ece16f7009c70550147a398c0251c5a6875e76fc4f699e46e764dfb  ./v31_constants_m10.json
a8536a40c88a2b3b0950152725ceca6310cfa2cff5cd9fdcdb6f07ffe8c19855  ./v31_frontend_certificate.json
c1debbda3583dbaf0dc7120684ba89c457fef1227f4aa13504b21cf11e029acb  ./v31_frontend_verifier_output.
    txt
8bba6177339ab2e5d711bd6db74c1ba95a67e069cf9bd87f28d4f5f3e746363e  ./
    v31_generate_frontend_certificate.py
e95920b0bc6831dfdf5cdbb8472e22d69338384176623882a61744dc17e5ddd5  ./v31_generate_tail_certificate.
    py
216e27bea5296cba960d7a5ac85c1374f2d9931050acda5907c476592fa5da3a  ./v31_tail_certificate_m10.json
4c76c0124dc09ba72ca94fc9376551ab1bcdb56c423270f9e6e189b9f12d557f  ./v31_verifier_output_m10.txt
27bf722529ffaca91643919ae09d78d9ac523aa1a449fddbbf95880fac3828a3  ./v31_verifier_output_m6e12.txt
4bc1967ee844ece2a58af69bc84d497223e5108ddf369712c0e04e5af669e131  ./v31_verify_frontend_certificate
    .py
732efb458af595a341faf3b2727d699925be651be26ad9f8adc0d5eed4917cd7  ./v31_verify_tail_certificate.py
34006b82464026e8e2cf303dd707dc04bf31e932c7b17bdb86e5e19127641996  ./verify_all.sh
```

A.3 Commands

From the directory `v31_repro_pack/`:

1. `sha256sum -c SHA256SUMS.txt`
2. `python3 v31_verify_tail_certificate.py --constants v29_constants.json --certificate v29_tail_certificate.json`
3. `python3 v31_verify_tail_certificate.py --constants v31_constants_m10.json --certificate v31_tail_certificate_m10.json`

A.4 Expected verifier output: $m = 6 \cdot 10^{12}$ (verbatim)

```
m_band = 6000000000000
eta    = 1e-6
alpha  = 1
LHS interval = {'lo':
    '4.24380258847434019390458058905974961069243127420911137611143519277177721326276319815844743E
    -8', 'hi':
    '4.27046745474560981164768110630538666299243686556043572824338915520455458325067899725292224E
    -8'}
RHS interval = {'lo':
    '0.12992563972639583619781214628600466661575620137102480749193490417761535986231651699228296',
     'hi':
    '0.12992564814184654752920869249759011136463063691861340521214709304530071998514932663830003 0'}
CHECK: lhs.hi < rhs.lo : True
PASS
```

A.5 Expected verifier output: $m = 10$ (verbatim)

```
m_band = 10
eta    = 1e-6
alpha  = 1
LHS interval = {'lo':
    '0.000015234618117183977728816943709993041593642697985953816494658857500760927943834366802068230 6',
     'hi':
    '0.000015295523930535839514304394697188122292015566114252487797680042385872496897861554745860440 0'}

RHS interval = {'lo':
    '0.12925751235651552580469614916666369940576283723400886895742977187563564796207246452228193 7',
     'hi':
    '0.12925766152206786103256180378854784139000193703519047294633718909174196433912337543963383 3'}
CHECK: lhs.hi < rhs.lo : True
PASS
```

A.6 Bundle files (verbatim)

```
{
  "certificate_version": "v29",
  "created_utc": "2025-12-13T01:29:44Z",
  "m_band": "6000000000000",
  "eta": "1e-6",
  "alpha_worst": "1",
  "intervals": {
    "C1": {
      "lo": "15.0",
      "hi": "15.1"
```

```
    },
    "C2": {
      "lo": "50.0",
      "hi": "50.1"
    },
    "C_up": {
      "lo": "1100.0",
      "hi": "1100.1"
    },
    "C_hpp": {
      "lo": "1100.0",
      "hi": "1100.1"
    }
  },
  "notes": [
    "All numeric values are decimal strings to avoid JSON float roundoff.",
    "These constants are used by generate_tail_certificate.py and verify_tail_certificate.py.",
    "They are interpreted as closed intervals [lo, hi]."
  ]
}
```

```
{
  "certificate_version": "v29",
  "m_band": "6000000000000",
  "eta": "1e-6",
  "alpha": "1",
  "prec": 90,
  "pi_interval": {
    "lo": "3.14159265358979323846264338327950288419716939937510",
    "hi": "3.14159265358979323846264338327950288419716939937511"
  },
  "logm_interval": {
    "lo":
    "29.422780585156603209028374814593072763936208555728280418255309154741759231316533445311445 5",
    "hi":
    "29.422780585156603209028374814593072763936208555728280418255309154741759231316533445311445 5"
  },
  "delta_interval": {
    "lo":
    "1.155134550010678025928649502893215225740482864575533827374479264909888064530164972931512 95E
    -9",
    "hi":
    "1.155134550010678025928649502893215225740482864575533827374479264909888064530164972931512 96E
    -9"
  },
  "L_interval": {
    "lo":
    "491.34170877734904813542562221889609145904312833592420627382963732112638846974800167967168 2",
    "hi":
    "494.38398683586470845632845970035539873543674919149703431565516823660056439287965502420282 8"
  },
  "lhs_interval": {
    "lo":
    "4.24380258847434019390458058905974961069243127420911137611143519277177721326276319815844743E
    -8",
    "hi":
```

```
    "4.270467454745609811647681106305386662992436865560435728243389155204554583250678997252922 24E
    -8"
  },
  "rhs_interval": {
    "lo":
    "0.1299256397263958361978121462860046666157562013710248074919349041776153598623165169922828 96",
    "hi":
    "0.1299256481418465475292086924975901113646306369186134052121470930453007199851493266383000 30"
  },
  "derived_constants": {
    "ln2_interval": {
      "lo":
    "0.6931471805599453094172321214581765680755001343602552541206800094933936219696947156058633 27",
      "hi":
    "0.6931471805599453094172321214581765680755001343602552541206800094933936219696947156058633 27"
    },
    "c_interval": {
      "lo":
    "0.1299650963549897455157310227734081065141562751925478601476275017800113041193177591760993 73",
      "hi":
    "0.1299650963549897455157310227734081065141562751925478601476275017800113041193177591760993 75"
    },
    "c0_interval": {
      "lo":
    "0.0827383500572443475236711620442491341185086557736206913728528561387020242248387512851407 512",

      "hi":
    "0.0827383500572443475236711620442491341185086557736209547372007536994885577445868650239268 751"
    },
    "Kalloc_interval": {
      "lo":
    "16.856406460551018348219570732046978935542442030483045024446455835615464135270400296649169 5",
      "hi":
    "16.856406460551018348219570732046978935542442030483045024446455835615464135270400296649169 6"
    }
  },
  "pass": true
}



{
  "certificate_version": "v31",
  "created_utc": "2026-01-17T00:00:00Z",
  "m_band": "10",
  "eta": "1e-6",
  "alpha_worst": "1",
  "intervals": {
    "C1": {
      "lo": "15.0",
      "hi": "15.1"
    },
    "C2": {
      "lo": "50.0",
      "hi": "50.1"
    },
    "C_up": {
      "lo": "1100.0",
```

```
      "hi": "1100.1"
    },
    "C_hpp": {
      "lo": "1100.0",
      "hi": "1100.1"
    }
  },
  "notes": [
    "v31 derived constants file: same ledger intervals as v29_constants.json, with m_band=10 for
    low-anchor tail certificate.",
    "All numeric values are decimal strings to avoid JSON float roundoff.",
    "These constants are used by generate_tail_certificate.py and verify_tail_certificate.py.",
    "They are interpreted as closed intervals [lo, hi]."
  ]
}
```

```
{
  "certificate_version": "v29",
  "m_band": "10",
  "eta": "1e-6",
  "alpha": "1",
  "prec": 90,
  "pi_interval": {
    "lo": "3.14159265358979323846264338327950288419716939937510",
    "hi": "3.14159265358979323846264338327950288419716939937511"
  },
  "logm_interval": {
    "lo":
    "2.30258509299404568401799145468436420760110148862877297603332790096757260967735248023599721",
    "hi":
    "2.30258509299404568401799145468436420760110148862877297603332790096757260967735248023599721"
  },
  "delta_interval": {
    "lo":
    "1.88611697011613929219960829965060873665900545176220488941908879591085361622963010761197468E
    -7",
    "hi":
    "1.88611697011613929219960829965060873665900545176220488941908879591085361622963010761197469E
    -7"
  },
  "L_interval": {
    "lo":
    "84.538776394910685260269871820265463114016522329431594640499918514513589145160287203539958 1",
    "hi":
    "84.869034904210089828671670965733899534776632478294471938103251304610346406128022451563557 9"
  },
  "lhs_interval": {
    "lo":
    "0.000015234618117183977728816943709993041593642697985953816494658857500760927943834366802068 2306",

    "hi":
    "0.000015295523930535839514304394697188122292015566114252487797680042385872496897861554745860 4400"

  },
  "rhs_interval": {
```

```
    "lo":
    "0.12925751235651552580469614916666369940576283723400886895742977187563564796207246452228193 7",
    "hi":
    "0.12925766152206786103256180378854784139000193703519047294633718909174196433912337543963383 3"
  },
  "derived_constants": {
    "ln2_interval": {
      "lo":
    "0.69314718055994530941723212145817656807550013436025525412068000949339362196969471560586332 7",
      "hi":
    "0.69314718055994530941723212145817656807550013436025525412068000949339362196969471560586332 7"
    },
    "c_interval": {
      "lo":
    "0.12996509635498974551573102277340810651415627519254786014762750178001130411931775917609937 3",
      "hi":
    "0.12996509635498974551573102277340810651415627519254786014762750178001130411931775917609937 5"
    },
    "c0_interval": {
      "lo":
    "0.08273835005724434752367116204424913411850865577362069137285285613870202422483875128514075 12",

      "hi":
    "0.08273835005724434752367116204424913411850865577362095473720075369948855774458686502392687 51"
    },
    "Kalloc_interval": {
      "lo":
    "16.85640646055101834821957073204697893554244203048304502444645583561546413527040029664916 95",
      "hi":
    "16.85640646055101834821957073204697893554244203048304502444645583561546413527040029664916 96"
    }
  },
  "pass": true
}
```


```
#!/usr/bin/env python3
"""
generate_tail_certificate.py  (v29)

Deterministically generates tail_certificate.json from constants.json using
directed-rounding interval arithmetic implemented with Python's decimal module.

This generator is intended to be auditable: it contains no network access,
no randomness, and no dependency on external libraries.

Usage:
  python3 generate_tail_certificate.py constants.json tail_certificate.json
"""

import json
import sys
from dataclasses import dataclass
from decimal import Decimal, getcontext, localcontext, ROUND_FLOOR, ROUND_CEILING, ROUND_HALF_EVEN

# ---- Fixed enclosure for pi (50 decimal places) ----
# Verified digits: pi = 3.14159265358979323846264338327950288419716939937510...
```

```
# Hence:
PI_LO = Decimal("3.14159265358979323846264338327950288419716939937510")
PI_HI = Decimal("3.14159265358979323846264338327950288419716939937511")

@dataclass
class Interval:
    lo: Decimal
    hi: Decimal
    def __post_init__(self):
        if self.lo > self.hi:
            raise ValueError(f"Bad interval: {self.lo} > {self.hi}")

def ctx(prec: int, rounding):
    c = getcontext().copy()
    c.prec = prec
    c.rounding = rounding
    return c

def iv(lo: str, hi: str = None) -> Interval:
    if hi is None:
        hi = lo
    return Interval(Decimal(lo), Decimal(hi))

def add(a: Interval, b: Interval, prec: int) -> Interval:
    with localcontext(ctx(prec, ROUND_FLOOR)):
        lo = a.lo + b.lo
    with localcontext(ctx(prec, ROUND_CEILING)):
        hi = a.hi + b.hi
    return Interval(lo, hi)

def sub(a: Interval, b: Interval, prec: int) -> Interval:
    with localcontext(ctx(prec, ROUND_FLOOR)):
        lo = a.lo - b.hi
    with localcontext(ctx(prec, ROUND_CEILING)):
        hi = a.hi - b.lo
    return Interval(lo, hi)

def mul(a: Interval, b: Interval, prec: int) -> Interval:
    with localcontext(ctx(prec, ROUND_FLOOR)):
        cands_lo = [a.lo*b.lo, a.lo*b.hi, a.hi*b.lo, a.hi*b.hi]
        lo = min(cands_lo)
    with localcontext(ctx(prec, ROUND_CEILING)):
        cands_hi = [a.lo*b.lo, a.lo*b.hi, a.hi*b.lo, a.hi*b.hi]
        hi = max(cands_hi)
    return Interval(lo, hi)

def div(a: Interval, b: Interval, prec: int) -> Interval:
    if b.lo <= 0 <= b.hi:
        raise ZeroDivisionError("Interval division by an interval containing 0.")
    with localcontext(ctx(prec, ROUND_FLOOR)):
        rlo = Decimal(1) / b.hi
    with localcontext(ctx(prec, ROUND_CEILING)):
        rhi = Decimal(1) / b.lo
    return mul(a, Interval(rlo, rhi), prec)

def sqrt(a: Interval, prec: int) -> Interval:
    if a.lo < 0:
        raise ValueError("sqrt of negative interval")
    with localcontext(ctx(prec, ROUND_FLOOR)):
```

```
        lo = a.lo.sqrt()
    with localcontext(ctx(prec, ROUND_CEILING)):
        hi = a.hi.sqrt()
    return Interval(lo, hi)

def ln(a: Interval, prec: int) -> Interval:
    if a.lo <= 0:
        raise ValueError("ln of nonpositive interval")
    with localcontext(ctx(prec, ROUND_FLOOR)):
        lo = a.lo.ln()
    with localcontext(ctx(prec, ROUND_CEILING)):
        hi = a.hi.ln()
    return Interval(lo, hi)

def pow_3_2(a: Interval, prec: int) -> Interval:
    return mul(a, sqrt(a, prec), prec)

def compute(constants, prec: int = 90):
    m = iv(constants["m_band"])
    eta = iv(constants["eta"])
    alpha = iv(constants["alpha_worst"])

    C1 = iv(constants["intervals"]["C1"]["lo"], constants["intervals"]["C1"]["hi"])
    C2 = iv(constants["intervals"]["C2"]["lo"], constants["intervals"]["C2"]["hi"])
    Cup = iv(constants["intervals"]["C_up"]["lo"], constants["intervals"]["C_up"]["hi"])
    Chpp = iv(constants["intervals"]["C_hpp"]["lo"], constants["intervals"]["C_hpp"]["hi"])

    logm = ln(m, prec)
    delta = div(mul(eta, alpha, prec), mul(logm, logm, prec), prec)

    L = add(mul(C1, logm, prec), C2, prec)

    # ln 2
    ln2 = ln(iv("2"), prec)

    # c = (3 ln 2)/16
    c = div(mul(iv("3"), ln2, prec), iv("16"), prec)

    # c0 = (3 ln 2)/(8 pi), pi enclosed
    pi = Interval(PI_LO, PI_HI)
    c0 = div(mul(iv("3"), ln2, prec), mul(iv("8"), pi, prec), prec)

    # Kalloc = 3 + 8 sqrt(3)
    sqrt3 = sqrt(iv("3"), prec)
    Kalloc = add(iv("3"), mul(iv("8"), sqrt3, prec), prec)

    logm_plus1 = add(logm, iv("1"), prec)

    # LHS = 2*Cup*delta^(3/2)*L
    lhs = mul(mul(mul(iv("2"), Cup, prec), pow_3_2(delta, prec), prec), L, prec)

    # RHS = c - delta*(Kalloc*c0*L + Chpp*(logm+1))
    term1 = mul(mul(Kalloc, c0, prec), L, prec)
    term2 = mul(Chpp, logm_plus1, prec)
    rhs = sub(c, mul(delta, add(term1, term2, prec), prec), prec)

    passed = (lhs.hi < rhs.lo)

    return {
```

```
        "prec": prec,
        "pi_interval": {"lo": str(PI_LO), "hi": str(PI_HI)},
        "logm_interval": {"lo": str(logm.lo), "hi": str(logm.hi)},
        "delta_interval": {"lo": str(delta.lo), "hi": str(delta.hi)},
        "L_interval": {"lo": str(L.lo), "hi": str(L.hi)},
        "lhs_interval": {"lo": str(lhs.lo), "hi": str(lhs.hi)},
        "rhs_interval": {"lo": str(rhs.lo), "hi": str(rhs.hi)},
        "derived_constants": {
            "ln2_interval": {"lo": str(ln2.lo), "hi": str(ln2.hi)},
            "c_interval": {"lo": str(c.lo), "hi": str(c.hi)},
            "c0_interval": {"lo": str(c0.lo), "hi": str(c0.hi)},
            "Kalloc_interval": {"lo": str(Kalloc.lo), "hi": str(Kalloc.hi)},
        },
        "pass": bool(passed),
    }

def main():
    if len(sys.argv) != 3:
        print("Usage: generate_tail_certificate.py constants.json tail_certificate.json", file=sys.
    stderr)
        sys.exit(2)

    with open(sys.argv[1], "r", encoding="utf-8") as f:
        constants = json.load(f)

    out = {
        "certificate_version": "v29",
        "m_band": constants["m_band"],
        "eta": constants["eta"],
        "alpha": constants["alpha_worst"],
    }
    out.update(compute(constants, prec=90))

    with open(sys.argv[2], "w", encoding="utf-8") as f:
        json.dump(out, f, indent=2)

    print("[generate] wrote", sys.argv[2])
    print("[generate] PASS =", out["pass"])
    print("[generate] lhs_interval.hi =", out["lhs_interval"]["hi"])
    print("[generate] rhs_interval.lo =", out["rhs_interval"]["lo"])

if __name__ == "__main__":
    main()
```

```
#!/usr/bin/env python3
"""v31_generate_tail_certificate.py

Canonical generator entrypoint shipped with manuscript v31.

This is a thin wrapper around the v29 generator implementation
(v29_generate_tail_certificate.py). It exists so that v31 provides stable,
versioned entrypoints without modifying the v29 artifacts.

Properties:
- deterministic (no network, no randomness)
- uses directed-rounding interval arithmetic (Decimal ROUND_FLOOR/ROUND_CEILING)
```

```
  as implemented in the v29 generator

CLI:
  python3 v31_generate_tail_certificate.py \
    --constants v29_constants.json \
    --out regen_tail_certificate.json

Notes:
- The JSON input may be named *.json or *.json.txt; it is parsed as JSON either way.
- The output is intended to match the v29 pinned certificate exactly when run
  with the same constants file.
"""

from __future__ import annotations

import argparse
import importlib.util
import json
import os
from typing import Any, Dict


def _load_module_from_path(module_name: str, path: str):
    spec = importlib.util.spec_from_file_location(module_name, path)
    if spec is None or spec.loader is None:
        raise ImportError(f"Unable to load module {module_name} from {path}")
    module = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(module)  # type: ignore[attr-defined]
    return module


def _load_json(path: str) -> Dict[str, Any]:
    with open(path, "r", encoding="utf-8") as f:
        return json.load(f)


def main() -> int:
    ap = argparse.ArgumentParser(description="Generate tail_certificate.json deterministically (v31
     wrapper around v29).")
    ap.add_argument("--constants", required=True, help="Path to constants JSON (e.g., v29_constants
    .json or v29_constants.json.txt)")
    ap.add_argument("--out", required=True, help="Output path for generated certificate JSON")
    ap.add_argument(
        "--v29-generator",
        default=os.path.join(os.path.dirname(__file__), "v29_generate_tail_certificate.py"),
        help="Path to v29_generate_tail_certificate.py (default: alongside this script)",
    )
    ap.add_argument("--prec", type=int, default=90, help="Decimal precision used by the generator (
    default: 90)")
    args = ap.parse_args()

    constants = _load_json(args.constants)

    gen = _load_module_from_path("v29_generate_tail_certificate", args.v29_generator)

    out: Dict[str, Any] = {
        "certificate_version": "v29",
        "m_band": constants["m_band"],
        "eta": constants["eta"],
```

```
        "alpha": constants["alpha_worst"],
    }
    out.update(gen.compute(constants, prec=args.prec))

    with open(args.out, "w", encoding="utf-8") as f:
        json.dump(out, f, indent=2)

    print("[v31_generate] wrote", args.out)
    print("[v31_generate] PASS =", bool(out.get("pass")))
    print("[v31_generate] lhs_interval.hi =", out["lhs_interval"]["hi"])
    print("[v31_generate] rhs_interval.lo =", out["rhs_interval"]["lo"])

    return 0


if __name__ == "__main__":
    raise SystemExit(main())

#!/usr/bin/env python3
"""v31_verify_tail_certificate.py

Canonical verifier entrypoint shipped with manuscript v31.

This verifier:
1) Regenerates a fresh tail certificate from a constants JSON file using the v29
   directed-rounding generator implementation.
2) Checks that the provided/pinned certificate matches the regenerated certificate
   on a fixed set of required fields.
3) Checks the strict interval inequality:
       lhs_interval.hi < rhs_interval.lo

Exit codes:
- 0 on PASS
- nonzero on FAIL

Usage:
  python3 v31_verify_tail_certificate.py \
    --constants v29_constants.json \
    --certificate v29_tail_certificate.json

Notes:
- The JSON files may be named *.json or *.json.txt; they are parsed as JSON either way.
"""

from __future__ import annotations

import argparse
import importlib.util
import json
import os
from decimal import Decimal
from typing import Any, Dict, List


def _load_module_from_path(module_name: str, path: str):
    spec = importlib.util.spec_from_file_location(module_name, path)
    if spec is None or spec.loader is None:
        raise ImportError(f"Unable to load module {module_name} from {path}")
    module = importlib.util.module_from_spec(spec)
```

```
    spec.loader.exec_module(module)  # type: ignore[attr-defined]
    return module


def _load_json(path: str) -> Dict[str, Any]:
    with open(path, "r", encoding="utf-8") as f:
        return json.load(f)


def main() -> int:
    ap = argparse.ArgumentParser(description="Verify the v29 tail certificate using a v31 stable
    entrypoint.")
    ap.add_argument("--constants", required=True, help="Path to constants JSON (e.g., v29_constants
    .json or v29_constants.json.txt)")
    ap.add_argument("--certificate", required=True, help="Path to pinned tail certificate JSON (e.g
    ., v29_tail_certificate.json or *.json.txt)")
    ap.add_argument(
        "--generator",
        default=os.path.join(os.path.dirname(__file__), "v29_generate_tail_certificate.py"),
        help="Path to v29_generate_tail_certificate.py (default: alongside this script)",
    )
    ap.add_argument("--prec", type=int, default=90, help="Decimal precision used by the generator (
    default: 90)")
    args = ap.parse_args()

    constants = _load_json(args.constants)
    cert = _load_json(args.certificate)

    gen = _load_module_from_path("v29_generate_tail_certificate", args.generator)

    regen: Dict[str, Any] = {
        "certificate_version": "v29",
        "m_band": constants["m_band"],
        "eta": constants["eta"],
        "alpha": constants["alpha_worst"],
    }
    regen.update(gen.compute(constants, prec=args.prec))

    required_keys: List[str] = [
        "certificate_version",
        "m_band",
        "eta",
        "alpha",
        "prec",
        "pi_interval",
        "logm_interval",
        "delta_interval",
        "L_interval",
        "lhs_interval",
        "rhs_interval",
        "derived_constants",
        "pass",
    ]

    mism: List[str] = [k for k in required_keys if regen.get(k) != cert.get(k)]

    lhs_hi = Decimal(str(cert["lhs_interval"]["hi"]))
    rhs_lo = Decimal(str(cert["rhs_interval"]["lo"]))
    strict_ok = lhs_hi < rhs_lo
```

```
    print("m_band =", cert.get("m_band"))
    print("eta    =", cert.get("eta"))
    print("alpha  =", cert.get("alpha"))
    print("LHS interval =", cert.get("lhs_interval"))
    print("RHS interval =", cert.get("rhs_interval"))
    print(f"CHECK: lhs.hi < rhs.lo : {strict_ok}")

    if mism:
        print("FAIL: certificate mismatch in keys:", ", ".join(mism))
        return 1
    if not cert.get("pass", False):
        print("FAIL: certificate 'pass' field is false")
        return 1
    if not strict_ok:
        print("FAIL: inequality does not hold")
        return 1

    print("PASS")
    return 0


if __name__ == "__main__":
    raise SystemExit(main())
```

B Finite-height front-end certificate (literature-based)

The required front-end for v31 is RH up to height $H_0 = 5$. We record a discharge using Platt–Trudgian's published verification of RH up to $3 \cdot 10^{12}$.

```
{
  "certificate_version": "v31",
  "created_utc": "2026-01-17T15:29:50Z",
  "needed_frontend_statement": {
    "type": "RH_to_height",
    "H0": 5.0,
    "text": "All nontrivial zeros rho=beta+i gamma with 0<gamma<=H0 satisfy beta=1/2."
  },
  "discharged_by": {
    "type": "literature_citation",
    "verification_height": 3000000000000.0,
    "reference": {
      "authors": "D. J. Platt and T. S. Trudgian",
      "title": "The Riemann hypothesis is true up to 3*10^12",
      "venue": "Bulletin of the London Mathematical Society",
      "year": 2021,
      "doi": "10.1112/blms.12460",
      "arxiv": "2004.09765",
      "statement": "All zeros beta+i gamma with 0<gamma<=3*10^12 satisfy beta=1/2 (rigorous
    interval arithmetic)."
    },
    "logic": "If RH holds for 0<gamma<=H_cited and H0<=H_cited, then RH holds for 0<gamma<=H0."
  },
```

```
  "notes": [
    "This JSON is not itself a computation of zeros; it is a pinned statement+reference used by v31
    .",
    "For a fully self-contained proof without external computational input, one would need to
    implement and certify an argument-principle zero count in this region using ball arithmetic (
    not provided here)."
  ]
}
```

```
H0 (needed) = 5.0
H_cited     = 3000000000000.0
CHECK: H0 <= H_cited : True
PASS
```

```
#!/usr/bin/env python3
"""v31_generate_frontend_certificate.py

Generates a small, explicit "front-end" certificate JSON for the low-height region.

Important: This script does NOT attempt to re-run any large-scale RH verification
computation (e.g. Platt--Trudgian). Instead, it records:
  - the exact finite-height statement needed by manuscript v31, and
  - the external published verification result used to discharge it.

This is consistent with standard mathematical practice: an externally published,
peer-reviewed, rigorous computation can be cited as an input.

Usage:
  python3 v31_generate_frontend_certificate.py --H0 5 --out v31_frontend_certificate.json

The corresponding verifier script v31_verify_frontend_certificate.py checks that
H0 is indeed within the cited verification height.
"""

from __future__ import annotations

import argparse
import json
from datetime import datetime, timezone


# Published rigorous verification height used for discharge.
# Platt--Trudgian (BLMS 2021) proves RH for all zeros with 0 < Im(s) <= 3e12.
H_CITED = 3_000_000_000_000.0

REFERENCE = {
    "authors": "D. J. Platt and T. S. Trudgian",
    "title": "The Riemann hypothesis is true up to 3*10^12",
    "venue": "Bulletin of the London Mathematical Society",
    "year": 2021,
    "doi": "10.1112/blms.12460",
    "arxiv": "2004.09765",
    "statement": "All zeros beta+i gamma with 0<gamma<=3*10^12 satisfy beta=1/2 (rigorous interval
    arithmetic).",
```

```
}


def main() -> int:
    ap = argparse.ArgumentParser(description="Generate a front-end certificate JSON (v31).")
    ap.add_argument("--H0", type=float, required=True, help="Finite height in the s-plane to be
    discharged (e.g. H0=5 when m*=10).")
    ap.add_argument("--out", required=True, help="Output JSON path")
    args = ap.parse_args()

    out = {
        "certificate_version": "v31",
        "created_utc": datetime.now(timezone.utc).strftime("%Y-%m-%dT%H:%M:%SZ"),
        "needed_frontend_statement": {
            "type": "RH_to_height",
            "H0": args.H0,
            "text": "All nontrivial zeros rho=beta+i gamma with 0<gamma<=H0 satisfy beta=1/2.",
        },
        "discharged_by": {
            "type": "literature_citation",
            "verification_height": H_CITED,
            "reference": REFERENCE,
            "logic": "If RH holds for 0<gamma<=H_cited and H0<=H_cited, then RH holds for 0<gamma<=
    H0.",
        },
        "notes": [
            "This JSON is not itself a computation of zeros; it is a pinned statement+reference
    used by v31.",
            "For a fully self-contained proof without external computational input, one would need
    to implement and certify an argument-principle zero count in this region using ball arithmetic
    (not provided here).",
        ],
    }

    with open(args.out, "w", encoding="utf-8") as f:
        json.dump(out, f, indent=2)

    print("[v31_frontend_generate] wrote", args.out)
    print("[v31_frontend_generate] H0 =", args.H0)
    print("[v31_frontend_generate] discharged by verification height =", H_CITED)
    return 0


if __name__ == "__main__":
    raise SystemExit(main())

#!/usr/bin/env python3
"""v31_verify_frontend_certificate.py

Verifier for the front-end certificate JSON produced by v31_generate_frontend_certificate.py.

This verifier checks the internal logic only:
- parses the JSON
- confirms that the required finite-height H0 is <= the cited verification height

It does NOT re-run the cited large-scale computation (Platt--Trudgian); that result is treated as
    an
external, peer-reviewed input in the manuscript.
```

```
Usage:
  python3 v31_verify_frontend_certificate.py --certificate v31_frontend_certificate.json

Exit codes:
- 0 on PASS
- nonzero on FAIL
"""

from __future__ import annotations

import argparse
import json


def main() -> int:
    ap = argparse.ArgumentParser(description="Verify v31 front-end certificate JSON (internal logic
     only).")
    ap.add_argument("--certificate", required=True, help="Path to v31_frontend_certificate.json")
    args = ap.parse_args()

    with open(args.certificate, "r", encoding="utf-8") as f:
        cert = json.load(f)

    needed = cert.get("needed_frontend_statement", {})
    discharged = cert.get("discharged_by", {})

    H0 = float(needed.get("H0"))
    Hc = float(discharged.get("verification_height"))

    ok = H0 <= Hc

    print("H0 (needed) =", H0)
    print("H_cited     =", Hc)
    print(f"CHECK: H0 <= H_cited : {ok}")

    if not ok:
        print("FAIL")
        return 1

    print("PASS")
    return 0


if __name__ == "__main__":
    raise SystemExit(main())
```

C Ledger certification plan (not yet discharged in v31)

This appendix records a concrete plan to certify the ledger constants. No claim of completion is made in v31.

Targets

The ledger constants are the interval enclosures for:

- C_1, C_2: residual envelope bound (Lemma 7.1) for $\sup_{\partial B} |F'/F|$.
- C_{up}: shape-only constant in Lemma 10.2.
- C_h'': shape-only constant in Lemma 10.3.

Certification plan (high-level)

1. **Mathematical specifications.** Restate each lemma with the constant defined as an explicit supremum/operator norm on the normalized square boundary ∂Q (for the shape-only constants) and with an explicit zero-removal and residual formulation for the analytic constants.
2. **Certified computations (recommended).** Use ball/interval arithmetic (e.g. Arb/Acb) to compute rigorous enclosures for:
   - the relevant operator norms on ∂Q (discretize, bound quadrature error, and use a stability argument for the discretization-to-continuum gap), and
   - the residual envelope bound constants for F'/F (reduce to explicit kernels plus a finite zero list and an explicit bound for the remaining zeta contribution).
3. **Reproducibility and pinning.** Provide a deterministic build (Docker image digest or lockfile), one-command scripts that output JSON interval enclosures, and SHA–256 hashes of code, inputs, outputs, and environment.

Internal self-check (format only)

The repro pack includes a small self-check script that validates internal consistency of a ledger JSON (lo/hi format, required keys):

```
#!/usr/bin/env python3
"""v31_check_ledger_format.py

Self-consistency check for a constants-ledger JSON.

This script is NOT an independent certification of the mathematical truth of the
interval enclosures. It only checks that the file is syntactically well-formed
and internally consistent (interval endpoints parse as decimals and satisfy lo<=hi).

Usage:
  python3 v31_check_ledger_format.py --constants v29_constants.json

Exit codes:
- 0 on PASS
- nonzero on FAIL
"""

from __future__ import annotations

import argparse
import json
from decimal import Decimal
```

```
REQUIRED_INTERVALS = ["C1", "C2", "C_up", "C_hpp"]


def main() -> int:
    ap = argparse.ArgumentParser(description="Check internal consistency of a constants ledger JSON
     (v31).")
    ap.add_argument("--constants", required=True, help="Path to constants JSON")
    args = ap.parse_args()

    with open(args.constants, "r", encoding="utf-8") as f:
        c = json.load(f)

    if "intervals" not in c:
        print("FAIL: missing key 'intervals'")
        return 1

    intervals = c["intervals"]
    missing = [k for k in REQUIRED_INTERVALS if k not in intervals]
    if missing:
        print("FAIL: missing required intervals:", ", ".join(missing))
        return 1

    ok = True
    for k in REQUIRED_INTERVALS:
        lo_s = intervals[k]["lo"]
        hi_s = intervals[k]["hi"]
        lo = Decimal(str(lo_s))
        hi = Decimal(str(hi_s))
        if lo > hi:
            print(f"FAIL: interval {k} has lo>hi: {lo} > {hi}")
            ok = False
        else:
            print(f"{k}: [{lo}, {hi}]")

    if not ok:
        return 1

    print("PASS")
    return 0


if __name__ == "__main__":
    raise SystemExit(main())
```

References

References

[1] D. J. Platt and T. S. Trudgian, *The Riemann hypothesis is true up to* $3 \cdot 10^{12}$, Bulletin of the London Mathematical Society **53** (2021), no. 3, 792–797. DOI: 10.1112/blms.12460. arXiv:2004.09765.