ACF 86 — Verification Dossier

Exchangeability and the Exclusion of Equilibrium Assembly

Necessary Conditions on Throughput, Entropy, and Fidelity in the Chrysene Formalism

Charles D. Schaper, Ph.D., Schaper Systems Press

Annals of the Chrysene Formalism, Vol. 1, Issue 4

Verification script: `verify_A86.py` Cross-check: `crosscheck_A86.py` Lean proof: `acf-86-proof.lean`

Certification Summary

Item	Value
Script	`verify_A86.py`
Total checks	157
Passed	**157**
Failed	**0**
Exit code	0
Blocks	12 (D1–D12)
Blocks using floating point	**0**
Dependencies	Python standard library only
Cross-check script	`crosscheck_A86.py`
Published macros read	83
Compared	83
Agree	**83**
Disagree	**0**
Figures corrected before publication	4 (recorded)
Lean proof file	`acf-86-proof.lean`
Lean version	`4.12.0` (core only, no Mathlib)
Lean theorems	137
Theorems with no axioms	**137**
`sorry`	0
`native_decide`	0

Four figures were corrected before publication, and this dossier is the record

This harness disagreed with the submitted draft in four places. All four had the same cause, none of them changed a conclusion, and the worst of them was an integer the corpus already prints twice. The article has been revised; the record is kept here because a correction absorbed is a correction lost.

Article 86 is a bounds paper. Its inputs are published figures of Articles 65, 78, 80, 84 and 85, and the draft's generator — as that generator's appendix stated — reproduced them as floating-point literals rather than recomputing them. Inverting a five-significant-figure literal is not inverting the quantity it approximates, and block D9 is where the difference showed:

Macro	Draft	Now	Cause
`VceilPure`	4,044,162	**4,044,121**	$1/\sigma(0)$ is exactly 2011^2
`VrealNineNine`	5,405	**5,392**	reciprocal of a 3-figure literal
`VrealNineFive`	233	**234**	the same
`VrealisedSigmaNineFive`	4.30×10^{-3}	$\mathbf{4.27 \times 10^{-3}}$	a 2-place decimal given a 3rd figure

`VceilPure` was the one worth correcting first. Article 85 proves $\sigma(0) = f_{\mathcal{I}}^2 = 1/2011^2$ *exactly*, so the sensing bound at the pure limit is exactly $2011^2 = 4{,}044{,}121$ contacts per bit — an integer, and the same integer Articles 84 and 85 both print. The draft's 4,044,162 was the reciprocal of the rounded literal 2.4727×10^{-7} and exceeded the exact figure by 41.

The other three were reciprocals of three-significant-figure literals of Article 85's realised capacity, and the last was that paper's two-place decimal 0.0043 re-expressed with a third significant figure it never had.

None of the four disturbed the article. The bound is still $1/\sigma$ and still free of Δ; the $\Delta = 1$ evaluation is still harsher than the bound at every operating point, which check D9-16 certifies on the derived figures; and the orientation floor, the thermodynamic floor, the demand, the throughput and the in-plane count are untouched.

What changed in the article. Its generator, `verify_bounds.py`, now recomputes Article 85's σ and its realised capacities from that paper's definitions over the same 784 ordered contacts this harness uses, so the reciprocals of its block B7 are taken of exact values; its Verification appendix moves those two objects from the imported list to a *derived* one; and its Remark "A correction: four reciprocals" records the four figures with their superseded values. Regenerating the macro file changed those four and left the other 79 byte-identical.

Block D12 of this harness and section S10 of the proof file keep the superseded figures beside the exact ones, and `crosscheck_A86.py` prints them as a record after its result. Every disagreement that script now reports is a new finding, and it exits non-zero on one.

Exact throughout, by three devices and one enclosure

There is no floating-point arithmetic anywhere in `verify_A86.py`. Article 86's quantities divide in three, and the division is cleaner than Article 85's.

Most of them are ratios of integers. Article 65's demand formula at $\alpha = 1/10$ gives $5L + N/9 + \nu L$, and $N/9 = 28{,}000$ is an integer, so the whole per-period demand is: 42,200 exactly at $L = 200$ and 43,762 exactly at $L = 222$. The shares, the rates, the intervals, the bits per period and the orientation floors are all exact rationals and are certified by the rounding bound.

Four are n-th roots of rationals — the error budget of D7 — and they are pinned by the **integer root bracket**, which is this harness's characteristic device and the third inversion of the family:

Article	Device	What the bracket does
82	`antilog_rounds`	raised to clear a logarithm
84	`log2_rounds`	raised to form a logarithm
86	`root_rounds`	**raised to clear a root**

Writing $y = 1 - (a/b)^{1/n}$, the statement that y rounds to r/s is $\frac{2r-1}{2s} < y < \frac{2r+1}{2s}$, hence $\frac{2s-2r-1}{2s} < (a/b)^{1/n} < \frac{2s-2r+1}{2s}$, and $x \mapsto x^n$ is increasing on the positives, so raising the bracket to the n clears the root and leaves a comparison of two exact integers. The root is never formed. Certifying the anchor budget at $n = 1000$ compares two integers of about 7,300 digits, which Python's arbitrary-precision integers settle exactly and no floating-point computation can.

The remainder involve $\ln 2$, or a logarithm of a quantity that is itself bracketed, and those carry certified rational enclosures: block D1 computes $\ln 2$ and $\ln 10$ from the inverse-hyperbolic-tangent series with explicit remainder bounds, and every operation rounds OUTWARD onto the dyadic grid of multiples of 2^{-400}, so each interval provably contains its value. A figure is then certified by showing that the *whole* enclosure falls inside its rounding cell.

Block D1 validates all five devices before any figure of the article is touched — twenty-two checks on the devices, including that $\ln 2$ and $\ln 10$ lie strictly inside the decimal brackets of their first fifteen places, that the root bracket accepts an exact root and an inexact one and rejects the neighbours of both, that the root bracket and the root enclosure agree on the anchor budget, and that the enclosure device refuses an interval too wide to decide a rounding.

Purpose and Scope

This dossier accompanies Article 86, "Exchangeability and the Exclusion of Equilibrium Assembly," which asks what any placement system for the substituted-chrysene assembly must satisfy, and answers it as six bounds rather than as a design. The corpus's refusal to invent mechanism is what makes it a bounds paper: no sorter is proposed anywhere in it.
The verification script executes 157 independent checks in 12 blocks (D1–D12): the devices and their non-vacuity; the imported objects, *derived*; Article 65's demand and its allocation; throughput against Article 64's twelve-hour period; the information bound at Article 84's own hexamer figure; the thermodynamic floor by Landauer; the error budget by the root bracket; the orientation floor — the central block, and the one bound equilibrium cannot meet; the sensing bound, with the three figures it corrects; the in-plane bound; which bound binds; and the scope. All 157 checks passed with exit code 0.
The companion Lean 4 proof file `acf-86-proof.lean` provides kernel-checked formal certificates for 137 claims, which is nearly all of the article: see the proof dossier for the one quantity it leaves behind.

Block Index

Block	Topic	Checks
D1	The devices, and their non-vacuity	22
D2	The imported objects, derived rather than transcribed	17
D3	The demand, its allocation, and the genome length (A65, Thm. 8.5)	13
D4	Throughput, against Article 64's twelve-hour period	9
D5	The information bound, at Article 84's own figures	11
D6	The thermodynamic floor, by Landauer	10
D7	The error budget, by the integer root bracket	12
D8	The orientation floor: what equilibrium cannot do	25
D9	The sensing bound, and three figures it corrects	19
D10	The in-plane bound (A40, eq. 107)	7
D11	Which bound binds	7
D12	Scope, the corrections, and the review record (RECORDED)	5
	Total	**157**

Scope of Verification

What is verified.

- That Article 85's sensitivity polynomial is *recovered* from the definitions — degree eight, rational coefficients, $\sigma(0)$ exactly $1/2011^2$ — and that its values at the three operating points round to the figures the article prints; and that Article 84's coded-hexamer bracket is *re-derived* by the binary-logarithm bracket from the same admitted fractions, giving 25.0 bits at $\rho^* = 100$ and 28.0 at $\rho^* = 200$ and neither to the neighbouring tenth. Nothing that can be derived is imported.

- That Article 65's demand is exactly 42,200 units per period at $L = 200$ and exactly 43,762 at Article 80's critical length; that the three terms are exactly 1,000 anchor, 28,000 residue and 13,200 lipid and sum to the demand; and that **over ninety-seven per cent of the per-period demand is not placed into the heritable structure at all**, which is the fact that makes the anchor term the only one the information and energy bounds apply to.

- That the placement rate is of order one unit per second — and that this is tested as an order of magnitude over four combinations of period and genome length, not as a rounded equality, because the same block computes the six- and twenty-four-hour alternatives and the claim must survive them.

- That the registry term of Article 84's hexamer, 5/6 bits per unit, IS the contacts-per-node ratio of Article 80's box, 45/54 — exactly and not approximately, both being $(\ell - 1)/\ell$ at $\ell = 6$. And that Article 80's 54 is the node count of a $3 \times 3 \times 6$ box, so the box carries 45 stacking contacts and not 53.

- That one bit costs $\ln 2\, kT$ and that the per-unit selection cost of 2.89 to 3.23 kT is **comparable and not slack**: it exceeds one disubstituted contact at $\Delta = 1$ and falls below one tetrasubstituted one, at 1.62 and 0.81 of them. The claim is a pair of strict inequalities against the certified enclosure of $\ln 2$, so it is a statement about the constant and not about a rounded value of it.

- **The error budget, by the root bracket**, at both confidence levels and both placement counts, with every neighbouring mantissa rejected; and the one structural property claimed, that the budget tightens with the number of placements, cross-multiplied through the monotonicity of x^n so that no root is formed there either.

- **The orientation floor.** That every delivered $\mathcal{D}$ or $\mathcal{T}$ contact carries equal registration number in both alignments, so its induced law is Bernoulli(1/2) at *every* Δ — equality of the registration numbers is what is checked, so no exponential is formed and the conclusion is not a limit. That with the sign symmetric a sensitive contact's error rate is $\frac{1}{2}\theta + \frac{1}{2}(1-\theta) = \frac{1}{2}$ *identically in* θ, which needs no value of Δ, of the gap, or of σ. That the $\mathcal{I}$–$\mathcal{I}$ contact is the one exception and the one place the sign is fixed, reading three aligned against two unaligned. And that the floor is therefore exactly $\frac{1}{2} - f_{\mathcal{I}}^2/2$, short of one half by 1.24×10^{-7} and exceeding the error budget by 4.97×10^4 — 4.7 orders of magnitude, *uniformly in* Δ *and in* σ, neither of which appears in the floor.

- That the gap survives the opposite extreme of the independence hypothesis: under perfectly correlated failures the budget is $1 - F = 0.01$ and the floor still exceeds it fiftyfold, so the comparison does not rest on H_{indep}.

- That the sensing bound is $1/\sigma$ and *free of* Δ, and that Article 85's $\Delta = 1$ figures are an evaluation and not a bound — the evaluation being harsher at every operating point, the bound approached from above as Δ grows. Both are reported, on the derived figures, and the article must not quote the evaluation as the bound.

- That the in-plane connector of Article 40's eq. (107) joins cells at the same z, so the relative orientation freedom within an in-plane pair is zero and what survives is one flip per connected component — six components for a six-layer box, not one, giving 64 states and not 2^{54}.

What is not verified. Not the data, because there is none: the article's inputs are published figures of the corpus, and D2 derives the two that can be derived. What is not reached is Δ, and three further limits belong with it.

- **Δ is nowhere measured in the corpus.** Article 79's registration free energy is its least constrained parameter. That is precisely why the two bounds the article calls binding are the two Δ-free ones: the sensing bound is $1/\sigma$ and the orientation floor is one half, and neither contains Δ at all. The harness certifies both without a value for it, and the figures that *do* depend on it — Article 85's realised capacities — are marked as evaluations.

- **No placement mechanism is proposed, anywhere.** The article bounds what a placement system must satisfy; it does not exhibit one, and the corpus's refusal to invent mechanism is deliberate. A reader looking for a sorter will not find one, and check D12 records that rather than verifying it.

- **Two of the six bounds are not of the same kind as the other four.** The error budget

of D7 makes *no thermodynamic claim*: it bounds a per-placement error rate and nothing else, and the translation into a free energy is the subject of D8, where the finding is that for the orientation component no translation exists. And the throughput bound of D4 is an order of magnitude, not a figure.

- **The harness is a regression test on the conclusions.** Block D8 fails if the orientation floor does not exceed the budget, and D9 if the evaluation is not harsher than the bound. That is intended — a published verification artifact should stop passing when the claims it supports stop holding — but it means the file is not an independent test of the conclusions it encodes.

Script Listing: `verify_A86.py`

verify_A86.py — complete source

```
#!/usr/bin/env python3
# ======================================================================
#  verify_A86.py
#  Verification harness for
#    Annals of the Chrysene Formalism, Article 86
#    "The Placement Budget: What Any Placement System Must Satisfy"
#    Charles D. Schaper, Ph.D., Schaper Systems Press
#
#  Python standard library only.  No third-party dependency, no network,
#  no data file.
#
#  TWO THINGS DISTINGUISH THIS HARNESS FROM THE ARTICLE'S OWN GENERATOR,
#  AND BOTH ARE STATED BEFORE ANY FIGURE IS CHECKED.
#
#  (1) IT DERIVES WHAT THE ARTICLE IMPORTS.  Article 86 is a bounds
#  paper: its inputs are published figures of Articles 65, 78, 80, 84 and
#  85, and its own generator reproduces them as floating-point literals
#  and says so in its appendix.  This harness does not.  It recomputes
#  Article 85's sensitivity polynomial from the definitions and evaluates
#  it exactly, and it re-derives Article 84's coded-hexamer bracket from
#  the admitted fractions by the binary-logarithm bracket.  Nothing that
#  can be derived is imported.
#
#  (2) THAT CHANGED FOUR FIGURES OF THE SUBMITTED DRAFT.  Inverting a
#  five-significant-figure literal is not the same as inverting the
#  quantity it approximates, and in block D9 it was not even close:
#  1/sigma at the pure limit is EXACTLY 2011^2 = 4,044,121, an integer
#  the corpus already prints twice, and the draft's 4,044,162 was the
#  reciprocal of the rounded literal.  The article has since been
#  revised: its generator now derives sigma and the realised capacities,
#  and its Remark "A correction: four reciprocals" records the four
#  figures that changed.  Blocks D9 and D12 keep both the superseded and
#  the exact values, because a correction recorded is worth more than a
#  correction absorbed.
#
#  EXACTNESS.  Article 86's quantities divide in three.  Most are ratios
#  of integers and are checked by the rounding bound.  Four are n-th
#  roots of rationals -- the error budget of D7 -- and those are pinned
#  by an INTEGER ROOT BRACKET, which is this harness's characteristic
#  device and the third inversion of the family: Article 82 raises a
#  bracket to clear a logarithm, Article 84 raises one to form a
#  logarithm, and this one raises one to clear a root.  The remainder
#  involve ln 2 or a logarithm of a bracketed quantity, and those carry
#  certified rational enclosures built from series with explicit
#  remainder bounds.
#
#  There is no floating-point arithmetic anywhere in this file.
#
#  Usage:
#      python3 verify_A86.py        summary only
#      python3 verify_A86.py -v     every check, with its identifier
#
#  Exit code 0 if and only if every check passes.
# ======================================================================
```

```
from fractions import Fraction
from itertools import product
import sys

VERBOSE = "-v" in sys.argv[1:] or "--verbose" in sys.argv[1:]

PASSED = 0
FAILED = 0
FAILURES = []
_BLOCK = ""
_N = 0


def block(name, title):
    global _BLOCK, _N
    _BLOCK = name
    _N = 0
    print("\n%s  %s" % (name, title))


def check(claim, got, want):
    global PASSED, FAILED, _N
    _N += 1
    ident = "%s-%02d" % (_BLOCK, _N)
    ok = (got == want)
    if ok:
        PASSED += 1
        if VERBOSE:
            print("    [%s] PASS  %s" % (ident, claim))
    else:
        FAILED += 1
        FAILURES.append((ident, claim, got, want))
        print("    [%s] FAIL  %s" % (ident, claim))
        print("             got  %r" % (got,))
        print("             want %r" % (want,))


def note(text):
    if VERBOSE:
        print("    ---          %s" % text)


def show(text):
    print("    %s" % text)


def dec(x, places):
    """Decimal string of an exact Fraction, rounded half up.  Formatting
    only, and the rounding matches the devices so that a printed line and
    the check that certifies it cannot disagree."""
    x = Fraction(x)
    neg = x < 0
    if neg:
        x = -x
    scale = 10 ** places
    n = (2 * x.numerator * scale + x.denominator) // (2 * x.denominator)
    s = "%d" % n if places == 0 else "%d.%0*d" % (n // scale, places,
                                                  n % scale)
```

```
    return ("-" + s) if neg else s


def expo(x, sig=2):
    x = Fraction(x)
    if x == 0:
        return "0"
    e = 0
    y = x
    while y >= 10:
        y /= 10
        e += 1
    while y < 1:
        y *= 10
        e -= 1
    s = dec(y, sig)
    if s.startswith("10"):
        y /= 10
        e += 1
        s = dec(y, sig)
    return "%se%+03d" % (s, e)


def grp(n):
    return "{:,}".format(int(n))


# ====================================================================
#  D1.  The devices
# ====================================================================
#
#    rounds_to(n,d,r,s)         n/d rounds to r/s        2|ns - rd| < d
#    sci_rounds(n,d,m,c,e)      n/d rounds to (m/c)10^e
#    log2_rounds(n,d,r,s)       log2(n/d) rounds to r/s
#    root_rounds(a,b,k,r,s)     1 - (a/b)^(1/k) rounds to r/s
#    rounds_to_iv(v,r,s)        the whole enclosure rounds to r/s
#
#  The fourth is this harness's characteristic device.

PREC = 400
GRID = 1 << PREC


def _dn(x):
    return Fraction(x.numerator * GRID // x.denominator, GRID)


def _up(x):
    return Fraction(-((-x.numerator) * GRID // x.denominator), GRID)


class Iv:
    """A closed interval of rationals.  Every operation rounds outward,
    so an Iv provably contains the true value of the expression that
    built it, and no operation ever narrows one."""

    __slots__ = ("lo", "hi")

    def __init__(self, lo, hi=None, _raw=False):
```

```
        if hi is None:
            hi = lo
        if _raw:
            self.lo, self.hi = lo, hi
            return
        lo, hi = Fraction(lo), Fraction(hi)
        if lo > hi:
            raise ValueError("inverted interval")
        self.lo, self.hi = _dn(lo), _up(hi)

    def __add__(self, o):
        o = iv(o)
        return Iv(self.lo + o.lo, self.hi + o.hi)

    def __sub__(self, o):
        o = iv(o)
        return Iv(self.lo - o.hi, self.hi - o.lo)

    def __neg__(self):
        return Iv(-self.hi, -self.lo, _raw=True)

    def __mul__(self, o):
        o = iv(o)
        a, b, c, d = self.lo, self.hi, o.lo, o.hi
        if a >= 0 and c >= 0:
            return Iv(a * c, b * d)
        p = (a * c, a * d, b * c, b * d)
        return Iv(min(p), max(p))

    __rmul__ = __mul__
    __radd__ = __add__

    def inv(self):
        if not (self.lo > 0 or self.hi < 0):
            raise ZeroDivisionError("interval straddles zero")
        return Iv(Fraction(1) / self.hi, Fraction(1) / self.lo)

    def __truediv__(self, o):
        return self * iv(o).inv()

    def width(self):
        return self.hi - self.lo

    def mid(self):
        return (self.lo + self.hi) / 2

    def __repr__(self):
        return "[%s, %s]" % (dec(self.lo, 6), dec(self.hi, 6))


def iv(x):
    return x if isinstance(x, Iv) else Iv(x)


IV0 = Iv(0)
IV1 = Iv(1)


def rounds_to(n, d, r, s):
```

```
    """True iff n/d rounds to r/s.  Half-way cases are rejected."""
    assert d > 0 and s > 0
    return 2 * abs(n * s - r * d) < d


def sci_rounds(n, d, mant, scale, e):
    """True iff n/d rounds to (mant/scale) x 10^e.  The power of ten
    multiplies the VALUE and never the target; checks D1-05 and D1-06
    foreclose the other form."""
    if e >= 0:
        return rounds_to(n, d * 10 ** e, mant, scale)
    return rounds_to(n * 10 ** (-e), d, mant, scale)


def digits_is(n, k):
    assert n > 0 and k >= 1
    return 10 ** (k - 1) <= n < 10 ** k


def log2_rounds(n, d, r, s):
    """True iff log2(n/d) rounds to r/s, for n/d >= 1.  Article 84's
    device, reused here to re-derive that paper's hexamer bracket."""
    assert n > 0 and d > 0 and s > 0 and r >= 0
    lo = 2 ** (2 * r - 1) * d ** (2 * s)
    hi = 2 ** (2 * r + 1) * d ** (2 * s)
    return lo <= n ** (2 * s) < hi


def root_rounds(a, b, k, r, s):
    """True iff 1 - (a/b)^(1/k) rounds to r/s, for 0 < a/b <= 1.

    THE DEVICE OF THIS HARNESS.  The root is never formed.  Writing
    y = 1 - (a/b)^(1/k), the statement that y rounds to r/s is
        (2r-1)/(2s) < y < (2r+1)/(2s),
    hence
        1 - (2r+1)/(2s) < (a/b)^(1/k) < 1 - (2r-1)/(2s),
    and x -> x^k is increasing on the positives, so raising the bracket
    to the k clears the root and leaves a comparison of two exact
    integers:
        (2s - 2r - 1)^k * b < a * (2s)^k  and  a * (2s)^k < (2s - 2r + 1)^k *
           b.
    The integers are large -- certifying the anchor budget at k = 1000
    compares two of about 7,300 digits -- and Python's integers are
    arbitrary precision, so the comparison is exact.
    """
    assert 0 < a <= b and k >= 1 and s > 0 and r >= 0
    assert 2 * s > 2 * r + 1
    lo = (2 * s - 2 * r - 1) ** k
    hi = (2 * s - 2 * r + 1) ** k
    tot = (2 * s) ** k
    return lo * b < a * tot and a * tot < hi * b


def sci_root_rounds(a, b, k, mant, scale, e):
    """The same, with the target in scientific notation."""
    assert e <= 0
    return root_rounds(a, b, k, mant, scale * 10 ** (-e))
```

```
def rounds_to_iv(v, r, s):
    """True iff the WHOLE enclosure rounds to r/s.  Fails -- rather than
    guessing -- if the enclosure is too wide to decide the rounding."""
    v = iv(v)
    return (Fraction(2 * r - 1, 2 * s) < v.lo
            and v.hi < Fraction(2 * r + 1, 2 * s))


def sci_rounds_iv(v, mant, scale, e):
    v = iv(v)
    w = v * Iv(Fraction(1, 10 ** e)) if e >= 0 else v * Iv(10 ** (-e))
    return rounds_to_iv(w, mant, scale)


# --- ln 2 and the logarithm, by series with explicit remainder --------
def _atanh(z, terms):
    """atanh(z) for a rational 0 <= z <= 1/3.  The tail beyond the given
    number of terms is at most z^(2t+1)/((2t+1)(1-z^2)).  Each step
    rounds outward onto the grid, so no denominator grows with the term
    index."""
    zi = iv(z)
    z2 = zi * zi
    assert z2.hi < 1
    zp = zi
    s = IV0
    for k in range(terms):
        s = s + zp * Iv(Fraction(1, 2 * k + 1))
        zp = zp * z2
    tail = zp.hi / ((2 * terms + 1) * (1 - z2.hi))
    return Iv(s.lo, s.hi + tail)


LN2 = Iv(2) * _atanh(Fraction(1, 3), 130)
_INVLN2 = LN2.inv()
LN10 = None          # set below, once ln is available
_LNCACHE = {}


def ln_iv(x):
    """ln(x) for a positive rational x.  Powers of two come out first, so
    the series is only asked for an argument in [1, 2)."""
    x = Fraction(x)
    assert x > 0
    if x in _LNCACHE:
        return _LNCACHE[x]
    k, y = 0, x
    while y >= 2:
        y /= 2
        k += 1
    while y < 1:
        y *= 2
        k -= 1
    r = Iv(k) * LN2 + Iv(2) * _atanh((y - 1) / (y + 1), 120)
    _LNCACHE[x] = r
    return r


LN10 = ln_iv(10)
```

```
def log10_iv(v):
    """log10 of an interval of positive rationals."""
    v = iv(v)
    assert v.lo > 0
    return Iv(ln_iv(v.lo).lo, ln_iv(v.hi).hi) / LN10


def root_iv(a, b, k, bits=120):
    """A certified rational enclosure of (a/b)^(1/k), by bisection on a
    dyadic grid with the EXACT integer predicate x^k * b <= a.  Needed
    only where a root enters a further quotient, as in D8."""
    assert 0 < a <= b and k >= 1
    lo, hi = Fraction(0), Fraction(1)
    for _ in range(bits):
        mid = (lo + hi) / 2
        if mid.numerator ** k * b <= mid.denominator ** k * a:
            lo = mid
        else:
            hi = mid
    return Iv(lo, hi)


def block_D1():
    block("D1", "the devices, and their non-vacuity")

    check("the rounding bound accepts a correct rounding",
          rounds_to(1, 3, 33, 100), True)
    check("and rejects the next value down", rounds_to(1, 3, 32, 100), False)
    check("and rejects the next value up", rounds_to(1, 3, 34, 100), False)
    check("and rejects a half-way case", rounds_to(1, 2, 1, 1), False)
    check("the scientific bracket puts the power into the value",
          sci_rounds(12273, 1000, 12, 10, 1), True)
    check("and the other form rejects the same correct value",
          rounds_to(12273, 1000, 120, 10), False)
    check("the digit count accepts four digits", digits_is(1000, 4), True)

    check("the binary-logarithm bracket accepts log2(8) = 3.0",
          log2_rounds(8, 1, 30, 10), True)
    check("and rejects 2.9 and 3.1",
          (log2_rounds(8, 1, 29, 10), log2_rounds(8, 1, 31, 10)),
          (False, False))

    # The root bracket, on a case whose answer is exact.
    check("the root bracket accepts 1 - (1/4)^(1/2) = 1/2 exactly",
          root_rounds(1, 4, 2, 5000, 10000), True)
    check("and rejects 0.4999 and 0.5001",
          (root_rounds(1, 4, 2, 4999, 10000),
           root_rounds(1, 4, 2, 5001, 10000)), (False, False))
    check("it accepts a root that is not exact: 1 - (1/2)^(1/3) = 0.2063",
          root_rounds(1, 2, 3, 2063, 10000), True)
    check("and rejects the neighbours of that",
          (root_rounds(1, 2, 3, 2062, 10000),
           root_rounds(1, 2, 3, 2064, 10000)), (False, False))
    check("1 - (a/b)^(1/k) falls as k grows, which the bracket agrees"
          " with", all(root_rounds(99, 100, k, r, s) for k, r, s
                       in ((45, 223, 10 ** 6), (1000, 101, 10 ** 7))), True)

    # The enclosures, validated rather than trusted.
```

```
    check("ln 2 lies strictly inside the bracket of its first fifteen"
          " places",
          (Fraction(693147180559945, 10 ** 15) < LN2.lo,
           LN2.hi < Fraction(693147180559946, 10 ** 15)), (True, True))
    check("and its enclosure is narrower than 10^-80",
          LN2.width() < Fraction(1, 10 ** 80), True)
    check("ln 10 lies strictly inside the bracket of its first fifteen",
          (Fraction(2302585092994045, 10 ** 15) < LN10.lo,
           LN10.hi < Fraction(2302585092994046, 10 ** 15)), (True, True))
    check("ln is additive on the arguments tested: ln 4 = 2 ln 2",
          ((Iv(2) * LN2).lo <= ln_iv(4).hi
           and ln_iv(4).lo <= (Iv(2) * LN2).hi), True)
    check("log10 encloses its exact values at three powers of ten",
          all(log10_iv(Iv(10 ** k)).lo <= k <= log10_iv(Iv(10 ** k)).hi
              for k in (1, 3, 6)), True)
    check("the root enclosure brackets an exact root: (1/4)^(1/2) = 1/2",
          (root_iv(1, 4, 2).lo <= Fraction(1, 2) <= root_iv(1, 4, 2).hi),
          True)
    check("and agrees with the root bracket on the anchor budget",
          rounds_to_iv(IV1 - root_iv(99, 100, 1000), 101, 10 ** 7), True)
    check("an enclosure too wide to decide a rounding is rejected",
          rounds_to_iv(Iv(Fraction(100, 10 ** 7), Fraction(102, 10 ** 7)),
                       101, 10 ** 7), False)
    show("ln 2  = %s  (width %s)" % (dec(LN2.mid(), 20), expo(LN2.width())))
    note("every figure below is carried by one of these five devices")


# ======================================================================
#  D2.  The imported objects, derived
# ======================================================================
#
#  Article 86's inputs are other papers' outputs.  Its own generator
#  carries them as literals; this block recomputes the two that can be
#  recomputed, so that the bounds of D3 to D11 rest on derivations and
#  not on transcriptions.

LOC = (3, 6, 9, 12)
IOTA = {3: 9, 9: 3, 6: 12, 12: 6}
D_ = frozenset({6, 12})
I_ = frozenset({3, 6, 12})
T_ = frozenset({3, 6, 9, 12})
FAMS = (("D", D_), ("I", I_), ("T", T_))
ARMS2 = ("h", "e")

RHO = 200
X34 = Fraction(1, 10)


def admitted(rho, x):
    tot = Fraction(rho) + Fraction(x) + 1
    return (Fraction(rho) / tot, Fraction(x) / tot, Fraction(1) / tot)


FD, FI, FT = admitted(RHO, X34)
FRAC = {"D": FD, "I": FI, "T": FT}


def arm_words(S, arms):
    ks = sorted(S)
```

```
    return [dict(zip(ks, v)) for v in product(arms, repeat=len(ks))]


def pres(S, a):
    return frozenset((k, a[k]) for k in S)


def pres_flip(S, a):
    return frozenset((IOTA[k], a[k]) for k in S)


def nu(S1, a1, S2, a2, aligned):
    P2 = pres(S2, a2) if aligned else pres_flip(S2, a2)
    return len(pres(S1, a1) & P2)


def sensitive(S1, a1, S2, a2):
    return nu(S1, a1, S2, a2, True) != nu(S1, a1, S2, a2, False)


class Poly:
    __slots__ = ("c",)

    def __init__(self, c=()):
        c = [Fraction(x) for x in c]
        while c and c[-1] == 0:
            c.pop()
        self.c = c

    def __add__(self, o):
        n = max(len(self.c), len(o.c))
        return Poly([(self.c[i] if i < len(self.c) else 0)
                     + (o.c[i] if i < len(o.c) else 0) for i in range(n)])

    def __mul__(self, o):
        if not self.c or not o.c:
            return Poly()
        r = [Fraction(0)] * (len(self.c) + len(o.c) - 1)
        for i, a in enumerate(self.c):
            if a == 0:
                continue
            for j, b in enumerate(o.c):
                r[i + j] += a * b
        return Poly(r)

    def scale(self, k):
        return Poly([x * Fraction(k) for x in self.c])

    def __call__(self, x):
        x = Fraction(x)
        s = Fraction(0)
        for a in reversed(self.c):
            s = s * x + a
        return s

    def degree(self):
        return len(self.c) - 1
```

```
PONE, PH, PE = Poly([1]), Poly([0, 1]), Poly([1, -1])


def word_poly(S, a):
    p = PONE
    for k in sorted(S):
        p = p * (PH if a[k] == "h" else PE)
    return p


def sensitivity_poly():
    """Article 85's sigma(pi_h), recomputed from the definitions."""
    tot = Poly()
    for n1, S1 in FAMS:
        for n2, S2 in FAMS:
            w = FRAC[n1] * FRAC[n2]
            for a1 in arm_words(S1, ARMS2):
                for a2 in arm_words(S2, ARMS2):
                    if sensitive(S1, a1, S2, a2):
                        tot = tot + (word_poly(S1, a1)
                                     * word_poly(S2, a2)).scale(w)
    return tot


OPS = (("Pure", Fraction(0)), ("NineNine", Fraction(1, 100)),
       ("NineFive", Fraction(1, 20)), ("Nine", Fraction(1, 10)))


def seq_prob(rho):
    """Article 84's six-site alternating column D,T,D,T,D,T."""
    fD, fI, fT = admitted(rho, Fraction(1, 10))
    return (fD ** 3) * (fT ** 3)


def block_D2():
    block("D2", "the imported objects, derived rather than transcribed")

    sig = sensitivity_poly()
    check("Article 85's sigma is recovered: degree eight, rational"
          " coefficients", (sig.degree(), isinstance(sig.c[0], Fraction)),
          (8, True))
    check("and sigma(0) is exactly f_I^2 = 1/2011^2",
          sig(0), Fraction(1, 2011 ** 2))
    check("which is 1/4044121", 2011 ** 2, 4044121)

    sv = {}
    for nm, ph in OPS:
        sv[nm] = sig(ph)
        check("sigma at %s is exactly rational" % nm,
              isinstance(sv[nm], Fraction), True)
    for nm, m, e in (("Pure", 247, -7), ("NineNine", 392, -4),
                     ("NineFive", 903, -3), ("Nine", 324, -2)):
        check("sigma at %s rounds to the published %d.%02de%d"
              % (nm, m // 100, m % 100, e),
              sci_rounds(sv[nm].numerator, sv[nm].denominator, m, 100, e),
              True)
    note("the submitted draft imported these as five-significant-figure"
         " literals; the figures above are the exact values, and the"
         " article's generator now computes them the same way")
```

```
    # Article 84's coded hexamer, re-derived by the binary-logarithm
    # bracket from the same admitted fractions.
    p100, p200 = seq_prob(100), seq_prob(200)
    check("the species probability at rho* = 100 is 10^12/1011^6",
          p100, Fraction(10 ** 12, 1011 ** 6))
    check("and at rho* = 200 it is 8 x 10^12/2011^6",
          p200, Fraction(8 * 10 ** 12, 2011 ** 6))
    check("the total cost of a coded hexamer at rho* = 100 rounds to 25.0"
          " bits", log2_rounds(p100.denominator * 2 ** 5, p100.numerator,
                               250, 10), True)
    check("and at rho* = 200 to 28.0 bits",
          log2_rounds(p200.denominator * 2 ** 5, p200.numerator, 280, 10),
          True)
    check("and neither to the neighbouring tenth",
          (log2_rounds(p100.denominator * 2 ** 5, p100.numerator, 249, 10),
           log2_rounds(p200.denominator * 2 ** 5, p200.numerator, 281, 10)),
          (False, False))
    check("the registry term is five bits, which needs no logarithm",
          2 ** 5, 32)
    note("so Article 86's 25.0 and 28.0 are derived here, not imported,"
         " and the article's division of them is exact given the figures")
    show("sigma: %s" % ", ".join("%s %s" % (nm, expo(sv[nm]))
                                 for nm, _ in OPS))
    return sig, sv


# ======================================================================
#  D3.  The demand, its allocation, and the genome length
# ======================================================================
#
#  Article 65, Thm. 8.5: the per-period demand of a protocell of genome
#  length L is 5L + alpha/(1-alpha) N + nu L units, with N = 2.52e5 the
#  particle count and nu = 66 the lipid pool.

L200, LCRIT = 200, 222
NU, NPART = 66, 252000
A_TENTH, A_FIFTIETH = Fraction(1, 10), Fraction(1, 50)


def demand(L, alpha):
    a = Fraction(alpha)
    return 5 * L + a / (1 - a) * NPART + NU * L


def block_D3():
    block("D3", "the demand, its allocation, and the genome length")

    r10 = demand(L200, A_TENTH)
    r02 = demand(L200, A_FIFTIETH)
    check("the demand at L = 200, alpha = 1/10 is exactly 42,200 units"
          " per period", r10, Fraction(42200))
    check("and at alpha = 1/50 it is exactly 135400/7, which rounds to"
          " 19,343",
          (r02, rounds_to(r02.numerator, r02.denominator, 19343, 1)),
          (Fraction(135400, 7), True))
    note("Article 65 reports the second rounded to 19,300; the harness"
         " records the unrounded figure its own formula gives")
```

```
    anchor = 5 * L200
    residue = A_TENTH / (1 - A_TENTH) * NPART
    lipid = NU * L200
    check("the three terms are 1,000 anchor, 28,000 residue, 13,200 lipid",
          (anchor, residue, lipid), (1000, Fraction(28000), 13200))
    check("and they sum to the demand exactly", anchor + residue + lipid,
          r10)
    for nm, v, m in (("anchor", anchor, 24), ("residue", residue, 664),
                     ("lipid", lipid, 313)):
        sh = 100 * Fraction(v) / r10
        check("the %s share rounds to %s per cent"
              % (nm, dec(Fraction(m, 10), 1)),
              rounds_to(sh.numerator, sh.denominator, m, 10), True)
    nonstruct = 100 * (residue + lipid) / r10
    check("the non-structural share is exactly 41200/422 per cent and"
          " rounds to 97.6",
          (nonstruct, rounds_to(nonstruct.numerator, nonstruct.denominator,
                                976, 10)), (Fraction(41200, 422), True))
    check("so over ninety-seven per cent of the per-period demand is not"
          " placed into the heritable structure at all",
          nonstruct > 97, True)

    rcrit = demand(LCRIT, A_TENTH)
    check("Article 80 rules out L = 200; at L = 222 the demand is exactly"
          " 43,762", rcrit, Fraction(43762))
    check("the anchor term there is 1,110 units", 5 * LCRIT, 1110)
    rise = 100 * (rcrit / r10 - 1)
    check("and the demand rises by 3.7 per cent",
          rounds_to(rise.numerator, rise.denominator, 37, 10), True)
    check("which is exactly 78100/21100 per cent", rise,
          Fraction(78100, 21100))
    show("anchor %s, residue %s, lipid %s; at L = %d the demand is %s"
         % (grp(anchor), grp(residue), grp(lipid), LCRIT, grp(rcrit)))
    return r10, rcrit, anchor


# ======================================================================
#  D4.  Throughput
# ======================================================================
#
#  Article 78, Rem. 7.2, attributing the figure to Article 64: the
#  protocell period is twelve hours.

PERIOD_H = 12


def block_D4(r10, rcrit):
    block("D4", "throughput")

    T = PERIOD_H * 3600
    check("the period is twelve hours, which is 43,200 seconds", T, 43200)
    for tag, R, r, s in (("nominal", r10, 977, 1000),
                         ("critical", rcrit, 1013, 1000)):
        rate = R / T
        check("the %s placement rate rounds to %s units per second"
              % (tag, dec(Fraction(r, s), 3)),
              rounds_to(rate.numerator, rate.denominator, r, s), True)
    for tag, R, r, s in (("nominal", r10, 1024, 1000),
                         ("critical", rcrit, 987, 1000)):
```

```
        gap = Fraction(T) / R
        check("and the %s interval between placements rounds to %s seconds"
              % (tag, dec(Fraction(r, s), 3)),
              rounds_to(gap.numerator, gap.denominator, r, s), True)
    for hours, r in ((6, 1954), (24, 488)):
        rate = r10 / (hours * 3600)
        check("at a %d-hour period the rate rounds to %s"
              % (hours, dec(Fraction(r, 1000), 3)),
              rounds_to(rate.numerator, rate.denominator, r, 1000), True)

    # "Of order one per second" must survive the alternative periods the
    # same block computes, so it is tested as an order of magnitude and
    # not as a rounded equality.
    worst = []
    for R, P in ((r10, PERIOD_H), (r10, 6), (r10, 24), (rcrit, PERIOD_H)):
        worst.append(R / (P * 3600))
    check("the rate is within an order of magnitude of one unit per"
          " second at every period and genome length tested",
          all(Fraction(1, 10) < x < 10 for x in worst), True)
    check("and the spread over those four cases is exactly four-fold",
          max(worst) / min(worst), Fraction(4))
    show("%s units per second at L = %d; %s and %s at 6 h and 24 h"
         % (dec(r10 / T, 3), L200, dec(r10 / 21600, 3),
            dec(r10 / 86400, 3)))
    return T


# ======================================================================
#  D5.  The information bound
# ======================================================================
#
#  The corpus already prices a coded hexamer: Article 84, Prop. 9.5 and
#  Rem. 9.6, 25.0 to 28.0 bits for six sites of which the registry term
#  is five.  This paper does not improve on that and does not invent a
#  species term of its own; it carries that figure and divides.

BOX = (3, 3, 6)


def block_D5(anchor, T):
    block("D5", "the information bound, at Article 84's own figures")

    lo, hi, reg, n = (Fraction(25), Fraction(28), Fraction(5), 6)
    for nm, v, m in (("lower", lo / n, 417), ("upper", hi / n, 467)):
        check("the %s figure is %s bits per unit" % (nm, dec(v, 2)),
              rounds_to(v.numerator, v.denominator, m, 100), True)
    check("the registry term per unit is 5/6 and rounds to 0.833",
          (reg / n, rounds_to(5, 6, 833, 1000)), (Fraction(5, 6), True))

    nx, ny, nz = BOX
    nodes, contacts = nx * ny * nz, nx * ny * (nz - 1)
    check("Article 80's heritable minimum is a 3x3x6 box of 54 nodes"
          " carrying 45 stacking contacts", (nodes, contacts), (54, 45))
    check("and not a column of 54 with 53", 54 - 1 == contacts, False)
    check("the box is nine columns of six", (nx * ny, nz), (9, 6))
    check("the registry term per unit IS the contacts-per-node ratio of"
          " the box, exactly and not approximately",
          Fraction(contacts, nodes), reg / n)
    note("both are (ell-1)/ell at ell = 6, which is why they coincide")
```

```
    for nm, v, want in (("lower", lo / n, 4167), ("upper", hi / n, 4667)):
        tot = v * anchor
        check("over the %d anchor units the %s figure is %s bits per"
              " period" % (anchor, nm, grp(want)),
              rounds_to(tot.numerator, tot.denominator, want, 1), True)
    for nm, v, m in (("lower", lo / n, 96), ("upper", hi / n, 108)):
        rate = v * anchor / T
        check("which is a %s bit rate of %s bits per second"
              % (nm, dec(Fraction(m, 1000), 3)),
              rounds_to(rate.numerator, rate.denominator, m, 1000), True)
    show("%s to %s bits per unit; over %d anchor units, %s to %s bits per"
         " period" % (dec(lo / n, 2), dec(hi / n, 2), anchor,
                      grp(dec(lo / n * anchor, 0)),
                      grp(dec(hi / n * anchor, 0))))
    return lo / n, hi / n


# ======================================================================
#  D6.  The thermodynamic floor
# ======================================================================
#
#  Landauer: erasing one bit costs kT ln 2.  This is the one place in the
#  article where a transcendental constant enters the arithmetic rather
#  than the bound, and it is carried by the enclosure of D1.

def block_D6(lo, hi):
    block("D6", "the thermodynamic floor")

    check("one bit costs ln 2 kT, which rounds to 0.6931",
          rounds_to_iv(LN2, 6931, 10000), True)
    check("and not to 0.6930 or 0.6932",
          (rounds_to_iv(LN2, 6930, 10000), rounds_to_iv(LN2, 6932, 10000)),
          (False, False))

    sel_lo = Iv(lo) * LN2
    sel_hi = Iv(hi) * LN2
    check("the selection cost per unit is 2.89 kT at the lower figure",
          rounds_to_iv(sel_lo, 289, 100), True)
    check("and 3.23 kT at the upper", rounds_to_iv(sel_hi, 323, 100), True)

    # Against the contact the architecture actually makes.  H_rgy prices a
    # contact by its registration number; a disubstituted contact carries
    # nu = 2 and a tetrasubstituted one nu = 4, and the stream is
    # overwhelmingly disubstituted, so nu = 2 is the honest comparator.
    check("a disubstituted contact carries two registrations and a"
          " tetrasubstituted one four", (len(D_), len(T_)), (2, 4))
    check("the selection cost EXCEEDS one disubstituted contact at"
          " Delta = 1", sel_hi.lo > 2, True)
    check("and falls below one tetrasubstituted contact", sel_hi.hi < 4,
          True)
    check("the ratio to a disubstituted contact rounds to 1.62",
          rounds_to_iv(sel_hi / Iv(2), 162, 100), True)
    check("and to a tetrasubstituted one, 0.81",
          rounds_to_iv(sel_hi / Iv(4), 81, 100), True)
    check("so energy is COMPARABLE and not slack: the cost sits strictly"
          " between the two contacts the architecture makes",
          (2 < sel_hi.lo, sel_hi.hi < 4), (True, True))
    show("%s to %s kT per unit placed; %s of a disubstituted contact"
```

```
          % (dec(sel_lo.mid(), 2), dec(sel_hi.mid(), 2),
             dec((sel_hi / Iv(2)).mid(), 2)))
    return sel_lo, sel_hi


# ======================================================================
#  D7.  The error budget
# ======================================================================
#
#  This block makes no thermodynamic claim.  It bounds a per-placement
#  error rate and nothing else: if n placements must all succeed and the
#  structure is to survive with probability F, then under independence
#  the per-placement error rate satisfies eps <= 1 - F^(1/n).
#
#  Every figure here is an n-th root of a rational, and every one is
#  pinned by the integer root bracket of D1.  Nothing is approximated.

def block_D7(anchor):
    block("D7", "the error budget, by the integer root bracket")

    nx, ny, nz = BOX
    n_box = nx * ny * (nz - 1)
    n_anchor = int(anchor)
    check("the box has 45 placements and a period's anchor term 1,000",
          (n_box, n_anchor), (45, 1000))

    cases = (("box", n_box, 99, 100, 223, 100, -4),
             ("box", n_box, 1, 2, 153, 100, -2),
             ("anchor", n_anchor, 99, 100, 101, 100, -5),
             ("anchor", n_anchor, 1, 2, 693, 100, -4))
    for nm, k, a, b, m, sc, e in cases:
        check("at F = %d/%d over the %s's %d placements the budget is"
              " %d.%02de%d" % (a, b, nm, k, m // 100, m % 100, e),
              sci_root_rounds(a, b, k, m, sc, e), True)
        check("and it is not the neighbouring mantissa",
              (sci_root_rounds(a, b, k, m - 1, sc, e),
               sci_root_rounds(a, b, k, m + 1, sc, e)), (False, False))

    # The one structural property claimed: the budget tightens with the
    # number of placements.  Cross-multiplied, so no root is formed.
    #   1 - 0.99^(1/45) > 1 - 0.99^(1/1000)  <=>  0.99^(1/45) < 0.99^(1/1000)
    #   <=> 0.99^1000 < 0.99^45 ... as powers of a number below one.
    check("the budget tightens with the number of placements, by the"
          " monotonicity of x^k on (0,1)",
          99 ** 1000 * 100 ** 45 < 99 ** 45 * 100 ** 1000, True)
    check("so the box's budget exceeds the anchor term's at F = 0.99",
          root_iv(99, 100, n_box).lo < root_iv(99, 100, n_anchor).lo, True)
    budget = IV1 - root_iv(99, 100, n_anchor)
    check("and the anchor budget encloses 1.01e-5",
          sci_rounds_iv(budget, 101, 100, -5), True)
    note("the bracket is the certificate; the enclosure is formed only"
         " because D8 divides by this quantity")
    show("box, F = 0.99: eps <= %s ; anchor, F = 0.99: eps <= %s"
         % (expo((IV1 - root_iv(99, 100, n_box)).mid()),
            expo(budget.mid())))
    return budget


# ======================================================================
```

```
# D8.  The orientation floor
# ======================================================================
#
# THE CENTRAL BLOCK.  Article 84, Cor. 7.5: on the stock Article 79
# delivers, the registration weight of a contact is independent of its
# alignment bit at every contact except I-I.  Article 85, Thm. 8.2: the
# expected signed bias of the stream is exactly zero absent the
# initiator, the law of nu_0 - nu_1 being symmetric under a
# measure-preserving involution, and Prop. 7.3 records that the favoured
# state need not be the aligned one.
#
# So the mean orientation error rate cannot fall below one half, however
# large the registration free energy -- and the argument needs no value
# of that free energy, which is why no exponential is formed below.

def block_D8(sv, budget):
    block("D8", "the orientation floor: what equilibrium cannot do")

    # (a) A blind contact.  The two alignments carry equal weight, so the
    # induced law is Bernoulli(1/2) whatever Delta is.  What is checked is
    # the equality of the EXPONENTS, so no exponential is formed and the
    # conclusion holds at every Delta rather than at the ones tested.
    delivered = {"D": {6: "e", 12: "e"},
                 "I": {3: "e", 6: "e", 12: "e"},
                 "T": {3: "e", 6: "e", 9: "e", 12: "e"}}
    fam = dict(FAMS)
    blind_pairs = []
    for a in ("D", "T"):
        for b in ("D", "T"):
            n0 = nu(fam[a], delivered[a], fam[b], delivered[b], True)
            n1 = nu(fam[a], delivered[a], fam[b], delivered[b], False)
            blind_pairs.append(n0 == n1)
    check("on the delivered stock every D or T contact carries equal"
          " registration number in both alignments", all(blind_pairs),
          True)
    check("so its error rate is exactly one half at EVERY Delta, no"
          " exponential having been formed", Fraction(1, 2),
          Fraction(1, 2))

    # (b) The sign.  At a sensitive contact the favoured state is the
    # correct one with probability one half, so the contact's error rate
    # is (1/2)theta + (1/2)(1 - theta) = 1/2 IDENTICALLY -- an algebraic
    # identity in theta, checked here over a grid of rationals because it
    # holds for every value theta could take.
    ids = []
    for num in range(1, 1000):
        th = Fraction(num, 1000)
        ids.append(Fraction(1, 2) * th + Fraction(1, 2) * (1 - th)
                   == Fraction(1, 2))
    check("with the sign symmetric a sensitive contact's error rate is"
          " exactly one half, identically in theta", all(ids), True)
    check("and the identity needs no value of Delta, of the gap, or of"
          " sigma", len(ids), 999)

    # (c) The sign-agnostic floor, and the exact floor with the
    # initiator.  Both are rational once sigma is.
    for nm, want in (("Pure", 500000), ("NineNine", 499804),
                     ("NineFive", 495487), ("Nine", 483799)):
        fl = (1 - sv[nm]) / 2
```

```
        check("the sign-agnostic floor at %s rounds to %s"
              % (nm, dec(Fraction(want, 10 ** 6), 6)),
              rounds_to(fl.numerator, fl.denominator, want, 10 ** 6), True)
    weak = (1 - sv["Nine"]) / 2
    check("and to 0.4838 at four places",
          rounds_to(weak.numerator, weak.denominator, 4838, 10000), True)

    # (c') The initiator, the one exception and the one place the sign is
    # fixed: an I-I contact reads three aligned against two unaligned, so
    # there the ALIGNED state is favoured.  There are three classes of
    # contact, not two, and the floor is a mixture over all three.
    n0 = nu(I_, delivered["I"], I_, delivered["I"], True)
    n1 = nu(I_, delivered["I"], I_, delivered["I"], False)
    check("an I-I contact of the delivered stock reads three aligned"
          " against two unaligned", (n0, n1), (3, 2))
    check("so the initiator is the only contact whose preference points"
          " the right way", n0 > n1, True)
    fI2 = FI * FI
    check("its frequency is exactly f_I^2 = 1/4044121", fI2,
          Fraction(1, 4044121))
    check("and rounds to 2.47e-7",
          sci_rounds(1, 4044121, 247, 100, -7), True)

    floor = (1 - fI2) / 2
    check("the floor is exactly 1/2 - f_I^2/2", floor,
          Fraction(1, 2) - fI2 / 2)
    check("which is 4044120/8088242 and rounds to 0.499999876",
          (floor, rounds_to(floor.numerator, floor.denominator,
                            499999876, 10 ** 9)),
          (Fraction(4044120, 8088242), True))
    short = Fraction(1, 2) - floor
    check("it falls short of one half by exactly f_I^2/2, which rounds to"
          " 1.24e-7",
          (short, sci_rounds(short.numerator, short.denominator, 124, 100,
                             -7)), (fI2 / 2, True))
    check("the floor is below one half, and the initiator is why",
          floor < Fraction(1, 2), True)
    check("absent the initiator it would be exactly one half",
          Fraction(1, 2), Fraction(1, 2))

    # (d) The gap between the floor and the budget, and its uniformity.
    ratio = Iv(floor) / budget
    check("the floor exceeds the error budget", ratio.lo > 1, True)
    check("by a factor that rounds to 4.97e4",
          sci_rounds_iv(ratio, 497, 100, 4), True)
    orders = log10_iv(ratio)
    check("which is 4.7 orders of magnitude",
          rounds_to_iv(orders, 47, 10), True)
    check("and the gap is uniform in Delta and in sigma, neither"
          " appearing in the floor", floor == (1 - fI2) / 2, True)

    # (e) And it survives the opposite extreme of the independence
    # hypothesis: perfectly correlated failures permit only eps <= 1 - F.
    corr = Fraction(1, 100)
    check("under perfectly correlated failures the budget is 1 - F = 0.01",
          corr, Fraction(1, 100))
    check("and the floor still exceeds it fifty-fold",
          rounds_to((floor / corr).numerator, (floor / corr).denominator,
                    50, 1), True)
```

```
    check("so the comparison does not rest on the independence"
          " hypothesis", floor > corr, True)
    show("floor %s against budget %s: a factor of %s, which is %s orders"
         % (dec(floor, 9), expo(budget.mid()), expo(ratio.mid()),
            dec(orders.mid(), 1)))
    return floor, ratio


# ======================================================================
#  D9.  The sensing bound
# ======================================================================
#
#  The bound is 1/sigma contacts per bit and is free of Delta.  Article
#  85's realised capacities at Delta = 1 are an EVALUATION, not a bound:
#  realised capacity rises with Delta toward the ceiling, so the
#  requirement falls from the Delta = 1 figure toward 1/sigma.  Both are
#  reported and the article must not quote the evaluation as the bound.
#
#  THIS IS THE BLOCK WHERE DERIVING RATHER THAN IMPORTING CHANGED THE
#  ANSWER.  Three figures of the submitted draft were reciprocals of
#  rounded literals and differed from the reciprocals of the quantities
#  those literals approximate.  Both the superseded and the exact values
#  are carried here: the article records the correction rather than
#  absorbing it, and so does this harness.

def hbin_needed():
    """The realised capacity at Delta = 1 is Article 85's quantity and is
    transcendental; this harness needs it only to the three significant
    figures the article prints, and takes it from the enclosure below.
    An earlier form of the article's generator imported it as a
    three-significant-figure literal instead."""
    return None


def _atanh_log2(x):
    return ln_iv(x) * _INVLN2


def exp_int(n, terms=70):
    """e^n for a non-negative integer n, from the factorial series."""
    s, t = IV0, IV1
    for k in range(1, terms + 1):
        s = s + t
        t = t / Iv(k)
    e = Iv(s.lo, s.hi + t.hi * 2)
    r = IV1
    for _ in range(n):
        r = r * e
    return r


def theta_iv(g, delta):
    return (IV1 + exp_int(g * delta)).inv()


def hbin_iv(p):
    p = iv(p)
    assert p.lo > 0 and p.hi < 1
    q = IV1 - p
```

```
    lp = Iv(_atanh_log2(p.lo).lo, _atanh_log2(p.hi).hi)
    lq = Iv(_atanh_log2(q.lo).lo, _atanh_log2(q.hi).hi)
    return -(p * lp + q * lq)


def gap_dist(ph):
    out = {}
    for n1, S1 in FAMS:
        for n2, S2 in FAMS:
            w = FRAC[n1] * FRAC[n2]
            for a1 in arm_words(S1, ARMS2):
                for a2 in arm_words(S2, ARMS2):
                    g = abs(nu(S1, a1, S2, a2, True)
                            - nu(S1, a1, S2, a2, False))
                    if g == 0:
                        continue
                    p = Fraction(1)
                    for k in sorted(S1):
                        p *= ph if a1[k] == "h" else (1 - ph)
                    for k in sorted(S2):
                        p *= ph if a2[k] == "h" else (1 - ph)
                    p *= w
                    if p:
                        out[g] = out.get(g, Fraction(0)) + p
    return out


def realised(ph, delta=1):
    """Article 85's realised capacity per contact at the given Delta,
    recomputed here as a certified enclosure."""
    tot = IV0
    for g, p in gap_dist(ph).items():
        tot = tot + Iv(p) * (IV1 - hbin_iv(theta_iv(g, delta)))
    return tot


def block_D9(sv):
    block("D9", "the sensing bound, and three figures it corrects")

    # The Delta-free bound, exactly.
    check("the bound at the pure limit is EXACTLY 2011^2 = 4,044,121"
          " contacts per bit, sigma(0) being exactly 1/2011^2",
          Fraction(1) / sv["Pure"], Fraction(4044121))
    check("and it is an integer, which is what made the draft's"
          " 4,044,162 a transcription of a rounded reciprocal",
          (Fraction(1) / sv["Pure"]).denominator, 1)
    for nm, want in (("NineNine", 2548), ("NineFive", 111), ("Nine", 31)):
        inv = Fraction(1) / sv[nm]
        check("the bound at %s rounds to %s contacts per bit"
              % (nm, grp(want)),
              rounds_to(inv.numerator, inv.denominator, want, 1), True)

    # The Delta = 1 evaluation, recomputed rather than imported.
    rl = {}
    for nm, ph in OPS[1:]:
        rl[nm] = realised(ph)
        check("the realised capacity at %s is strictly below its ceiling"
              % nm, rl[nm].hi < sv[nm], True)
    for nm, m, e in (("NineNine", 185, -4), ("NineFive", 427, -3),
```

```
                        ("Nine", 153, -2)):
        check("the realised capacity at %s rounds to %d.%02de%d"
              % (nm, m // 100, m % 100, e),
              sci_rounds_iv(rl[nm], m, 100, e), True)
    check("at pi_e = 0.95 that is 4.27e-3 and NOT the 4.30e-3 the draft"
          " carried, which was Article 85's two-place decimal 0.0043"
          " re-expressed with a third significant figure",
          sci_rounds_iv(rl["NineFive"], 430, 100, -3), False)

    for nm, want in (("NineNine", 5392), ("NineFive", 234), ("Nine", 65)):
        req = rl[nm].inv()
        check("the Delta = 1 requirement at %s is %s contacts per bit"
              % (nm, grp(want)), rounds_to_iv(req, want, 1), True)
    check("the draft's 5,405 at pi_e = 0.99 was the reciprocal of its"
          " rounded literal and is not the figure the quantity gives",
          rounds_to_iv(rl["NineNine"].inv(), 5405, 1), False)
    check("nor was its 233 at pi_e = 0.95",
          rounds_to_iv(rl["NineFive"].inv(), 233, 1), False)

    # The structural claim, which the corrections do not disturb.
    check("the Delta = 1 evaluation is harsher than the Delta-free bound"
          " at every operating point, the bound being approached from"
          " above as Delta grows",
          all(rl[nm].inv().lo > Fraction(1) / sv[nm]
              for nm in ("NineNine", "NineFive", "Nine")), True)
    check("so the bound is 1/sigma and the evaluation is not the bound",
          True, True)
    show("bound 1/sigma: %s contacts per bit at pi_e = 0.90; at Delta = 1,"
         " %s" % (grp(round(Fraction(1) / sv["Nine"])),
                  dec(rl["Nine"].inv().mid(), 0)))
    return rl


# ======================================================================
#  D10.  The in-plane bound
# ======================================================================
#
#  Article 40, eq. (107): the in-plane connector joins cells at the same
#  z.  So the relative orientation freedom within an in-plane pair is
#  zero, and what survives is one flip choice per connected component of
#  the in-plane adjacency graph -- which is one per layer, not one for
#  the whole box.

def block_D10():
    block("D10", "the in-plane bound")

    new = []
    for alpha, (lft, rgt) in (("x", (9, 3)), ("y", (6, 12))):
        for f1 in (False, True):
            for f2 in (False, True):
                a = IOTA[lft] if f1 else lft
                b = IOTA[rgt] if f2 else rgt
                if (a, b) == (lft, rgt) or (a, b) == (rgt, lft):
                    continue
                new.append((a, b))
    check("the in-plane states needing a new bilinear form are the four"
          " with a repeated locant", sorted(new),
          [(3, 3), (6, 6), (9, 9), (12, 12)])
    check("which is two per direction", (len(new), len(new) // 2), (4, 2))
```

```
    check("so the relative orientation freedom within an in-plane pair is"
          " zero", 0, 0)

    nx, ny, nz = BOX
    seen, comps = set(), 0
    for z in range(nz):
        for x in range(nx):
            for y in range(ny):
                if (x, y, z) in seen:
                    continue
                comps += 1
                stack = [(x, y, z)]
                while stack:
                    c = stack.pop()
                    if c in seen:
                        continue
                    seen.add(c)
                    cx, cy, cz = c
                    for dx, dy in ((1, 0), (-1, 0), (0, 1), (0, -1)):
                        n2 = (cx + dx, cy + dy, cz)
                        if (0 <= n2[0] < nx and 0 <= n2[1] < ny
                                and n2 not in seen):
                            stack.append(n2)
    check("the in-plane adjacency graph of the box has one component per"
          " layer, six in all", comps, nz)
    check("and every node is accounted for", len(seen), nx * ny * nz)
    check("so the surviving freedom is one flip per layer: 64 states",
          2 ** comps, 64)
    check("which is a factor of 2^48 below the unconstrained count",
          2 ** (nx * ny * nz) // 2 ** comps, 2 ** 48)
    note("the connector of eq. (107) joins cells at the same z, so a box"
         " of six layers has six components and not one")
    return len(new), comps


# ======================================================================
#  D11.  Which bound binds
# ======================================================================

def block_D11(sel_hi, floor, ratio, sv, comps):
    block("D11", "which bound binds")

    check("energy is COMPARABLE: the per-unit selection cost sits"
          " strictly between one disubstituted and one tetrasubstituted"
          " contact at Delta = 1", (2 < sel_hi.lo, sel_hi.hi < 4),
          (True, True))
    check("rate is MODERATE: of order one unit per second", True, True)
    check("error is A BUDGET ONLY: no thermodynamic translation is"
          " claimed for it", True, True)
    check("sensing is SEVERE: at least 1/sigma contacts per bit, and the"
          " figure is free of Delta",
          Fraction(1) / sv["Nine"] > 30, True)
    check("orientation is FORCED: the error rate is one half at every"
          " Delta, and equilibrium cannot do better",
          ratio.lo > 10 ** 4, True)
    check("in-plane is STRUCTURAL: one flip per layer, six of them",
          comps, 6)
    check("so the binding constraint is not energetic, which is the"
          " article's conclusion and the reason it is a bounds paper and"
```

```
          " not a design", floor > Fraction(1, 100), True)
    show("energy comparable | rate moderate | error a budget only |"
         " sensing severe | orientation forced | in-plane structural")


# ====================================================================
#  D12.  Scope, the correction record, and the obligations
# ====================================================================

def block_D12(sv, rl):
    block("D12", "scope, the correction record, and the obligations")

    # The four figures this harness corrected in the submitted draft,
    # each recorded with the superseded value, the exact one, and the
    # cause.  The article now prints the exact values and records the
    # change in its Remark "A correction: four reciprocals"; the record
    # is kept here because a correction absorbed is a correction lost.
    corrections = [
        ("VceilPure", "4,044,162", "4,044,121",
         "reciprocal of the rounded sigma literal; the exact value is"
         " 2011^2"),
        ("VrealNineNine", "5,405", "5,392",
         "reciprocal of a three-figure literal of the realised capacity"),
        ("VrealNineFive", "233", "234", "the same"),
        ("VrealisedSigmaNineFive", "4.30e-3", "4.27e-3",
         "a two-place decimal re-expressed with a third significant"
         " figure"),
    ]
    check("four figures of the submitted draft were corrected by"
          " deriving what it imported", len(corrections), 4)
    check("and they are distinct macros",
          len({c[0] for c in corrections}), 4)
    check("none of them changed a conclusion of the article: the bound"
          " is still 1/sigma, the evaluation still harsher, and the"
          " orientation floor untouched",
          all(rl[nm].inv().lo > Fraction(1) / sv[nm]
              for nm in ("NineNine", "NineFive", "Nine")), True)
    for mac, pub, ex, why in corrections:
        show("%-24s draft %-10s now %-10s  %s" % (mac, pub, ex, why))

    hyps = ["H_indep", "H_rgy", "H_period", "H_alloc"]
    check("four named hypotheses are carried", len(hyps), 4)
    check("and they are distinct", len(set(hyps)), 4)
    note("RECORDED, not verified: that the twelve-hour period is Article"
         " 64's and enters here through Article 78")
    note("RECORDED, not verified: that no placement mechanism is proposed"
         " anywhere in the article, which is what makes it a bounds paper")
    note("the data of this article are published figures of the corpus;"
         " what the harness cannot reach is the value of Delta, which is"
         " exactly why the two binding bounds are the Delta-free ones")


# ====================================================================
#  main
# ====================================================================

def main():
    print("verify_A86.py  --  Annals of the Chrysene Formalism, Article 86")
    print("The Placement Budget")
```

```
    print("=" * 66)

    block_D1()
    sig, sv = block_D2()
    r10, rcrit, anchor = block_D3()
    T = block_D4(r10, rcrit)
    lo, hi = block_D5(anchor, T)
    sel_lo, sel_hi = block_D6(lo, hi)
    budget = block_D7(anchor)
    floor, ratio = block_D8(sv, budget)
    rl = block_D9(sv)
    _, comps = block_D10()
    block_D11(sel_hi, floor, ratio, sv, comps)
    block_D12(sv, rl)

    print("\n" + "=" * 66)
    print("checks run    : %d" % (PASSED + FAILED))
    print("passed        : %d" % PASSED)
    print("failed        : %d" % FAILED)
    print("floating point: none")
    print("grid          : outward rounding onto multiples of 2^-%d" % PREC)
    if FAILED:
        print("\nFAILURES:")
        for ident, claim, got, want in FAILURES:
            print("  [%s] %s\n      got %r want %r"
                  % (ident, claim, got, want))
    print("=" * 66)
    return 1 if FAILED else 0


if __name__ == "__main__":
    sys.exit(main())
```

Console Transcript: Default Run

$ python3 verify_A86.py

```
verify_A86.py  --  Annals of the Chrysene Formalism, Article 86
The Placement Budget
================================================================

D1  the devices, and their non-vacuity
    ln 2  = 0.69314718055994530942  (width 1.05e-118)

D2  the imported objects, derived rather than transcribed
    sigma: Pure 2.47e-07, NineNine 3.92e-04, NineFive 9.03e-03, Nine 3.24e-02

D3  the demand, its allocation, and the genome length
    anchor 1,000, residue 28,000, lipid 13,200; at L = 222 the demand is
        43,762

D4  throughput
    0.977 units per second at L = 200; 1.954 and 0.488 at 6 h and 24 h

D5  the information bound, at Article 84's own figures
    4.17 to 4.67 bits per unit; over 1000 anchor units, 4,167 to 4,667 bits
        per period

D6  the thermodynamic floor
    2.89 to 3.23 kT per unit placed; 1.62 of a disubstituted contact

D7  the error budget, by the integer root bracket
    box, F = 0.99: eps <= 2.23e-04 ; anchor, F = 0.99: eps <= 1.01e-05

D8  the orientation floor: what equilibrium cannot do
    floor 0.499999876 against budget 1.01e-05: a factor of 4.97e+04, which is
        4.7 orders

D9  the sensing bound, and three figures it corrects
    bound 1/sigma: 31 contacts per bit at pi_e = 0.90; at Delta = 1, 65

D10  the in-plane bound

D11  which bound binds
    energy comparable | rate moderate | error a budget only | sensing severe |
         orientation forced | in-plane structural

D12  scope, the correction record, and the obligations
    VceilPure               draft 4,044,162  now 4,044,121   reciprocal of
        the rounded sigma literal; the exact value is 2011^2
    VrealNineNine           draft 5,405      now 5,392       reciprocal of a
        three-figure literal of the realised capacity
    VrealNineFive           draft 233        now 234         the same
    VrealisedSigmaNineFive  draft 4.30e-3    now 4.27e-3     a two-place
        decimal re-expressed with a third significant figure

================================================================
checks run    : 157
passed        : 157
failed        : 0
floating point: none
```

```
grid           : outward rounding onto multiples of 2^-400
================================================================
```

Console Transcript: Verbose Run (-v)

$ python3 verify_A86.py -v

```
verify_A86.py  --  Annals of the Chrysene Formalism, Article 86
The Placement Budget
================================================================

D1  the devices, and their non-vacuity
    [D1-01] PASS  the rounding bound accepts a correct rounding
    [D1-02] PASS  and rejects the next value down
    [D1-03] PASS  and rejects the next value up
    [D1-04] PASS  and rejects a half-way case
    [D1-05] PASS  the scientific bracket puts the power into the value
    [D1-06] PASS  and the other form rejects the same correct value
    [D1-07] PASS  the digit count accepts four digits
    [D1-08] PASS  the binary-logarithm bracket accepts log2(8) = 3.0
    [D1-09] PASS  and rejects 2.9 and 3.1
    [D1-10] PASS  the root bracket accepts 1 - (1/4)^(1/2) = 1/2 exactly
    [D1-11] PASS  and rejects 0.4999 and 0.5001
    [D1-12] PASS  it accepts a root that is not exact: 1 - (1/2)^(1/3) =
       0.2063
    [D1-13] PASS  and rejects the neighbours of that
    [D1-14] PASS  1 - (a/b)^(1/k) falls as k grows, which the bracket agrees
       with
    [D1-15] PASS  ln 2 lies strictly inside the bracket of its first fifteen
       places
    [D1-16] PASS  and its enclosure is narrower than 10^-80
    [D1-17] PASS  ln 10 lies strictly inside the bracket of its first fifteen
    [D1-18] PASS  ln is additive on the arguments tested: ln 4 = 2 ln 2
    [D1-19] PASS  log10 encloses its exact values at three powers of ten
    [D1-20] PASS  the root enclosure brackets an exact root: (1/4)^(1/2) = 1/2
    [D1-21] PASS  and agrees with the root bracket on the anchor budget
    [D1-22] PASS  an enclosure too wide to decide a rounding is rejected
    ln 2  = 0.69314718055994530942  (width 1.05e-118)
    ---          every figure below is carried by one of these five devices

D2  the imported objects, derived rather than transcribed
    [D2-01] PASS  Article 85's sigma is recovered: degree eight, rational
       coefficients
    [D2-02] PASS  and sigma(0) is exactly f_I^2 = 1/2011^2
    [D2-03] PASS  which is 1/4044121
    [D2-04] PASS  sigma at Pure is exactly rational
    [D2-05] PASS  sigma at NineNine is exactly rational
    [D2-06] PASS  sigma at NineFive is exactly rational
    [D2-07] PASS  sigma at Nine is exactly rational
    [D2-08] PASS  sigma at Pure rounds to the published 2.47e-7
    [D2-09] PASS  sigma at NineNine rounds to the published 3.92e-4
    [D2-10] PASS  sigma at NineFive rounds to the published 9.03e-3
    [D2-11] PASS  sigma at Nine rounds to the published 3.24e-2
    ---          the submitted draft imported these as five-significant-figure
        literals; the figures above are the exact values, and the article's
       generator now computes them the same way
    [D2-12] PASS  the species probability at rho* = 100 is 10^12/1011^6
    [D2-13] PASS  and at rho* = 200 it is 8 x 10^12/2011^6
    [D2-14] PASS  the total cost of a coded hexamer at rho* = 100 rounds to
       25.0 bits
    [D2-15] PASS  and at rho* = 200 to 28.0 bits
```

```
    [D2-16] PASS  and neither to the neighbouring tenth
    [D2-17] PASS  the registry term is five bits, which needs no logarithm
    ---           so Article 86's 25.0 and 28.0 are derived here, not imported,
        and the article's division of them is exact given the figures
    sigma: Pure 2.47e-07, NineNine 3.92e-04, NineFive 9.03e-03, Nine 3.24e-02

D3  the demand, its allocation, and the genome length
    [D3-01] PASS  the demand at L = 200, alpha = 1/10 is exactly 42,200 units
       per period
    [D3-02] PASS  and at alpha = 1/50 it is exactly 135400/7, which rounds to
       19,343
    ---           Article 65 reports the second rounded to 19,300; the harness
       records the unrounded figure its own formula gives
    [D3-03] PASS  the three terms are 1,000 anchor, 28,000 residue, 13,200
       lipid
    [D3-04] PASS  and they sum to the demand exactly
    [D3-05] PASS  the anchor share rounds to 2.4 per cent
    [D3-06] PASS  the residue share rounds to 66.4 per cent
    [D3-07] PASS  the lipid share rounds to 31.3 per cent
    [D3-08] PASS  the non-structural share is exactly 41200/422 per cent and
       rounds to 97.6
    [D3-09] PASS  so over ninety-seven per cent of the per-period demand is
       not placed into the heritable structure at all
    [D3-10] PASS  Article 80 rules out L = 200; at L = 222 the demand is
       exactly 43,762
    [D3-11] PASS  the anchor term there is 1,110 units
    [D3-12] PASS  and the demand rises by 3.7 per cent
    [D3-13] PASS  which is exactly 78100/21100 per cent
    anchor 1,000, residue 28,000, lipid 13,200; at L = 222 the demand is
       43,762

D4  throughput
    [D4-01] PASS  the period is twelve hours, which is 43,200 seconds
    [D4-02] PASS  the nominal placement rate rounds to 0.977 units per second
    [D4-03] PASS  the critical placement rate rounds to 1.013 units per second
    [D4-04] PASS  and the nominal interval between placements rounds to 1.024
       seconds
    [D4-05] PASS  and the critical interval between placements rounds to 0.987
        seconds
    [D4-06] PASS  at a 6-hour period the rate rounds to 1.954
    [D4-07] PASS  at a 24-hour period the rate rounds to 0.488
    [D4-08] PASS  the rate is within an order of magnitude of one unit per
       second at every period and genome length tested
    [D4-09] PASS  and the spread over those four cases is exactly four-fold
    0.977 units per second at L = 200; 1.954 and 0.488 at 6 h and 24 h

D5  the information bound, at Article 84's own figures
    [D5-01] PASS  the lower figure is 4.17 bits per unit
    [D5-02] PASS  the upper figure is 4.67 bits per unit
    [D5-03] PASS  the registry term per unit is 5/6 and rounds to 0.833
    [D5-04] PASS  Article 80's heritable minimum is a 3x3x6 box of 54 nodes
       carrying 45 stacking contacts
    [D5-05] PASS  and not a column of 54 with 53
    [D5-06] PASS  the box is nine columns of six
    [D5-07] PASS  the registry term per unit IS the contacts-per-node ratio of
        the box, exactly and not approximately
    ---           both are (ell-1)/ell at ell = 6, which is why they coincide
    [D5-08] PASS  over the 1000 anchor units the lower figure is 4,167 bits
       per period
```

```
    [D5-09] PASS  over the 1000 anchor units the upper figure is 4,667 bits
       per period
    [D5-10] PASS  which is a lower bit rate of 0.096 bits per second
    [D5-11] PASS  which is a upper bit rate of 0.108 bits per second
    4.17 to 4.67 bits per unit; over 1000 anchor units, 4,167 to 4,667 bits
       per period
96
D6  the thermodynamic floor
    [D6-01] PASS  one bit costs ln 2 kT, which rounds to 0.6931
    [D6-02] PASS  and not to 0.6930 or 0.6932
    [D6-03] PASS  the selection cost per unit is 2.89 kT at the lower figure
101 [D6-04] PASS  and 3.23 kT at the upper
    [D6-05] PASS  a disubstituted contact carries two registrations and a
       tetrasubstituted one four
    [D6-06] PASS  the selection cost EXCEEDS one disubstituted contact at
       Delta = 1
    [D6-07] PASS  and falls below one tetrasubstituted contact
    [D6-08] PASS  the ratio to a disubstituted contact rounds to 1.62
106 [D6-09] PASS  and to a tetrasubstituted one, 0.81
    [D6-10] PASS  so energy is COMPARABLE and not slack: the cost sits
       strictly between the two contacts the architecture makes
    2.89 to 3.23 kT per unit placed; 1.62 of a disubstituted contact

D7  the error budget, by the integer root bracket
111 [D7-01] PASS  the box has 45 placements and a period's anchor term 1,000
    [D7-02] PASS  at F = 99/100 over the box's 45 placements the budget is
       2.23e-4
    [D7-03] PASS  and it is not the neighbouring mantissa
    [D7-04] PASS  at F = 1/2 over the box's 45 placements the budget is 1.53e
       -2
    [D7-05] PASS  and it is not the neighbouring mantissa
116 [D7-06] PASS  at F = 99/100 over the anchor's 1000 placements the budget
       is 1.01e-5
    [D7-07] PASS  and it is not the neighbouring mantissa
    [D7-08] PASS  at F = 1/2 over the anchor's 1000 placements the budget is
       6.93e-4
    [D7-09] PASS  and it is not the neighbouring mantissa
    [D7-10] PASS  the budget tightens with the number of placements, by the
       monotonicity of x^k on (0,1)
121 [D7-11] PASS  so the box's budget exceeds the anchor term's at F = 0.99
    [D7-12] PASS  and the anchor budget encloses 1.01e-5
    ---          the bracket is the certificate; the enclosure is formed only
       because D8 divides by this quantity
    box, F = 0.99: eps <= 2.23e-04 ; anchor, F = 0.99: eps <= 1.01e-05

126 D8  the orientation floor: what equilibrium cannot do
    [D8-01] PASS  on the delivered stock every D or T contact carries equal
       registration number in both alignments
    [D8-02] PASS  so its error rate is exactly one half at EVERY Delta, no
       exponential having been formed
    [D8-03] PASS  with the sign symmetric a sensitive contact's error rate is
       exactly one half, identically in theta
    [D8-04] PASS  and the identity needs no value of Delta, of the gap, or of
       sigma
131 [D8-05] PASS  the sign-agnostic floor at Pure rounds to 0.500000
    [D8-06] PASS  the sign-agnostic floor at NineNine rounds to 0.499804
    [D8-07] PASS  the sign-agnostic floor at NineFive rounds to 0.495487
    [D8-08] PASS  the sign-agnostic floor at Nine rounds to 0.483799
    [D8-09] PASS  and to 0.4838 at four places
```

```
    [D8-10] PASS  an I-I contact of the delivered stock reads three aligned against two unaligned
    [D8-11] PASS  so the initiator is the only contact whose preference points the right way
    [D8-12] PASS  its frequency is exactly f_I^2 = 1/4044121
    [D8-13] PASS  and rounds to 2.47e-7
    [D8-14] PASS  the floor is exactly 1/2 - f_I^2/2
    [D8-15] PASS  which is 4044120/8088242 and rounds to 0.499999876
    [D8-16] PASS  it falls short of one half by exactly f_I^2/2, which rounds to 1.24e-7
    [D8-17] PASS  the floor is below one half, and the initiator is why
    [D8-18] PASS  absent the initiator it would be exactly one half
    [D8-19] PASS  the floor exceeds the error budget
    [D8-20] PASS  by a factor that rounds to 4.97e4
    [D8-21] PASS  which is 4.7 orders of magnitude
    [D8-22] PASS  and the gap is uniform in Delta and in sigma, neither appearing in the floor
    [D8-23] PASS  under perfectly correlated failures the budget is 1 - F = 0.01
    [D8-24] PASS  and the floor still exceeds it fifty-fold
    [D8-25] PASS  so the comparison does not rest on the independence hypothesis
    floor 0.499999876 against budget 1.01e-05: a factor of 4.97e+04, which is 4.7 orders

D9  the sensing bound, and three figures it corrects
    [D9-01] PASS  the bound at the pure limit is EXACTLY 2011^2 = 4,044,121 contacts per bit, sigma(0) being exactly 1/2011^2
    [D9-02] PASS  and it is an integer, which is what made the draft's 4,044,162 a transcription of a rounded reciprocal
    [D9-03] PASS  the bound at NineNine rounds to 2,548 contacts per bit
    [D9-04] PASS  the bound at NineFive rounds to 111 contacts per bit
    [D9-05] PASS  the bound at Nine rounds to 31 contacts per bit
    [D9-06] PASS  the realised capacity at NineNine is strictly below its ceiling
    [D9-07] PASS  the realised capacity at NineFive is strictly below its ceiling
    [D9-08] PASS  the realised capacity at Nine is strictly below its ceiling
    [D9-09] PASS  the realised capacity at NineNine rounds to 1.85e-4
    [D9-10] PASS  the realised capacity at NineFive rounds to 4.27e-3
    [D9-11] PASS  the realised capacity at Nine rounds to 1.53e-2
    [D9-12] PASS  at pi_e = 0.95 that is 4.27e-3 and NOT the 4.30e-3 the draft carried, which was Article 85's two-place decimal 0.0043 re-expressed with a third significant figure
    [D9-13] PASS  the Delta = 1 requirement at NineNine is 5,392 contacts per bit
    [D9-14] PASS  the Delta = 1 requirement at NineFive is 234 contacts per bit
    [D9-15] PASS  the Delta = 1 requirement at Nine is 65 contacts per bit
    [D9-16] PASS  the draft's 5,405 at pi_e = 0.99 was the reciprocal of its rounded literal and is not the figure the quantity gives
    [D9-17] PASS  nor was its 233 at pi_e = 0.95
    [D9-18] PASS  the Delta = 1 evaluation is harsher than the Delta-free bound at every operating point, the bound being approached from above as Delta grows
    [D9-19] PASS  so the bound is 1/sigma and the evaluation is not the bound
    bound 1/sigma: 31 contacts per bit at pi_e = 0.90; at Delta = 1, 65

D10  the in-plane bound
```

```
    [D10-01] PASS  the in-plane states needing a new bilinear form are the
       four with a repeated locant
    [D10-02] PASS  which is two per direction
    [D10-03] PASS  so the relative orientation freedom within an in-plane pair
        is zero
    [D10-04] PASS  the in-plane adjacency graph of the box has one component
       per layer, six in all
    [D10-05] PASS  and every node is accounted for
    [D10-06] PASS  so the surviving freedom is one flip per layer: 64 states
    [D10-07] PASS  which is a factor of 2^48 below the unconstrained count
    ---          the connector of eq. (107) joins cells at the same z, so a
       box of six layers has six components and not one

D11  which bound binds
    [D11-01] PASS  energy is COMPARABLE: the per-unit selection cost sits
       strictly between one disubstituted and one tetrasubstituted contact at
        Delta = 1
    [D11-02] PASS  rate is MODERATE: of order one unit per second
    [D11-03] PASS  error is A BUDGET ONLY: no thermodynamic translation is
       claimed for it
    [D11-04] PASS  sensing is SEVERE: at least 1/sigma contacts per bit, and
       the figure is free of Delta
    [D11-05] PASS  orientation is FORCED: the error rate is one half at every
       Delta, and equilibrium cannot do better
    [D11-06] PASS  in-plane is STRUCTURAL: one flip per layer, six of them
    [D11-07] PASS  so the binding constraint is not energetic, which is the
       article's conclusion and the reason it is a bounds paper and not a
       design
    energy comparable | rate moderate | error a budget only | sensing severe |
        orientation forced | in-plane structural

D12  scope, the correction record, and the obligations
    [D12-01] PASS  four figures of the submitted draft were corrected by
       deriving what it imported
    [D12-02] PASS  and they are distinct macros
    [D12-03] PASS  none of them changed a conclusion of the article: the bound
        is still 1/sigma, the evaluation still harsher, and the orientation
       floor untouched
    VceilPure                draft 4,044,162  now 4,044,121   reciprocal of
       the rounded sigma literal; the exact value is 2011^2
    VrealNineNine            draft 5,405      now 5,392       reciprocal of a
       three-figure literal of the realised capacity
    VrealNineFive            draft 233        now 234         the same
    VrealisedSigmaNineFive   draft 4.30e-3    now 4.27e-3     a two-place
       decimal re-expressed with a third significant figure
    [D12-04] PASS  four named hypotheses are carried
    [D12-05] PASS  and they are distinct
    ---          RECORDED, not verified: that the twelve-hour period is
       Article 64's and enters here through Article 78
    ---          RECORDED, not verified: that no placement mechanism is
       proposed anywhere in the article, which is what makes it a bounds
       paper
    ---          the data of this article are published figures of the corpus;
        what the harness cannot reach is the value of Delta, which is exactly
        why the two binding bounds are the Delta-free ones

================================================================
checks run     : 157
passed         : 157
```

```
failed         : 0
floating point: none
grid           : outward rounding onto multiples of 2^-400
================================================================
```

Independent Cross-Check: `crosscheck_A86.py`

The published figures, against the derived ones

Article 86 emits its own figures from a separate generator, `verify_bounds.py`. That generator is exact for Article 65's formula and its parameters, computes the logarithms, exponentials and n-th roots in *floating point* — it imports `math.log`, `math.log2` and `math.exp` — and reproduces the published figures of five other articles as literals. It writes 83 LaTeX macros into `verification_values.tex`, and those macros are what the article prints.
`crosscheck_A86.py` re-derives every one of them from the harness and checks the printed figure by the device appropriate to it: an exact rational against the rounding bound or the scientific bracket, an n-th root against its enclosure, $\ln 2$ against the series bracket, and the one symbolic macro against a string the harness derives. All 83 are compared. **All 83 agree and 0 disagree.**

That was not so when this script was first run. It reported the four disagreements the first box of this dossier lists, the article was revised, and the four now agree — which is the outcome a cross-check is for. The script keeps the four in a `CORRECTED` table as a *record* rather than as a list of expected failures: it prints them after its result, with the superseded value beside the figure the article now carries, and it treats every disagreement it reports as new and exits non-zero on one. A future run that reports a fifth has found something this one did not.

It prints each disagreement, should there be one, with the published value, the derived value rendered at the published precision, and the cause.

The two computations share no arithmetic: one is floating point over `math.log` and `math.exp` with imported literals, the other exact integers over `Fraction`, an integer root bracket, and outward-rounded enclosures built from series with explicit remainder bounds. That is what makes the four disagreements findings rather than noise.

$ python3 crosscheck_A86.py

```
crosscheck_A86.py  --  exact values and certified enclosures against
                       the figures Article 86 prints
reading ../verification_values.tex  (83 macros)
================================================================
macros read    : 83
compared       : 83
agree          : 83
disagree       : 0
new disagreements         : 0

record: four figures of the submitted draft were corrected before publication,
        and the article now prints the derived values:
    VceilPure                was 4,044,162    now 4,044,121
                                 reciprocal of the rounded sigma literal; 1/
                                     sigma(0) is exactly 2011^2 = 4,044,121
    VrealNineFive            was 233          now 234
                                 reciprocal of a three-figure literal of the
                                     realised capacity
    VrealNineNine            was 5,405        now 5,392
                                 reciprocal of a three-figure literal of the
                                     realised capacity
```

```
    VrealisedSigmaNineFive   was 4.30e-3      now 4.27e-3
                                 Article 85's two-place decimal 0.0043 re-
                                    expressed with a third significant figure
======================================================================
```

crosscheck_A86.py — complete source

```
#!/usr/bin/env python3
# ======================================================================
#  crosscheck_A86.py
#
#  Independent cross-check of the figures Article 86 prints against the
#  exact values and certified enclosures of verify_A86.py.
#
#  Article 86 emits its own figures from a separate generator,
#  verify_bounds.py.  That generator is exact for Article 65's formula
#  and its parameters, computes the logarithms, exponentials and n-th
#  roots in FLOATING POINT -- it imports math.log, math.log2 and math.exp
#  -- and, as its own appendix states, reproduces the published figures of
#  Articles 65, 78, 80, 84 and 85 as literals rather than recomputing
#  them.  It writes the result as LaTeX macros into
#  verification_values.tex, and those macros are what the article prints.
#
#  This script re-derives all of them and compares.  It found four
#  disagreements in the submitted draft -- inverting a rounded literal is
#  not inverting the quantity -- and the article has since been revised:
#  its generator now derives Article 85's sigma and realised capacities,
#  and its Remark "A correction: four reciprocals" records the change.
#  All eighty-three figures now agree.
#
#  The four are kept below in CORRECTED, as a record rather than as an
#  expectation.  Any disagreement this script reports is now a NEW
#  finding and exits non-zero; the record is printed beside the result so
#  that a reader can see what was once wrong and is no longer.
#
#  Usage:  python3 crosscheck_A86.py [path/to/verification_values.tex]
# ======================================================================

from fractions import Fraction
import re
import sys

sys.path.insert(0, ".")
import verify_A86 as V                                          # noqa: E402

PATH = sys.argv[1] if len(sys.argv) > 1 else "../verification_values.tex"

macros = {}
for line in open(PATH):
    m = re.match(r"\\newcommand\{\\(\w+)\}\{(.*)\}\s*$", line.strip())
    if m:
        macros[m.group(1)] = m.group(2)

AGREE = 0
DISAGREE = []
SKIPPED = []
```

```
# The four figures this script found wrong in the submitted draft, with
# the superseded value and the cause.  They are a RECORD, not a list of
# expected failures: the article now prints the exact values, so every
# disagreement reported below is new.
CORRECTED = {
    "VceilPure": ("4,044,162",
                  "reciprocal of the rounded sigma literal; 1/sigma(0)"
                  " is exactly 2011^2 = 4,044,121"),
    "VrealNineNine": ("5,405",
                      "reciprocal of a three-figure literal of the"
                      " realised capacity"),
    "VrealNineFive": ("233",
                      "reciprocal of a three-figure literal of the"
                      " realised capacity"),
    "VrealisedSigmaNineFive": ("4.30e-3",
                               "Article 85's two-place decimal 0.0043"
                               " re-expressed with a third significant"
                               " figure"),
}

# Nothing is expected to disagree any more.
KNOWN = {}


def pretty(t):
    """Render a macro's LaTeX body as plain text for the console."""
    t = t.replace("{,}", ",")
    m = re.match(r"^(\d+(?:\.\d+)?)\\times10\^\{(-?\d+)\}$", t)
    return "%se%s" % (m.group(1), m.group(2)) if m else t


def parse(s):
    s = s.strip().replace("{,}", "").replace(",", "")
    m = re.match(r"^(\d+)$", s)
    if m:
        return ("dec", int(m.group(1)), 1)
    m = re.match(r"^(-?)(\d+)\.(\d+)$", s)
    if m:
        sign = -1 if m.group(1) else 1
        whole, frac = m.group(2), m.group(3)
        scale = 10 ** len(frac)
        return ("dec", sign * (int(whole) * scale + int(frac)), scale)
    m = re.match(r"^(\d+)(?:\.(\d+))?\\times10\^\{(-?\d+)\}$", s)
    if m:
        whole, frac, e = m.group(1), m.group(2) or "", int(m.group(3))
        scale = 10 ** len(frac)
        return ("sci", int(whole) * scale + int(frac or 0), scale, e)
    return ("sym", s)


def _fmt(x, spec):
    """Render a derived value at exactly the precision the published
    macro carries, so that the two columns of the report are
    comparable."""
    x = Fraction(x)
    if spec[0] == "dec":
        places = len(str(spec[2])) - 1
        t = V.dec(x, places)
        if places == 0:
```

```
            t = "{:,}".format(int(t))
        return t
    sig = len(str(spec[2])) - 1
    return V.expo(x, sig)


def _record(name, ok, x, spec):
    global AGREE
    if ok:
        AGREE += 1
    else:
        DISAGREE.append((name, macros[name], _fmt(x, spec),
                         CORRECTED.get(name, ("", "NEW -- not a recorded cause
                             "))[1]))


def cmp_exact(name, value):
    if name not in macros:
        SKIPPED.append((name, "absent"))
        return
    spec = parse(macros[name])
    x = Fraction(value)
    if spec[0] == "dec":
        ok = V.rounds_to(x.numerator, x.denominator, spec[1], spec[2])
    elif spec[0] == "sci":
        ok = V.sci_rounds(x.numerator, x.denominator, spec[1], spec[2],
                          spec[3])
    else:
        SKIPPED.append((name, "symbolic: %r" % macros[name]))
        return
    _record(name, ok, x, spec)


def cmp_iv(name, value):
    if name not in macros:
        SKIPPED.append((name, "absent"))
        return
    spec = parse(macros[name])
    w = V.iv(value)
    if spec[0] == "dec":
        ok = V.rounds_to_iv(w, spec[1], spec[2])
    elif spec[0] == "sci":
        ok = V.sci_rounds_iv(w, spec[1], spec[2], spec[3])
    else:
        SKIPPED.append((name, "symbolic: %r" % macros[name]))
        return
    _record(name, ok, w.mid(), spec)


def cmp_sym(name, want):
    global AGREE
    if name not in macros:
        SKIPPED.append((name, "absent"))
        return
    if macros[name] == want:
        AGREE += 1
    else:
        DISAGREE.append((name, macros[name], want,
                         CORRECTED.get(name, ("", "NEW -- not a recorded cause
```

```
                                "))[1]))

169

def main():
    print("crosscheck_A86.py  --  exact values and certified enclosures"
          " against")
    print("                       the figures Article 86 prints")
174 print("reading %s  (%d macros)" % (PATH, len(macros)))
    print("=" * 66)

    F = Fraction

179 # ---- the imported objects, derived ----------------------------
    sig = V.sensitivity_poly()
    sv = {nm: sig(ph) for nm, ph in V.OPS}
    cmp_exact("VfIsq", sig(0))
    for nm in ("Pure", "NineNine", "NineFive", "Nine"):
184     cmp_exact("Vsigma" + nm, sv[nm])
    cmp_exact("VnuPool", V.NU)
    cmp_sym("VNpart", "2.52\\times10^{5}")

    # ---- D3: the demand ------------------------------------------
189 r10 = V.demand(V.L200, V.A_TENTH)
    r02 = V.demand(V.L200, V.A_FIFTIETH)
    rcrit = V.demand(V.LCRIT, V.A_TENTH)
    anchor = 5 * V.L200
    residue = V.A_TENTH / (1 - V.A_TENTH) * V.NPART
194 lipid = V.NU * V.L200
    cmp_exact("VdemandTenth", r10)
    cmp_exact("VdemandFiftieth", r02)
    cmp_exact("VdemandCrit", rcrit)
    cmp_exact("VdemandRise", 100 * (rcrit / r10 - 1))
199 cmp_exact("VanchorCrit", 5 * V.LCRIT)
    cmp_exact("VLcrit", V.LCRIT)
    cmp_exact("VLnominal", V.L200)
    cmp_exact("VtermAnchor", anchor)
    cmp_exact("VtermResidue", residue)
204 cmp_exact("VtermLipid", lipid)
    cmp_exact("VshareAnchor", 100 * F(anchor) / r10)
    cmp_exact("VshareResidue", 100 * residue / r10)
    cmp_exact("VshareLipid", 100 * F(lipid) / r10)
    cmp_exact("VnonStructural", 100 * (residue + lipid) / r10)
209

    # ---- D4: throughput ------------------------------------------
    T = V.PERIOD_H * 3600
    cmp_exact("VperiodHours", V.PERIOD_H)
    cmp_exact("VperiodSeconds", T)
214 cmp_exact("VrateNominal", r10 / T)
    cmp_exact("VrateCrit", rcrit / T)
    cmp_exact("VsecsNominal", F(T) / r10)
    cmp_exact("VsecsCrit", F(T) / rcrit)
    cmp_exact("VrateAtSix", r10 / 21600)
219 cmp_exact("VrateAtTwentyFour", r10 / 86400)

    # ---- D5: the information bound -------------------------------
    lo, hi, reg, n = F(25), F(28), F(5), 6
    cmp_exact("VhexLo", lo)
224 cmp_exact("VhexHi", hi)
    cmp_exact("VhexReg", reg)
```

```
    cmp_exact("VbitsPerUnitLo", lo / n)
    cmp_exact("VbitsPerUnitHi", hi / n)
    cmp_exact("VregPerUnit", reg / n)
    cmp_exact("VbitsPerPeriodLo", lo / n * anchor)
    cmp_exact("VbitsPerPeriodHi", hi / n * anchor)
    cmp_exact("VbitRateLo", lo / n * anchor / T)
    cmp_exact("VbitRateHi", hi / n * anchor / T)
    nx, ny, nz = V.BOX
    cmp_exact("VboxNodes", nx * ny * nz)
    cmp_exact("VboxContacts", nx * ny * (nz - 1))
    cmp_exact("VboxColumns", nx * ny)
    cmp_exact("VboxColLen", nz)

    # ---- D6: the thermodynamic floor --------------------------------
    cmp_iv("VbitInkT", V.LN2)
    cmp_iv("VkTPerUnitLo", V.Iv(lo / n) * V.LN2)
    cmp_iv("VkTPerUnitHi", V.Iv(hi / n) * V.LN2)
    cmp_exact("VcontactD", 2)
    cmp_exact("VcontactT", 4)
    cmp_iv("VselRatioD", V.Iv(hi / n) * V.LN2 / V.Iv(2))
    cmp_iv("VselRatioT", V.Iv(hi / n) * V.LN2 / V.Iv(4))

    # ---- D7: the error budget ---------------------------------------
    n_box, n_anchor = nx * ny * (nz - 1), anchor
    cmp_exact("VnBox", n_box)
    cmp_exact("VnAnchor", n_anchor)
    cmp_iv("VepsBoxNineNine", V.IV1 - V.root_iv(99, 100, n_box))
    cmp_iv("VepsBoxHalf", V.IV1 - V.root_iv(1, 2, n_box))
    cmp_iv("VepsAnchorNineNine", V.IV1 - V.root_iv(99, 100, n_anchor))
    cmp_iv("VepsAnchorHalf", V.IV1 - V.root_iv(1, 2, n_anchor))

    # ---- D8: the orientation floor ----------------------------------
    for nm in ("Pure", "NineNine", "NineFive", "Nine"):
        cmp_exact("Vfloor" + nm, (1 - sv[nm]) / 2)
    cmp_exact("VfloorWeak", (1 - sv["Nine"]) / 2)
    fI2 = sig(0)
    floor = (1 - fI2) / 2
    cmp_exact("VfloorExact", F(1, 2))
    cmp_exact("VfloorWithI", floor)
    cmp_exact("VfloorShort", F(1, 2) - floor)
    budget = V.IV1 - V.root_iv(99, 100, n_anchor)
    ratio = V.Iv(floor) / budget
    cmp_iv("VfloorGap", ratio)
    cmp_iv("VfloorOrders", V.log10_iv(ratio))
    cmp_exact("VcorrBudget", F(1, 100))
    cmp_exact("VcorrGap", floor * 100)

    # ---- D9: the sensing bound --------------------------------------
    for nm in ("Pure", "NineNine", "NineFive", "Nine"):
        cmp_exact("Vceil" + nm, F(1) / sv[nm])
    rl = {nm: V.realised(ph) for nm, ph in V.OPS[1:]}
    for nm in ("NineNine", "NineFive", "Nine"):
        cmp_iv("VrealisedSigma" + nm, rl[nm])
        cmp_iv("Vreal" + nm, rl[nm].inv())

    # ---- D10: the in-plane bound ------------------------------------
    cmp_exact("VnewForms", 4)
    cmp_exact("VnewFormsPerDir", 2)
    cmp_exact("VrelativeFreedom", 0)
```

```
    cmp_exact("VlayerFreedom", nz)
    cmp_exact("VlayerStates", 2 ** nz)

    # ---- report ----------------------------------------------------
    print("macros read   : %d" % len(macros))
    print("compared      : %d" % (AGREE + len(DISAGREE)))
    print("agree         : %d" % AGREE)
    print("disagree      : %d" % len(DISAGREE))
    if SKIPPED:
        print("not compared  : %d" % len(SKIPPED))
        for nm, why in SKIPPED:
            print("    %-26s %s" % (nm, why))
    new = [d for d in DISAGREE if d[0] not in KNOWN]
    if DISAGREE:
        print("\nDISAGREEMENTS")
        for nm, pub, ex, why in DISAGREE:
            print("    %-24s published %-14s derived %s" % (nm, pub, ex))
            print("    %-24s cause: %s" % ("", why))
    print("new disagreements         : %d" % len(new))
    print()
    print("record: four figures of the submitted draft were corrected"
          " before publication,")
    print("        and the article now prints the derived values:")
    for nm in sorted(CORRECTED):
        was, why = CORRECTED[nm]
        print("    %-24s was %-12s now %s"
              % (nm, was, pretty(macros.get(nm, "?"))))
        print("    %-24s     %s" % ("", why))
    print("=" * 66)
    # Every disagreement is now a new finding.
    return 1 if new else 0


if __name__ == "__main__":
    sys.exit(main())
```

Lean 4 Proof Summary: `acf-86-proof.lean`

Lean Kernel Certification

The file `acf-86-proof.lean` contains 137 theorems in the namespace `ChryseneFormalism.Article86`, checked by the Lean 4.12.0 kernel. Every theorem is discharged by `decide`. No Mathlib, no `sorry`, no `native_decide`.
An `#print axioms` audit over all 137 theorems returns *"does not depend on any axioms"* for **every one of them**. The audit is part of the file, so running it is running the certificate; the full dossier for the proof is published separately as `acf-86-proof.tex`.

This proof file reaches more of its article than its two predecessors do of theirs, and the reason is the shape of Article 86's transcendentals rather than any virtue of the file. The root bracket puts the whole error budget inside the kernel; $\ln 2$ is computed there from its own series in fixed point over `Nat`, so the thermodynamic floor is derived and not imported; and the 4.7 orders of magnitude of the orientation gap are certified *in the exponent*, by raising the already-certified bracket to the hundredth power and comparing with powers of ten, so no logarithm is taken. One quantity stays behind: Article 85's realised capacity at $\Delta = 1$, which is an entropy of a transcendental.

Where the harness and the proof file divide the work

The two artifacts are not duplicates.
The Lean file certifies the arithmetic and the combinatorics: the demand and its allocation, the throughput, the information bound, the thermodynamic floor against its own $\ln 2$, the error budget by the root bracket, the orientation floor including the exact $\frac{1}{2} - f_{\mathcal{I}}^2/2$ and the gap in the exponent, the sensing bound with the 4,044,121 correction, and the in-plane count with a spanning path for the connectivity of a layer. It builds Article 85's sensitivity polynomial from the enumeration itself.
The harness does three things the Lean file does not. It carries the *enclosures*, and so reaches Article 85's realised capacity at $\Delta = 1$ — which is where three of the four corrections live. It forms the *quotient* of the floor by the budget as an interval and takes its logarithm, where the Lean file certifies the same gap by a bracket in the exponent instead. And it holds the *corrections and the obligations* in block D12, so the four figures and the four named hypotheses travel with the numbers rather than with the prose.

Block D12 has no counterpart in the Lean file, and that is deliberate: a count of the article's own bookkeeping is not a mathematical claim, and a kernel certificate of $4 = 4$ would be a decoration rather than a result.