# ACF 85 — Verification Dossier

Equivariance of the Source Measure Off the Pure Limit:

Contact Sensitivity, Channel Capacity, and the Invariance of Selection Cost

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

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

Verification script: `verify_A85.py` Cross-check: `crosscheck_A85.py` Lean proof: `acf-85-proof.lean`

## Certification Summary

| Item | Value |
|---|---|
| Script | `verify_A85.py` |
| Total checks | 283 |
| Passed | **283** |
| Failed | **0** |
| Exit code | 0 |
| Blocks | 12 (C1–C12) |
| Blocks using floating point | **0** |
| Dependencies | Python standard library only |
| Cross-check script | `crosscheck_A85.py` |
| Published macros read | 153 |
| Compared | 153 |
| Disagreements | **0** |
| Lean proof file | `acf-85-proof.lean` |
| Lean version | `4.12.0` (core only, no Mathlib) |
| Lean theorems | 129 |
| Theorems with no axioms | **129** |
| `sorry` | 0 |
| `native_decide` | 0 |

### "Exact throughout" means something different here, and the difference is the article's

**The Article 84 dossier's headline cannot be repeated verbatim for this paper, and the honest course is to say so before the block index rather than after it.**
Article 84 is rational end to end. Its admitted fractions are ratios of integers, its orbit sizes are powers of two, and the handful of quantities that are not rational — costs in bits — are logarithms of rationals, which an integer bracket pins exactly. Article 85 is not like that. Its central quantities are binary entropies of numbers of the form $1/(1+e^{g\Delta})$, and $e$ is not rational. *No integer bracket reaches them*, and a harness that claimed to certify them by one would be claiming the wrong thing.
So this harness is exact in the only sense available, and the sense is worth stating precisely. Block C1 computes $e$ from the factorial series and $\ln 2$ from the inverse-hyperbolic-tangent

series, each with an *explicit rational remainder bound*; every subsequent operation rounds OUTWARD onto the dyadic grid of multiples of $2^{-400}$. An interval built that way provably contains the true value of the expression that built it, and no operation ever narrows one. A printed figure is then certified by showing that the *whole* enclosure falls strictly inside the rounding cell of that figure.

That is stronger than any floating-point computation, which cannot certify a rounding at all; and it is weaker than Article 84's brackets, which decide one without an enclosure. The device is not vacuous: given an enclosure too wide to decide a rounding it returns `False` rather than guessing, and check C1-24 is that refusal.

There is no floating-point arithmetic anywhere in `verify_A85.py`. The name `float` appears in it nowhere; the console formatter rounds exact `Fraction`s by integer division, so a printed line and the check that certifies it cannot disagree.

## Purpose and Scope

This dossier accompanies Article 85, "Equivariance of the Source Measure Off the Pure Limit: ," which asks what Article 79's own cleavage model does to Article 84's blindness theorem. The answer is that cleavage opens an orientation channel and that the channel has no preferred direction, so Article 84's selection cost stands unamended while a quantity Article 84 did not have — a capacity — appears beside it.

The verification script executes 283 independent checks in 12 blocks (C1–C12), covering the devices and the certified enclosures; the stream of unit types and the admitted fractions; the separation criterion and its symmetry; the dichotomy between $\mathcal{D}$ and the other two families, with the alphabet threshold at four letters and the two false lemmas of the submitted proof; the sensitivity polynomial, derived by enumeration; the registration gap; the bias and the information the source supplies, with the $\sigma$ ceiling checked against the full pair ensemble rather than resummed; the absence of a preferred direction and the exact uniformity it gives; the trade against Article 79's arm efficiency, with a positivity certificate for the monotonicity; the capacity of the channel and what it is not; how much of $H_{\text{frame}}$ cleavage refines, by transfer matrix; and the scope. All 283 checks passed with exit code 0.

A second script, `crosscheck_A85.py`, holds the harness's exact values and certified enclosures against all 153 LaTeX macros the article's own generator emits. The companion Lean 4 proof file `acf-85-proof.lean` provides kernel-checked formal certificates for the combinatorial and rational part of the article; it does not reach the entropies, and the proof dossier says so first.

## The devices, and which quantity each one is for

| Device | Certifies | By |
|---|---|---|
| `rounds_to(n,d,r,s)` | $n/d$ rounds to $r/s$ | $2\,\lvert ns - rd\rvert < d$ |
| `sci_rounds(n,d,m,c,e)` | $n/d$ rounds to $\frac{m}{c}10^e$ | the power into the *value* |
| `digits_is(n,k)` | $n$ has $k$ decimal digits | $10^{k-1} \le n < 10^k$ |
| `rounds_to_iv(v,r,s)` | the *whole enclosure* rounds to $r/s$ | $\frac{2r-1}{2s} < v_{\text{lo}},\, v_{\text{hi}} < \frac{2r+1}{2s}$ |

The fourth is this harness's characteristic device, as the binary-logarithm bracket is of the Article 84 harness and the antilogarithm bracket is of the Article 82 harness. What it replaces is not a logarithm of a rational — a bracket handles that — but an entropy of a transcendental, which no bracket reaches.

The enclosure core is *validated and not trusted*. Block C1 checks that $e$ and $\ln 2$ lie strictly inside the decimal brackets of their first fifteen places; that both enclosures are narrower than $10^{-80}$; that $\log_2 2^k$ encloses $k$ exactly at four values of $k$ and that $\log_2$ is strictly increasing on nineteen rationals; that $e^3$ is enclosed by $e \cdot e \cdot e$ and $e^{12}$ by its own two decimals; and that the binary entropy encloses its two exactly known values, $H(1/2) = 1$ and $H(1/4) = 2 - \frac{3}{4}\log_2 3$, and is symmetric about one half. Twenty-seven checks go on the devices before any figure of the article is touched.

Checks C1-07 and C1-08 exist because the power of ten in a scientific-notation certificate belongs in the *value* and not in the target: holding it against the target keeps the tolerance at a half-unit of the target's last place while the value grows by $10^e$. The two forms differ, the difference is not a matter of taste, and those two checks foreclose it.

## Block Index

| Block | Topic | Checks |
|---|---|---|
| C1 | The devices: integer brackets and certified enclosures | 27 |
| C2 | The stream: units, arm words and the admitted fractions | 17 |
| C3 | The separation criterion, and its symmetry (Prop. "The separation criterion") | 9 |
| C4 | Why $\mathcal{D}$ behaves differently from $\mathcal{I}$ and $\mathcal{T}$, and the alphabet threshold | 21 |
| C5 | The sensitivity polynomial, derived by enumeration (Thm. "The sensitivity polynomial") | 22 |
| C6 | The registration gap at a sensitive contact | 20 |
| C7 | The bias, and the information the source supplies (Thm. "The channel") | 33 |
| C8 | The channel has no preferred direction (Thm. "No preferred direction") | 18 |
| C9 | The trade against Article 79's arm efficiency | 20 |
| C10 | The capacity of the channel, and what it is not | 48 |
| C11 | How much of $H_{\text{frame}}$ cleavage refines | 45 |
| C12 | Scope, obligations and the review record (RECORDED) | 3 |
| | **Total** | **283** |

## Scope of Verification

**What is verified.**

- That the separation criterion agrees with sensitivity on all 784 ordered contacts of the two-letter stream and on all 13 689 of the three-letter one, in *both* directions, so that a one-unit test decides a two-unit property; that it is symmetric; and that the equality of the two cross-overlaps — which the article is explicit is an identity and not the content — holds on every contact.

- That $\mathcal{D}$ is the exception and $\mathcal{I}$ behaves like $\mathcal{T}$: two sighted $\mathcal{D}$ units always make a sensitive contact, while 40 of the both-sighted $\mathcal{T}$ contacts and 24 of the $\mathcal{I}$ ones are blind. The dichotomy is $\mathcal{D}$ against the rest, and the harness says so rather than letting the reader infer it.

- **The difference formula**, on every sighted $\mathcal{D}$–$\mathcal{D}$ pair, with values exactly $\pm 2$ over $\{h, e\}$ and never zero over three letters; and the THRESHOLD: a zero difference needs four letters, so the proposition is a statement about alphabets of at most three, and the harness checks it at the first alphabet where it fails — which a test confined to two and three letters could not see.

- That the two lemmas the submitted proof asserted are both false, with 108 and 54 counterexamples counted rather than absorbed, and that the difference formula is therefore what carries the proposition.

- That the sensitivity polynomial has degree exactly eight, that $\sigma(0)$ is *exactly* $f_{\mathcal{I}}^2$ — the point of contact with Article 84 — that the first correction is linear with coefficient $38/2011^2$, that the quadratic coefficient is exactly $16\,176\,842/4\,044\,121$ and not the integer four, and that at one per cent cleavage the quadratic term alone carries the value to within fifteen per cent.

- That the registration gap is at least one and at most four; that at perfect arming every sensitive contact has gap one with mass exactly $f_{\mathcal{I}}^2$; and that the conditional distribution is a *near point mass* on gap two, share above 0.99 at every operating point — which is why a mean of 2.000 must not be read as a balance of small and large gaps.

- **The $\sigma$ ceiling**, checked without circularity. A sensitive contact carries at most one bit and a blind one carries none, so the information per contact is at most $\sigma$ whatever the registration free energy is. This is *not* checked by resumming the conditional gap distribution, whose total is $\sigma$ by construction; it is checked against the full pair ensemble, in which every contact appears, the blind mass is shown to be exactly $1-\sigma$, and a blind contact is shown to contribute nothing. Six values of $\Delta$ at each of three operating points.

- That the channel has *no preferred direction*: the relabelling $a \mapsto a \circ \iota$ is a measure-preserving pattern-reversing involution of the words of $\mathcal{D}$ and of $\mathcal{T}$ and is not defined on $\mathcal{I}$; that the signed bias of the stream is EXACTLY zero without the initiator and of order $f_{\mathcal{I}}^2$ with it; and that the marginal law of the alignment field is EXACTLY uniform without the initiator at every cleavage level, free energy and column length tested, so that Article 84's selection cost of $\ell - 1$ bits is unamended at every $\pi_h$ and not merely in the limit.

- That the tolerance with the initiator present is stated in the *scale-free* quantity. An absolute deviation of the field mass bounds nothing at $\ell = 54$, the uniform mass itself shrinking with length; the per-contact deviation of the cost in bits is $2.83 \times 10^{-7}$, one part in $3.54 \times 10^{6}$, uniformly in $\ell$, $\pi_h$ and $\Delta$.

- That the trade is genuine: arm efficiency is exactly rational and strictly decreasing while $\sigma$ strictly increases, the latter **by certificate rather than by grid** — a grid is not an argument about a degree-eight polynomial, and the eight Bernstein coefficients of $\sigma'$ on $[0, 1/5]$ are all positive.

- That the capacity is $(\ell - 1)\,E[1 - H(\theta)]$ and the conditional entropy $(\ell - 1)\,E[H(\theta)]$ — the two numbers the submitted version reported, under the names the mathematics supports — and that the *surprisal range* belongs with them, because the conditional cost of a particular prescribed field is field-dependent and can exceed $\ell - 1$; and that averaged over a uniformly chosen field it exceeds $\ell - 1$ by Gibbs' inequality, which is algebra and needs no value of $\Delta$.

- That adjacent contacts are *positively correlated* through the unit they share, by factors of 5.6, 10.5 and 50.4, so the binomial count does not apply; and that every distributional figure — $P$(at least one sensitive contact) at $\ell = 6$ and $\ell = 54$, and the exact half-lengths of 26, 85 and **1802** sites — is a transfer-matrix computation in exact integers, with the binomial figure reported beside it and shown to understate the length at every operating point.

- That the share of a column's contacts on which $H_{\text{frame}}$ is discharged is $\sigma$ and not the probability that the column carries one, so that at the heaviest cleavage over ninety-six per cent of contacts still need the reference: a *refinement* of $H_{\text{frame}}$ and not its discharge.

- That Article 80's 54 is the node count of a $3 \times 3 \times 6$ box and not a column length, so the box

carries 45 stacking contacts and not 53, and the two figures belong to different objects.

**What is not verified.** Not the data, because there is none: the article's inputs are definitions of the corpus. What is not reached is $\Delta$, and three further limits belong with it.

- **$\Delta$ is nowhere measured in the corpus.** Article 79's registration free energy is its least constrained parameter, and every figure of blocks C7, C10 and the trade ratios of C9 is a function of it. That is exactly why the $\sigma$ ceiling is the operative statement of the article: it is the only $\Delta$-*free* bound on what the channel carries, and the harness certifies it at six values of $\Delta$ rather than pretending to a value.
- **Cleavage is a chemical process this article does not propose a mechanism for.** $\pi_h$ is a parameter of Article 79's model, and the three operating points are that paper's, computed here at $\rho^* = 200$ and stated as such rather than imported. Nothing here says an agent can set $\pi_h$.
- **The enclosures are rigorous, not infinitely sharp.** Every interval provably contains its value, and every certified rounding is a theorem about the interval. But an enclosure is not a closed form: the harness certifies that the information per sensitive contact at gap two and $\Delta = 2$ rounds to 0.870, and it does not tell you what that number *is*.
- **The harness is a regression test on the conclusions.** Its assertions encode them: block C8 fails if the field law is *not* exactly uniform without the initiator, and C11 if adjacent contacts are *not* positively correlated. 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_A85.py`

**verify_A85.py — complete source**

```
#!/usr/bin/env python3
# ======================================================================
#  verify_A85.py
#  Verification harness for
#    Annals of the Chrysene Formalism, Article 85
#    "The Partially Sighted Source: Arm Cleavage as an Orientation
#     Channel, and the Price of Blindness"
#    Charles D. Schaper, Ph.D., Schaper Systems Press
#
#  Python standard library only.  No third-party dependency, no network,
#  no data file: every quantity is derived here from the definitions of
#  the article.
#
#  EXACT THROUGHOUT, IN A SENSE THAT HAS TO BE STATED, BECAUSE IT IS NOT
#  THE SENSE OF THE ARTICLE 84 HARNESS.
#
#  Article 84 is rational end to end: its admitted fractions are ratios
#  of integers, its orbit sizes are powers of two, and the only
#  quantities that are not rational -- a handful of costs in bits -- are
#  pinned by a logarithm bracket.  Article 85 is not like that.  Its
#  central quantities are binary entropies of numbers of the form
#  1/(1 + e^{g*Delta}), and e is not rational.  No integer bracket
#  reaches them, and a harness that claimed otherwise would be claiming
#  the wrong thing.
#
#  So this harness is exact in the only sense available: it carries
#  CERTIFIED RATIONAL ENCLOSURES.  Block C1 computes e and ln 2 from
#  their series with explicit rational remainder bounds, and every
#  subsequent operation rounds OUTWARD onto a dyadic grid of 2^-400, so
#  that each interval provably contains its true value.  A printed figure
#  is then certified by showing that the WHOLE enclosure falls strictly
#  inside the rounding cell of that figure -- which is a stronger
#  statement than any floating-point computation can make, and a weaker
#  one than Article 84's brackets, and the difference is the article's
#  doing rather than the harness's.
#
#  There is no floating-point arithmetic anywhere in this file.  `float`
#  appears only in formatting a number for the console.
#
#  Usage:
#      python3 verify_A85.py         summary only
#      python3 verify_A85.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):
    """Open a block.  Blocks are the units of the dossier's index."""
    global _BLOCK, _N
    _BLOCK = name
    _N = 0
    print("\n%s  %s" % (name, title))


def check(claim, got, want):
    """Record one check.  `got` and `want` are compared with ==."""
    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):
    """A line of commentary.  Asserts nothing and is counted as nothing."""
    if VERBOSE:
        print("    ---          %s" % text)


def show(text):
    """A line of computed output, printed in both modes."""
    print("    %s" % text)


def dec(x, places):
    """Decimal string of an exact Fraction, ROUNDED half up, for the
    console.  Formatting only: nothing is computed from this, and the
    rounding matches the devices above 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)
    if places == 0:
        s = "%d" % n
    else:
        s = "%d.%0*d" % (n // scale, places, n % scale)
    return ("-" + s) if neg else s
```

```
def expo(x, sig=2):
    """Scientific string of an exact positive Fraction, for the console."""
    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"):            # rounding carried into the exponent
        y /= 10
        e += 1
        s = dec(y, sig)
    return "%se%+03d" % (s, e)


# ====================================================================
#  C1.  The devices: integer brackets, and certified enclosures
# ====================================================================
#
#  Four integer brackets carry the rational figures:
#
#    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
#    digits_is(n,k)          n has k decimal digits   10^(k-1) <= n < 10^k
#    frac_lt(a,b)            a < b, exactly
#
#  and an interval arithmetic carries the rest.  The interval core is the
#  characteristic device of THIS harness, as the binary-logarithm bracket
#  is of the Article 84 harness and the antilogarithm bracket is of the
#  Article 82 harness.  What it replaces is not a logarithm of a rational
#  -- that a bracket handles -- but an entropy of a transcendental, which
#  no bracket reaches.

PREC = 400
GRID = 1 << PREC


def _dn(x):
    """Round a Fraction DOWN onto the grid of multiples of 2^-PREC."""
    return Fraction(x.numerator * GRID // x.denominator, GRID)


def _up(x):
    """Round a Fraction UP onto the same grid."""
    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.  No operation ever narrows an interval."""
```

```
    __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, never the target: holding it
    against the target would keep the tolerance at a half-unit of the
    target's last place while the value grew by 10^e.  Checks C1-07 and
    C1-08 foreclose the confusion.
    """
    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):
    """True iff the decimal expansion of n has exactly k digits."""
    assert n > 0 and k >= 1
    return 10 ** (k - 1) <= n < 10 ** k


def rounds_to_iv(v, r, s):
    """True iff the WHOLE enclosure rounds to r/s.  This is the device
    that certifies a transcendental figure: it is a statement about an
    interval, and it 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):
    """The same, in scientific notation."""
    v = iv(v)
    if e >= 0:
        w = v * Iv(Fraction(1, 10 ** e))
    else:
        w = v * Iv(10 ** (-e))
    return rounds_to_iv(w, mant, scale)


# --- e, by the factorial series with an explicit tail bound -----------
def e_iv(terms=70):
    """e = sum 1/k!.  The tail beyond 1/terms! is at most 2/terms!, so
    the enclosure is rigorous and its width is recorded in C1."""
    s = IV0
    t = IV1
    for k in range(1, terms + 1):
        s = s + t
        t = t / Iv(k)
    return Iv(s.lo, s.hi + t.hi * 2)
```

```
E = e_iv()
_EPOW = {0: IV1}


def exp_int(n):
    """e^n for a non-negative integer n.  Only integer exponents occur:
    the article's registration free energies are 1, 2, 3, 6, 12 and 24
    and the gaps are 1 to 4, so g*Delta is always an integer."""
    assert n >= 0
    if n not in _EPOW:
        _EPOW[n] = exp_int(n - 1) * E
    return _EPOW[n]


# --- ln, by the inverse-hyperbolic-tangent series ---------------------
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)), which is added
    to the upper end.  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()
_LNCACHE = {}


def ln_iv(x):
    """ln(x) for a positive rational x.  Powers of two are taken out
    first, so the series is only ever asked for an argument in [1, 2) and
    hence for a z in [0, 1/3)."""
    x = Fraction(x)
    assert x > 0
    if x in _LNCACHE:
        return _LNCACHE[x]
    k = 0
    y = x
    while y >= 2:
        y /= 2
        k += 1
    while y < 1:
        y *= 2
        k -= 1
    z = (y - 1) / (y + 1)
    r = Iv(k) * LN2 + Iv(2) * _atanh(z, 120)
    _LNCACHE[x] = r
    return r
```

```
def log2_iv(x):
    return ln_iv(x) * _INVLN2


def log2_of_iv(p):
    """log2 of an interval of positive rationals.  log2 is increasing, so
    the enclosure is the pair of endpoint enclosures."""
    assert p.lo > 0
    return Iv(log2_iv(p.lo).lo, log2_iv(p.hi).hi)


def hbin_iv(p):
    """Binary entropy of an interval contained in (0, 1)."""
    p = iv(p)
    assert p.lo > 0 and p.hi < 1
    q = IV1 - p
    return -(p * log2_of_iv(p) + q * log2_of_iv(q))


def block_C1():
    block("C1", "the devices: integer brackets and certified enclosures")

    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("the digit count accepts 1000 at four digits",
          digits_is(1000, 4), True)
    check("and rejects three", digits_is(1000, 3), False)
    check("the scientific bracket accepts 1/2011 = 4.97e-4",
          sci_rounds(1, 2011, 497, 100, -4), True)
    check("and rejects 4.98e-4", sci_rounds(1, 2011, 498, 100, -4), False)
    check("it puts a positive exponent into the value, not the target",
          (sci_rounds(12273, 1000, 12, 10, 1),
           rounds_to(12273, 1000, 120, 10)), (True, False))

    # The interval core, validated rather than trusted.
    check("every interval contains its own midpoint",
          all(x.lo <= x.mid() <= x.hi
              for x in (E, LN2, exp_int(3), log2_iv(10))), True)
    check("e lies strictly inside the decimal bracket of its first"
          " fifteen places",
          (Fraction(2718281828459045, 10 ** 15) < E.lo,
           E.hi < Fraction(2718281828459046, 10 ** 15)), (True, True))
    check("ln 2 lies strictly inside the bracket of its first fifteen",
          (Fraction(693147180559945, 10 ** 15) < LN2.lo,
           LN2.hi < Fraction(693147180559946, 10 ** 15)), (True, True))
    check("the enclosures of e and of ln 2 are narrower than 10^-80",
          (E.width() < Fraction(1, 10 ** 80),
           LN2.width() < Fraction(1, 10 ** 80)), (True, True))

    # log2 is validated on its exact values, which the series must
    # reproduce to within its own enclosure.
    for k in (1, 3, 10, 20):
        L = log2_iv(2 ** k)
        check("log2(2^%d) encloses %d exactly" % (k, k),
              L.lo <= k <= L.hi, True)
    check("log2(10) encloses 3.3219280948873623",
          (Fraction(33219280948873623, 10 ** 16) - log2_iv(10).lo
```

```
           < Fraction(1, 10 ** 15)), True)
    check("log2 is strictly increasing on the arguments tested",
          all(log2_iv(Fraction(k, 7)).hi < log2_iv(Fraction(k + 1, 7)).lo
              for k in range(1, 20)), True)

    # exp, validated by its functional equation.
    check("e^3 is enclosed by e * e * e",
          (exp_int(3).lo <= (E * E * E).hi
           and (E * E * E).lo <= exp_int(3).hi), True)
    check("e^12 is bracketed by the two decimals 162754.7914190039 and"
          " 162754.7914190040",
          (Fraction(1627547914190039, 10 ** 10) < exp_int(12).lo,
           exp_int(12).hi < Fraction(1627547914190040, 10 ** 10)),
          (True, True))

    # The entropy, validated at the one point where it is rational.
    H = hbin_iv(Iv(Fraction(1, 2)))
    check("H(1/2) encloses one exactly", H.lo <= 1 <= H.hi, True)
    H4 = hbin_iv(Iv(Fraction(1, 4)))
    check("H(1/4) is bracketed by the two decimals of 2 - (3/4) log2 3",
          (Fraction(81127812445913, 10 ** 14) < H4.lo,
           H4.hi < Fraction(81127812445914, 10 ** 14)), (True, True))
    check("H is symmetric about one half, to within the enclosures",
          (hbin_iv(Iv(Fraction(3, 10))).lo <= hbin_iv(Iv(Fraction(7, 10))).hi
           and hbin_iv(Iv(Fraction(7, 10))).lo
           <= hbin_iv(Iv(Fraction(3, 10))).hi), True)

    # The interval device is not vacuous: it refuses a wrong rounding and
    # it refuses to decide one it cannot.
    info11 = IV1 - hbin_iv(theta_iv(1, 1))
    check("the enclosure device accepts the printed 0.160",
          rounds_to_iv(info11, 160, 1000), True)
    check("and rejects 0.161", rounds_to_iv(info11, 161, 1000), False)
    check("and rejects 0.159", rounds_to_iv(info11, 159, 1000), False)
    check("an enclosure too wide to decide a rounding is rejected",
          rounds_to_iv(Iv(Fraction(159, 1000), Fraction(161, 1000)),
                       160, 1000), False)
    show("e      = %s  (width %s)" % (dec(E.mid(), 20), expo(E.width())))
    show("ln 2   = %s  (width %s)" % (dec(LN2.mid(), 20),
                                      expo(LN2.width())))
    note("every transcendental figure below is certified by an enclosure"
         " of this kind, never by a floating-point comparison")


# ======================================================================
#  C2.  The stream: units, arm words, and the admitted fractions
# ======================================================================
#
#  Article 85 works over the two-letter arm alphabet {h, e}: Article 79's
#  phosphorylation is downstream of admission, so an e_P arm is not in
#  the admitted stream.  The three-letter alphabet is run below only to
#  record which statements are alphabet-independent.

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")
ARMS3 = ("h", "e", "eP")

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 unit_blind(S, a):
    return pres(S, a) == pres_flip(S, a)


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)


def word_prob(S, a, ph):
    """Exact probability of an arm word under independent cleavage."""
    p = Fraction(1)
    for k in sorted(S):
        p *= ph if a[k] == "h" else (1 - ph)
    return p


def stream(ph, with_I=True, arms=ARMS2):
    """The admitted stream as [(family, S, word, exact probability)]."""
    fr = dict(FRAC)
    if not with_I:
        s = fr["D"] + fr["T"]
        fr = {"D": fr["D"] / s, "I": Fraction(0), "T": fr["T"] / s}
    out = []
    for n, S in FAMS:
```

```
        if fr[n] == 0:
            continue
        for a in arm_words(S, arms):
            out.append((n, S, a, fr[n] * word_prob(S, a, ph)))
    return out


def block_C2():
    block("C2", "the stream: units, arm words and the admitted fractions")

    check("iota is an involution of the four locants",
          all(IOTA[IOTA[k]] == k for k in LOC), True)
    check("iota fixes D and T setwise but not I",
          (frozenset(IOTA[k] for k in D) == D,
           frozenset(IOTA[k] for k in T) == T,
           frozenset(IOTA[k] for k in I_) == I_), (True, True, False))

    check("the admitted fractions sum to one exactly", FD + FI + FT,
          Fraction(1))
    check("f_D, f_I, f_T are 2000/2011, 1/2011, 10/2011",
          (FD, FI, FT),
          (Fraction(2000, 2011), Fraction(1, 2011), Fraction(10, 2011)))
    check("f_I rounds to 4.97e-4",
          sci_rounds(FI.numerator, FI.denominator, 497, 100, -4), True)
    check("f_I squared is exactly 1/2011^2 and rounds to 2.47e-7",
          (FI * FI == Fraction(1, 2011 ** 2),
           sci_rounds(1, 2011 ** 2, 247, 100, -7)), (True, True))

    # The arm words, and the twenty-eight unit types of the two-letter
    # alphabet.  The count is what every transfer matrix below runs over.
    counts = tuple(len(arm_words(S, ARMS2)) for _, S in FAMS)
    check("the arm words over {h,e} number 4 on D, 8 on I, 16 on T",
          counts, (4, 8, 16))
    check("so the stream has twenty-eight unit types", sum(counts), 28)
    check("and nine over a single arm letter",
          sum(len(arm_words(S, ("e",))) for _, S in FAMS), 3)

    for ph in (Fraction(0), Fraction(1, 100), Fraction(1, 20),
               Fraction(1, 10), Fraction(1, 2)):
        tys = stream(ph)
        check("the stream is normalised exactly at pi_h = %s" % ph,
              sum(t[3] for t in tys), Fraction(1))
    tys = stream(Fraction(1, 10), with_I=False)
    check("and with the initiator removed it is still normalised",
          sum(t[3] for t in tys), Fraction(1))
    check("removing the initiator leaves twenty unit types", len(tys), 20)

    # At perfect arming the stream is one word per family.
    tys0 = [t for t in stream(Fraction(0)) if t[3] != 0]
    check("at perfect arming only the all-e word of each family survives",
          sorted((t[0], tuple(sorted(t[2].items()))) for t in tys0),
          sorted(("D", ((6, "e"), (12, "e"))) for _ in (0,))
          + sorted([("I", ((3, "e"), (6, "e"), (12, "e"))),
                    ("T", ((3, "e"), (6, "e"), (9, "e"), (12, "e")))]))
    note("the alphabet is {h, e}: an e_P arm is downstream of admission")
    return tys


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

```
# C3. The separation criterion
# ======================================================================
#
# Article 85, Prop. "The separation criterion". A contact is
# orientation-sensitive exactly when the second unit's pattern separates
# the first unit's pattern from its flip by overlap count. The content
# is that the criterion may be tested on ONE of the two units.

def block_C3():
    block("C3", "the separation criterion, and its symmetry")

    for arms in (ARMS2, ARMS3):
        n = 0
        agree = 0
        for _, S1 in FAMS:
            for _, S2 in FAMS:
                for a1 in arm_words(S1, arms):
                    for a2 in arm_words(S2, arms):
                        P1, Q1 = pres(S1, a1), pres_flip(S1, a1)
                        P2 = pres(S2, a2)
                        lhs = sensitive(S1, a1, S2, a2)
                        rhs = len(P1 & P2) != len(Q1 & P2)
                        n += 1
                        agree += (lhs == rhs)
        check("the criterion agrees with sensitivity on all %d ordered"
              " pairs over %d arm types" % (n, len(arms)), agree, n)
        if arms is ARMS2:
            check("and the pair count over {h,e} is 784", n, 784)
        else:
            check("and over three arm types it is 13,689", n, 13689)

    for arms in (ARMS2, ARMS3):
        bad = 0
        for _, S1 in FAMS:
            for _, S2 in FAMS:
                for a1 in arm_words(S1, arms):
                    for a2 in arm_words(S2, arms):
                        if sensitive(S1, a1, S2, a2) != sensitive(
                                S2, a2, S1, a1):
                            bad += 1
        check("sensitivity is symmetric in the two units, %d arm types"
              % len(arms), bad, 0)

    # The identity the criterion does NOT rest on, recorded because the
    # article is explicit that it is an identity and not the content.
    bad = 0
    for _, S1 in FAMS:
        for _, S2 in FAMS:
            for a1 in arm_words(S1, ARMS2):
                for a2 in arm_words(S2, ARMS2):
                    if (len(pres(S1, a1) & pres_flip(S2, a2))
                            != len(pres_flip(S1, a1) & pres(S2, a2))):
                        bad += 1
    check("the two cross-overlaps are equal, which is an identity",
          bad, 0)

    # Article 84's theorem, reproduced over both alphabets.
    for arms in (ARMS2, ARMS3):
        viol = 0
```

```
        for _, S1 in FAMS:
            for _, S2 in FAMS:
                for a1 in arm_words(S1, arms):
                    for a2 in arm_words(S2, arms):
                        if ((unit_blind(S1, a1) or unit_blind(S2, a2))
                                and sensitive(S1, a1, S2, a2)):
                            viol += 1
        check("one blind unit suffices for a blind contact, %d arm types"
              % len(arms), viol, 0)
    note("the criterion is exact in both directions, so a one-unit test"
         " decides a two-unit property")


# ======================================================================
#  C4.  Why D behaves differently from I and T
# ======================================================================
#
#  Article 85, Prop. "The two families behave differently".  The proof
#  rests on a difference formula, not on the two lemmas the submitted
#  version asserted; both of those are false and the counterexamples are
#  counted here rather than absorbed.

def both_sighted_table(arms):
    tab = {}
    for n1, S1 in FAMS:
        for n2, S2 in FAMS:
            blind = sens = 0
            for a1 in arm_words(S1, arms):
                if unit_blind(S1, a1):
                    continue
                for a2 in arm_words(S2, arms):
                    if unit_blind(S2, a2):
                        continue
                    if sensitive(S1, a1, S2, a2):
                        sens += 1
                    else:
                        blind += 1
            tab[(n1, n2)] = (blind, sens)
    return tab


def block_C4():
    block("C4", "why D behaves differently from I and T")

    tab2 = both_sighted_table(ARMS2)
    tab3 = both_sighted_table(ARMS3)

    for arms, tab in ((ARMS2, tab2), (ARMS3, tab3)):
        check("on D-D two sighted units always give a sensitive contact,"
              " %d arm types" % len(arms), tab[("D", "D")][0], 0)
    check("and there are four such D-D contacts over {h,e}",
          tab2[("D", "D")][1], 4)
    check("on T-T forty of the both-sighted contacts are blind",
          tab2[("T", "T")][0], 40)
    check("and one hundred and four are sensitive",
          tab2[("T", "T")][1], 104)
    tt = tab2[("T", "T")]
    check("so the sensitive share on T-T rounds to 0.722",
          rounds_to(tt[1], tt[0] + tt[1], 722, 1000), True)
```

```
    tt3 = tab3[("T", "T")]
    check("over three arm types the same share rounds to 0.806",
          rounds_to(tt3[1], tt3[0] + tt3[1], 806, 1000), True)
    check("on I-I twenty-four both-sighted contacts are blind, so I"
          " behaves like T and not like D", tab2[("I", "I")][0], 24)
    check("the dichotomy is D against the rest, not D and I against T",
          (tab2[("D", "D")][0] == 0, tab2[("I", "I")][0] > 0,
           tab2[("T", "T")][0] > 0), (True, True, True))

    # The difference formula, and its values.
    for arms, want in ((ARMS2, [-2, 2]), (ARMS3, None)):
        seen = set()
        bad = 0
        for a in arm_words(D, arms):
            if unit_blind(D, a):
                continue
            for b in arm_words(D, arms):
                if unit_blind(D, b):
                    continue
                d = ((a[6] == b[6]) + (a[12] == b[12])
                     - (a[6] == b[12]) - (a[12] == b[6]))
                if d != nu(D, a, D, b, True) - nu(D, a, D, b, False):
                    bad += 1
                seen.add(d)
        check("the difference formula holds on every sighted D-D pair,"
              " %d arm types" % len(arms), bad, 0)
        if want is not None:
            check("and over {h,e} its values are exactly -2 and +2",
                  sorted(seen), want)
        else:
            check("and over three arm types it is still never zero",
                  0 in seen, False)

    # The threshold.  A zero difference needs four letters, so the claim
    # is about alphabets of at most three and is checked at the first
    # alphabet where it fails -- which a test confined to two and three
    # letters could not see.
    thresh = {}
    for m in (2, 3, 4, 5):
        arms = tuple("abcde"[:m])
        nb = tot = 0
        for a in arm_words(D, arms):
            if unit_blind(D, a):
                continue
            for b in arm_words(D, arms):
                if unit_blind(D, b):
                    continue
                tot += 1
                if not sensitive(D, a, D, b):
                    nb += 1
        thresh[m] = (nb, tot)
    check("a sighted D-D contact is blind for no alphabet of two or three"
          " letters and for every larger one",
          [m for m in (2, 3, 4, 5) if thresh[m][0] == 0], [2, 3])
    check("at four letters there are 24 blind pairs of 144",
          thresh[4], (24, 144))
    check("so the proposition is a statement about alphabets of at most"
          " three letters", 3, 3)
```

```
    # The two lemmas the submitted proof asserted, both false.
    fail_a = fail_b = 0
    for _, S2 in FAMS:
        for a in arm_words(D, ARMS3):
            if unit_blind(D, a):
                continue
            for b in arm_words(S2, ARMS3):
                if unit_blind(S2, b):
                    continue
                m = nu(D, a, S2, b, True)
                if m == nu(D, a, S2, b, False):
                    if m > 0:
                        fail_a += 1
                    else:
                        fail_b += 1
    check("the first lemma of the submitted proof has 108 counterexamples",
          fail_a, 108)
    check("and the second has 54", fail_b, 54)
    check("so neither lemma is available and the difference formula is"
          " what carries the proposition",
          (fail_a > 0, fail_b > 0), (True, True))

    # Why: a sighted D pattern is disjoint from its flip; a sighted T
    # pattern need not be.
    disj = sum(1 for a in arm_words(D, ARMS3)
               if not unit_blind(D, a)
               and not (pres(D, a) & pres_flip(D, a)))
    tot = sum(1 for a in arm_words(D, ARMS3) if not unit_blind(D, a))
    check("every sighted D word is disjoint from its flip", disj, tot)
    meet = sum(1 for a in arm_words(T, ARMS2)
               if not unit_blind(T, a) and (pres(T, a) & pres_flip(T, a)))
    sight = sum(1 for a in arm_words(T, ARMS2) if not unit_blind(T, a))
    check("but eight of the twelve sighted T words meet their flips",
          (meet, sight), (8, 12))
    note("D carries one iota-orbit, T carries two, and that is the whole"
         " of the difference")
    return tab2


# ======================================================================
#  C5.  The sensitivity polynomial
# ======================================================================
#
#  Article 85, Thm. "The sensitivity polynomial".  Each arm is cleaved
#  independently with probability pi_h, so the probability that a contact
#  drawn from the admitted stream is orientation-sensitive is a
#  polynomial in pi_h with rational coefficients.  It is DERIVED here by
#  enumeration over the 784 ordered pairs, not restated.

class Poly:
    """Exact polynomial in pi_h, coefficients in increasing powers."""

    __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

    def coef(self, k):
        return self.c[k] if k < len(self.c) else Fraction(0)


PONE = Poly([1])
PH = Poly([0, 1])
PE = 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(arms=ARMS2):
    tot = Poly()
    for n1, S1 in FAMS:
        for n2, S2 in FAMS:
            w = FRAC[n1] * FRAC[n2]
            for a1 in arm_words(S1, arms):
                for a2 in arm_words(S2, arms):
                    if sensitive(S1, a1, S2, a2):
                        tot = tot + (word_poly(S1, a1)
                                     * word_poly(S2, a2)).scale(w)
    return tot
```

```
OPS = ((Fraction(99, 100), "pi_e = 0.99"),
       (Fraction(95, 100), "pi_e = 0.95"),
       (Fraction(90, 100), "pi_e = 0.90"))


def block_C5():
    block("C5", "the sensitivity polynomial, derived by enumeration")

    sig = sensitivity_poly()
    check("sigma has degree exactly eight, two tetrasubstituted units",
          (sig.degree(), sig.coef(8) != 0), (8, True))
    check("and its coefficients are rational, as a sum of products of"
          " rational weights", all(isinstance(c, Fraction) for c in sig.c),
          True)

    check("sigma(0) is exactly f_I^2, which is Article 84's figure",
          sig(0), FI * FI)
    check("so sigma(0) = 1/4044121 and rounds to 2.47e-7",
          (sig(0) == Fraction(1, 4044121),
           sci_rounds(1, 4044121, 247, 100, -7)), (True, True))
    check("the polynomial is a probability on [0,1]: every value tested"
          " lies in the unit interval",
          all(0 <= sig(Fraction(k, 20)) <= 1 for k in range(21)), True)

    c1, c2 = sig.coef(1), sig.coef(2)
    check("the first correction is linear and its coefficient rounds to"
          " 9.40e-6",
          sci_rounds(c1.numerator, c1.denominator, 940, 100, -6), True)
    check("the quadratic coefficient is exactly 16176842/4044121",
          c2, Fraction(16176842, 4044121))
    check("which is 4044121 = 2011^2 in the denominator", 2011 ** 2,
          4044121)
    check("and rounds to 4.00009, not to the integer four",
          (rounds_to(c2.numerator, c2.denominator, 400009, 100000),
           c2 == 4), (True, False))
    check("the linear coefficient is smaller than the quadratic by more"
          " than two orders", c1 * 100 < c2, True)

    x = Fraction(1, 100)
    share = c2 * x * x / sig(x)
    check("at one per cent cleavage the quadratic term alone carries the"
          " value to within fifteen per cent",
          abs(share - 1) < Fraction(15, 100), True)
    check("and the share rounds to 1.02",
          rounds_to(share.numerator, share.denominator, 102, 100),
          True)
    note("sensitivity needs TWO sighted units, so at small pi_h the"
         " cheapest route is one cleaved arm on each: hence quadratic")

    vals = {}
    for pe, lbl in OPS:
        ph = 1 - pe
        v = sig(ph)
        vals[pe] = v
        check("sigma is exactly rational at %s" % lbl,
              isinstance(v, Fraction), True)
    check("sigma at pi_e = 0.90 rounds to 3.24e-2",
          sci_rounds(vals[Fraction(90, 100)].numerator,
                     vals[Fraction(90, 100)].denominator, 324, 100, -2),
```

```
          True)
    check("at pi_e = 0.95 to 9.03e-3",
          sci_rounds(vals[Fraction(95, 100)].numerator,
                     vals[Fraction(95, 100)].denominator, 903, 100, -3),
          True)
    check("at pi_e = 0.99 to 3.92e-4",
          sci_rounds(vals[Fraction(99, 100)].numerator,
                     vals[Fraction(99, 100)].denominator, 392, 100, -4),
          True)

    lifts = {}
    for pe, lbl in OPS:
        lifts[pe] = vals[pe] / sig(0)
    check("the lift over the pure limit is exactly 131037 at pi_e = 0.90",
          rounds_to(lifts[Fraction(90, 100)].numerator,
                    lifts[Fraction(90, 100)].denominator, 131037, 1), True)
    check("36502 at pi_e = 0.95",
          rounds_to(lifts[Fraction(95, 100)].numerator,
                    lifts[Fraction(95, 100)].denominator, 36502, 1), True)
    check("and 1587 at pi_e = 0.99",
          rounds_to(lifts[Fraction(99, 100)].numerator,
                    lifts[Fraction(99, 100)].denominator, 1587, 1), True)
    check("so one per cent cleavage lifts sensitivity by over three"
          " orders", lifts[Fraction(99, 100)] > 1000, True)
    for pe, lbl in OPS:
        show("%s : sigma = %s  (lift %s over the pure limit)"
             % (lbl, expo(vals[pe]), dec(lifts[pe], 0)))
    return sig


# ======================================================================
#  C6.  The registration gap at a sensitive contact
# ======================================================================
#
#  Article 85, Prop. "The registration gap".  A larger gap is a stronger
#  bias, so the gap distribution is what decides how much information a
#  sensitive contact actually carries.

def gap_dist(ph, sensitive_only=True):
    """Exact distribution of |nu_0 - nu_1| over the stream."""
    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 sensitive_only and g == 0:
                        continue
                    p = (word_prob(S1, a1, ph) * word_prob(S2, a2, ph) * w)
                    if p:
                        out[g] = out.get(g, Fraction(0)) + p
    return out


def block_C6(sig):
    block("C6", "the registration gap at a sensitive contact")
```

```
    gaps = set()
    for _, S1 in FAMS:
        for _, S2 in FAMS:
            for a1 in arm_words(S1, ARMS2):
                for a2 in arm_words(S2, ARMS2):
                    if sensitive(S1, a1, S2, a2):
                        gaps.add(abs(nu(S1, a1, S2, a2, True)
                                     - nu(S1, a1, S2, a2, False)))
    check("the gap is never zero on a sensitive contact, by definition",
          0 in gaps, False)
    check("it is at least one and at most four", (min(gaps), max(gaps)),
          (1, 4))
    check("and the attained gaps are exactly 1, 2, 3, 4",
          sorted(gaps), [1, 2, 3, 4])
    check("four is two tetrasubstituted units whose patterns are disjoint"
          " from their flips", len(T), 4)

    d0 = gap_dist(Fraction(0))
    check("at perfect arming every sensitive contact has gap one",
          sorted(d0), [1])
    check("and its mass is exactly f_I^2", d0[1], FI * FI)

    for pe, lbl in OPS:
        d = gap_dist(1 - pe)
        tot = sum(d.values())
        check("the gap distribution is normalised to sigma at %s" % lbl,
              tot, sig(1 - pe))
        s1 = d.get(1, Fraction(0)) / tot
        s2 = d.get(2, Fraction(0)) / tot
        mean = sum(g * p for g, p in d.items()) / tot
        check("the conditional distribution concentrates on gap two at"
              " %s, share above 0.99" % lbl, s2 > Fraction(99, 100), True)
        show("%s : P(g=1|sens) = %s, P(g=2|sens) = %s, mean gap %s"
             % (lbl, dec(s1, 3), dec(s2, 5), dec(mean, 3)))
        if pe == Fraction(90, 100):
            check("at pi_e = 0.90 the gap-two share rounds to 0.99997",
                  rounds_to(s2.numerator, s2.denominator,
                            99997, 100000), True)
            check("the gap-one share rounds to 0.000",
                  rounds_to(s1.numerator, s1.denominator, 0, 1000),
                  True)
            check("the mean gap rounds to 2.000",
                  rounds_to(mean.numerator, mean.denominator,
                            2000, 1000), True)
            for g, m, e in ((1, 332, -5), (3, 546, -7), (4, 401, -7)):
                sh = d.get(g, Fraction(0)) / tot
                check("and the gap-%d share rounds to %d.%02de%d"
                      % (g, m // 100, m % 100, e),
                      sci_rounds(sh.numerator, sh.denominator, m, 100, e),
                      True)
        if pe == Fraction(99, 100):
            check("at pi_e = 0.99 the gap-one share rounds to 0.001",
                  rounds_to(s1.numerator, s1.denominator, 1, 1000),
                  True)
            check("and the mean gap rounds to 1.999",
                  rounds_to(mean.numerator, mean.denominator,
                            1999, 1000), True)
    note("a mean of 2.000 here is a near point mass, not a balance of"
         " small and large gaps, and the article says so")
```

```
    return gap_dist


# ======================================================================
#  C7.  The bias, and the information the source supplies
# ======================================================================
#
#  Article 85, Thm. "The channel".  Under Article 79's H_rgy a contact
#  carrying nu registrations has relative weight e^{nu Delta}, so at a
#  sensitive contact of gap g the two alignments carry weights in the
#  ratio e^{g Delta} and the endogenous law on that contact is
#  Bernoulli(theta) with theta = 1/(1 + e^{g Delta}).
#
#  This is where the article leaves the rationals.  Every figure of this
#  block and of C10 is certified by the enclosures of C1.

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


_INFO = {}


def info_at(g, delta):
    """1 - H(theta(g, delta)), as a certified enclosure."""
    if (g, delta) not in _INFO:
        _INFO[(g, delta)] = IV1 - hbin_iv(theta_iv(g, delta))
    return _INFO[(g, delta)]


def info_per_contact(ph, delta, gd=None):
    """Expected information per contact of the stream."""
    d = (gd or gap_dist)(ph)
    tot = IV0
    for g, p in d.items():
        tot = tot + Iv(p) * info_at(g, delta)
    return tot


def block_C7(sig):
    block("C7", "the bias, and the information the source supplies")

    check("at a blind contact the two alignments carry equal weight, so"
          " theta is exactly one half",
          (IV1 + exp_int(0)).inv().lo == Fraction(1, 2), True)
    check("and a blind contact therefore carries exactly no information",
          hbin_iv(Iv(Fraction(1, 2))).lo <= 1 <= hbin_iv(
              Iv(Fraction(1, 2))).hi, True)
    check("at every positive gap and free energy theta is strictly below"
          " one half",
          all(theta_iv(g, d).hi < Fraction(1, 2)
              for g in (1, 2, 3, 4) for d in (1, 2, 3, 6, 12, 24)), True)
    check("and the information per sensitive contact is strictly between"
          " zero and one",
          all(IV0.hi < info_at(g, d).lo and info_at(g, d).hi < 1
              for g in (1, 2, 3, 4) for d in (1, 2, 3)), True)

    want = {(1, 1): 160, (1, 2): 473, (1, 3): 725,
            (2, 1): 473, (2, 2): 870, (2, 3): 975,
```

```
            (4, 1): 870, (4, 2): 996, (4, 3): 1000}
    for (g, dl), r in sorted(want.items()):
        check("information at gap %d, Delta = %d rounds to 0.%03d"
              % (g, dl, r), rounds_to_iv(info_at(g, dl), r, 1000), True)
    check("information is increasing in the gap at fixed Delta",
          all(info_at(g, 2).hi < info_at(g + 1, 2).lo for g in (1, 2, 3)),
          True)
    check("and increasing in Delta at fixed gap",
          all(info_at(2, d).hi < info_at(2, d + 1).lo for d in (1, 2, 3)),
          True)

    # THE CEILING.  A sensitive contact carries at most one bit and a
    # blind one carries none, so the information per contact is at most
    # sigma, whatever the registration free energy.  This removes the
    # dependence on Delta, which Article 79 nowhere measures.
    #
    # It is NOT checked by resumming the conditional gap distribution,
    # whose total is sigma by construction and which would therefore be
    # circular.  It is checked against the full pair ensemble, in which
    # every contact appears and a blind one must be SHOWN to contribute
    # nothing.
    for pe, lbl in OPS:
        ph = 1 - pe
        full = gap_dist(ph, sensitive_only=False)
        check("the full gap distribution is normalised at %s" % lbl,
              sum(full.values()), Fraction(1))
        check("and its blind mass is exactly 1 - sigma at %s" % lbl,
              full[0], 1 - sig(ph))
        for delta in (1, 2, 3, 6, 12, 24):
            tot = IV0
            for g, p in full.items():
                if g == 0:
                    continue
                tot = tot + Iv(p) * info_at(g, delta)
            if tot.hi > sig(ph):
                check("the information per contact at %s, Delta = %d does"
                      " not exceed the sigma ceiling" % (lbl, delta),
                      False, True)
        check("the information per contact never exceeds sigma at %s,"
              " over six values of Delta" % lbl, True, True)
    near = info_per_contact(Fraction(1, 10), 24)
    check("and the ceiling is approached: at pi_e = 0.90, Delta = 24 the"
          " information is within 10^-9 of sigma",
          sig(Fraction(1, 10)) - near.lo < Fraction(1, 10 ** 9), True)

    want_ipc = {("Pure", 1): (396, 100, -8), ("Pure", 2): (117, 100, -7),
                ("NineNine", 1): (185, 100, -4),
                ("NineNine", 2): (341, 100, -4),
                ("NineFive", 1): (43, 10, -4),
                ("NineFive", 2): (79, 10, -4),
                ("Nine", 1): (153, 10, -4), ("Nine", 2): (282, 10, -4)}
    names = {Fraction(1, 1): "Pure", Fraction(99, 100): "NineNine",
             Fraction(95, 100): "NineFive", Fraction(90, 100): "Nine"}
    for pe in (Fraction(1, 1), Fraction(99, 100), Fraction(95, 100),
               Fraction(90, 100)):
        for delta in (1, 2):
            v = info_per_contact(1 - pe, delta)
            m, c, e = want_ipc[(names[pe], delta)]
            if names[pe] in ("NineFive", "Nine"):
```

```
                # printed as a four-place decimal rather than in
                # scientific form, so the device is the plain one
                check("the information per contact at pi_e = %s, Delta ="
                      " %d rounds to 0.%04d" % (pe, delta, m * 10 // c),
                      rounds_to_iv(v, m * 10 // c, 10000), True)
            else:
                check("the information per contact at pi_e = %s, Delta ="
                      " %d rounds to %s.%02de%d"
                      % (pe, delta, m // 100, m % 100, e),
                      sci_rounds_iv(v, m, c, e), True)
            show("pi_e = %-6s Delta = %d : %s bits per contact"
                 % (pe, delta, expo(v.mid())))
    note("the ceiling is the Delta-free statement, and it is the one the"
         " article rests on")
    return info_per_contact


# ======================================================================
#  C8.  The channel has no preferred direction
# ======================================================================
#
#  Article 85, Thm. "No preferred direction".  This is the block that
#  corrects the submitted version.  Cleavage makes units sighted, so the
#  channel of C7 is open; but the channel has no preferred direction, and
#  therefore the MARGINAL law of the alignment field is unchanged and
#  Article 84's selection cost stands unamended.
#
#  The mechanism: iota fixes D and T setwise but not I.  Relabelling a D
#  or T arm word by a -> a.iota is a bijection of that family's words,
#  measure-preserving because the cleavage law is independent and
#  identical across arms, and it reverses the presented pattern.

def relabel(S, a):
    return {k: a[IOTA[k]] for k in sorted(S)}


def field_law(tys, ell, r):
    """Exact marginal law of the alignment field on a column of ell
    sites, r standing for e^Delta and taken rational so that the
    arithmetic is exact."""
    n = len(tys)
    W = {}
    for i, (_, S1, a1, _) in enumerate(tys):
        for j, (_, S2, a2, _) in enumerate(tys):
            n0 = nu(S1, a1, S2, a2, True)
            n1 = nu(S1, a1, S2, a2, False)
            w0, w1 = Fraction(r) ** n0, Fraction(r) ** n1
            z = w0 + w1
            W[(i, j)] = (w0 / z, w1 / z)
    P = [t[3] for t in tys]
    law = {}
    for c in product((0, 1), repeat=ell - 1):
        v = [Fraction(1)] * n
        for b in reversed(c):
            v = [sum(P[j] * W[(i, j)][b] * v[j] for j in range(n))
                 for i in range(n)]
        law[c] = sum(P[i] * v[i] for i in range(n))
    return law
```

```
def block_C8(sig):
    block("C8", "the channel has no preferred direction")

    # (a) The relabelling, on the families where it is defined.
    for n, S in FAMS:
        closed = all(IOTA[k] in S for k in S)
        check("iota closes %s: %s" % (n, closed), closed, n in ("D", "T"))
    bad_measure = bad_pattern = tested = 0
    for ph in (Fraction(1, 10), Fraction(1, 2), Fraction(3, 7)):
        for n, S in FAMS:
            if not all(IOTA[k] in S for k in S):
                continue
            for a in arm_words(S, ARMS2):
                tested += 1
                ai = relabel(S, a)
                if word_prob(S, a, ph) != word_prob(S, ai, ph):
                    bad_measure += 1
                if pres(S, ai) != pres_flip(S, a):
                    bad_pattern += 1
    check("the relabelling is measure-preserving on D and T, at every"
          " cleavage level tested", bad_measure, 0)
    check("and it reverses the presented pattern", bad_pattern, 0)
    check("it is a bijection of each family's words",
          all(sorted(tuple(sorted(relabel(S, a).items()))
                     for a in arm_words(S, ARMS2))
              == sorted(tuple(sorted(a.items()))
                        for a in arm_words(S, ARMS2))
              for _, S in (("D", D), ("T", T))), True)
    check("and an involution",
          all(relabel(S, relabel(S, a)) == a
              for _, S in (("D", D), ("T", T))
              for a in arm_words(S, ARMS2)), True)
    check("on I it is not even defined, iota I being {6,9,12}",
          frozenset(IOTA[k] for k in I_), frozenset({9, 12, 6}))

    # (b) Hence the signed bias of the stream vanishes exactly when the
    # initiator is absent, and is of order f_I^2 when it is present.
    for pe, lbl in ((Fraction(90, 100), "pi_e = 0.90"),
                    (Fraction(95, 100), "pi_e = 0.95")):
        ph = 1 - pe
        for with_I in (False, True):
            tys = stream(ph, with_I)
            sgn = Fraction(0)
            for _, S1, a1, p1 in tys:
                for _, S2, a2, p2 in tys:
                    n0 = nu(S1, a1, S2, a2, True)
                    n1 = nu(S1, a1, S2, a2, False)
                    if n0 != n1:
                        sgn += p1 * p2 * (n0 - n1)
            if not with_I:
                check("without the initiator the signed bias is EXACTLY"
                      " zero at %s" % lbl, sgn, Fraction(0))
            else:
                check("with the initiator it is nonzero but below f_I^2"
                      " at %s" % lbl,
                      (sgn != 0, abs(sgn) < FI * FI), (True, True))
                if pe == Fraction(90, 100):
                    check("and rounds to 2.03e-7",
```

```
                        sci_rounds(abs(sgn).numerator,
                                   abs(sgn).denominator, 203, 100, -7),
                        True)

    # (c) The marginal law of the field is exactly uniform without the
    # initiator, at every cleavage level and every registration free
    # energy tested.  By transfer matrix over the unit types, exactly.
    worst_I = Fraction(0)
    for ph in (Fraction(1, 10), Fraction(1, 2)):
        for r in (3, 10):
            for ell in (3, 4):
                for with_I in (False, True):
                    tys = stream(ph, with_I)
                    law = field_law(tys, ell, r)
                    tgt = Fraction(1, 2 ** (ell - 1))
                    if sum(law.values()) != 1:
                        check("the field law is normalised", False, True)
                    dev = max(abs(p - tgt) for p in law.values())
                    if with_I:
                        worst_I = max(worst_I, dev)
                    elif dev != 0:
                        check("the field law is exactly uniform without"
                              " the initiator", False, True)
    check("without the initiator the field law is EXACTLY uniform, at"
          " every pi_h, Delta and column length tested", True, True)
    check("with the initiator it departs from uniform by at most"
          " 8.06e-8",
          sci_rounds(worst_I.numerator, worst_I.denominator, 806, 100, -8),
          True)

    # (d) So Article 84's selection cost is unamended: the cost of a
    # prescribed field is exactly ell - 1 bits at every pi_h, not merely
    # in the limit pi_h = 0.
    bad = 0
    for ell in (3, 4):
        law = field_law(stream(Fraction(1, 10), False), ell, 3)
        for c, p in law.items():
            if p != Fraction(1, 2 ** (ell - 1)):
                bad += 1
    check("the cost of a prescribed field is exactly ell - 1 bits at"
          " pi_h > 0, Article 84's figure unamended", bad, 0)

    # (e) The tolerance with the initiator present, in the scale-free
    # quantity.  An ABSOLUTE deviation of the field mass bounds nothing
    # at ell = 54, the uniform mass itself shrinking with length; the
    # scale-free quantity is the per-contact deviation of the COST.
    worst_bit = IV0
    for ph in (Fraction(1, 10), Fraction(1, 2)):
        for r in (3, 10, 100, 1000):
            for ell in (3, 4, 5):
                law = field_law(stream(ph, True), ell, r)
                for p in law.values():
                    # |-log2 p - (ell-1)| / (ell-1), as an enclosure
                    d = -log2_iv(p) - Iv(ell - 1)
                    m = Iv(max(abs(d.lo), abs(d.hi))) / Iv(ell - 1)
                    if m.hi > worst_bit.hi:
                        worst_bit = m
    check("the cost of a prescribed field is (ell-1)(1 +/- 2.83e-7) bits,"
          " uniformly in ell, pi_h and Delta",
```

```
          sci_rounds_iv(worst_bit, 283, 100, -7), True)
    check("which is one part in 3.54e6",
          sci_rounds_iv(worst_bit.inv(), 354, 100, 6), True)
    show("worst per-contact cost deviation %s, i.e. one part in %s"
         % (expo(worst_bit.mid()), expo(worst_bit.inv().mid())))
    note("the channel is open and carries no direction: that is the"
         " correction this block makes to the submitted version")


# ======================================================================
#  C9.  The trade against Article 79's arm efficiency
# ======================================================================
#
#  Article 79, Prop. "Cleavage": the accepted fractions are 1 - pi_h^2 on
#  D and pi_e^4 on T, so the arm efficiency after re-matching is
#       eta = (rho* + 1) / (rho*/(1 - pi_h^2) + 1/pi_e^4).
#  Cleavage buys sensitivity and spends efficiency; the block shows the
#  trade is a genuine trade and not a free lunch.

def eta_arm(ph, rho=RHO):
    ph = Fraction(ph)
    if ph == 0:
        return Fraction(1)
    return (Fraction(rho) + 1) / (Fraction(rho) / (1 - ph * ph)
                                  + 1 / (1 - ph) ** 4)


def block_C9(sig, ipc):
    block("C9", "the trade against Article 79's arm efficiency")

    check("arm efficiency is exactly one at perfect arming",
          eta_arm(0), Fraction(1))
    etas = {}
    for pe, lbl in OPS:
        e = eta_arm(1 - pe)
        etas[pe] = e
        check("arm efficiency is exactly rational at %s" % lbl,
              isinstance(e, Fraction), True)
    for pe, r in ((Fraction(99, 100), 9997), (Fraction(95, 100), 9964),
                  (Fraction(90, 100), 9875)):
        check("it rounds to 0.%04d at pi_e = %s" % (r, pe),
              rounds_to(etas[pe].numerator, etas[pe].denominator,
                        r, 10000), True)

    # Monotonicity, by certificate rather than by grid.  A grid is not an
    # argument about a degree-eight polynomial; the Bernstein
    # coefficients of sigma' on [0, 1/5] are all positive, and a
    # polynomial whose Bernstein coefficients on an interval are positive
    # is positive there, the basis functions being non-negative.
    b = Fraction(1, 5)
    dc = [sig.coef(k) * k for k in range(1, sig.degree() + 1)]
    q = [dc[k] * b ** k for k in range(len(dc))]
    n = len(q) - 1

    def binom(a, k):
        r = Fraction(1)
        for i in range(k):
            r = r * (a - i) / (i + 1)
        return r
```

```
    bern = []
    for j in range(n + 1):
        s = Fraction(0)
        for k in range(j + 1):
            s += binom(j, k) / binom(n, k) * q[k]
        bern.append(s)
    check("every Bernstein coefficient of sigma' on [0, 1/5] is positive,"
          " so sigma is strictly increasing there",
          all(c > 0 for c in bern), True)
    check("there are eight of them, sigma' having degree seven",
          len(bern), 8)
    check("the least is sigma'(0), which rounds to 9.40e-6",
          sci_rounds(min(bern).numerator, min(bern).denominator,
                     940, 100, -6), True)
    check("and the least equals the linear coefficient of sigma, as it"
          " must", min(bern), sig.coef(1))

    check("both accepted fractions are strictly positive on [0, 1/5]",
          all(1 - p * p > 0 and (1 - p) ** 4 > 0
              for p in (Fraction(0), Fraction(1, 10), b)), True)
    phs = [Fraction(i, 100) for i in range(0, 21)]
    es = [eta_arm(p) for p in phs]
    check("arm efficiency is strictly decreasing in pi_h over [0, 1/5],"
          " both accepted fractions being strictly decreasing there",
          all(es[i] > es[i + 1] for i in range(len(es) - 1)), True)
    check("so sensitivity rises and efficiency falls over the same range:"
          " the trade is genuine",
          (sig(0) < sig(Fraction(1, 10)), eta_arm(0) > eta_arm(
              Fraction(1, 10))), (True, True))

    base = {d: ipc(Fraction(0), d) for d in (1, 2)}
    want = {(Fraction(90, 100), 1): 123, (Fraction(90, 100), 2): 226,
            (Fraction(95, 100), 1): 118, (Fraction(95, 100), 2): 217,
            (Fraction(99, 100), 1): 61, (Fraction(99, 100), 2): 112}
    for pe, lbl in OPS:
        for delta in (1, 2):
            gained = ipc(1 - pe, delta) - base[delta]
            lost = Iv(1 - etas[pe])
            ratio = gained / lost
            r = want[(pe, delta)]
            check("the trade at %s, Delta = %d rounds to %d.%02d bits per"
                  " unit of efficiency" % (lbl, delta, r // 100, r % 100),
                  rounds_to_iv(ratio, r, 100), True)
            show("%s Delta = %d : %s bits gained per unit efficiency lost"
                 % (lbl, delta, dec(ratio.mid(), 2)))
    note("the trade is worst at the lightest cleavage, which is the"
         " opposite of what a linear reading would suggest")
    return eta_arm


# ====================================================================
#  C10.  The capacity of the channel, and what it is not
# ====================================================================
#
#  Article 84, Thm. "The registry term": a prescribed alignment field on
#  a column of ell sites with blind contacts costs exactly ell - 1 bits.
#  By C8 that figure is unamended at every pi_h.  What cleavage opens is
#  a CHANNEL, and this block computes its capacity: the mutual
```

```
#  information between the alignment field and the configuration of
#  units, which is what an agent that can read arm words would save over
#  one that cannot.

def block_C10(sig, ipc):
    block("C10", "the capacity of the channel, and what it is not")

    for ell, nm in ((6, "six"), (54, "fifty-four")):
        check("a column of %s sites has %d contacts" % (nm, ell - 1),
              ell - 1, ell - 1)
        check("and the unconditional cost of a prescribed field is"
              " exactly that, to four places",
              rounds_to(ell - 1, 1, (ell - 1) * 10000, 10000), True)

    want_mi = {
        (6, "Pure", 1): ("sci", 198, 100, -7),
        (6, "Pure", 2): ("sci", 585, 100, -7),
        (6, "NineFive", 1): ("dec", 213, 10000),
        (6, "NineFive", 2): ("dec", 393, 10000),
        (6, "Nine", 1): ("dec", 766, 10000),
        (6, "Nine", 2): ("dec", 1409, 10000),
        (54, "Pure", 1): ("sci", 210, 100, -6),
        (54, "Pure", 2): ("sci", 620, 100, -6),
        (54, "NineFive", 1): ("dec", 2262, 10000),
        (54, "NineFive", 2): ("dec", 4162, 10000),
        (54, "Nine", 1): ("dec", 8122, 10000),
        (54, "Nine", 2): ("dec", 14941, 10000),
    }
    want_cond = {
        (6, "Pure", 1): 50000, (6, "Pure", 2): 50000,
        (6, "NineFive", 1): 49787, (6, "NineFive", 2): 49607,
        (6, "Nine", 1): 49234, (6, "Nine", 2): 48591,
        (54, "Pure", 1): 530000, (54, "Pure", 2): 530000,
        (54, "NineFive", 1): 527738, (54, "NineFive", 2): 525838,
        (54, "Nine", 1): 521878, (54, "Nine", 2): 515059,
    }
    nm = {Fraction(1, 1): "Pure", Fraction(95, 100): "NineFive",
          Fraction(90, 100): "Nine"}
    for ell in (6, 54):
        for pe in (Fraction(1, 1), Fraction(95, 100), Fraction(90, 100)):
            for delta in (1, 2):
                mi = Iv(ell - 1) * ipc(1 - pe, delta)
                cond = Iv(ell - 1) - mi
                spec = want_mi[(ell, nm[pe], delta)]
                if spec[0] == "sci":
                    ok = sci_rounds_iv(mi, spec[1], spec[2], spec[3])
                else:
                    ok = rounds_to_iv(mi, spec[1], spec[2])
                check("the mutual information at ell = %d, pi_e = %s,"
                      " Delta = %d matches the printed figure"
                      % (ell, pe, delta), ok, True)
                c = want_cond[(ell, nm[pe], delta)]
                check("and the conditional entropy rounds to %s"
                      % dec(Fraction(c, 10000), 4),
                      rounds_to_iv(cond, c, 10000), True)
            show("ell = %-2d pi_e = %-6s I(Delta=1) = %s, I(Delta=2) = %s"
                 % (ell, pe, dec((Iv(ell - 1) * ipc(1 - pe, 1)).mid(), 4),
                    dec((Iv(ell - 1) * ipc(1 - pe, 2)).mid(), 4)))
```

```
    # The ceiling, carried up to the column: the capacity is at most
    # (ell-1) sigma for EVERY Delta, which is the Delta-free statement.
    for pe, lbl in OPS:
        ph = 1 - pe
        cap = 5 * sig(ph)
        over = 0
        for delta in (1, 2, 3, 6, 12, 24):
            if (Iv(5) * ipc(ph, delta)).hi > cap:
                over += 1
        check("the capacity of a six-site column never exceeds 5 sigma at"
              " %s, over six values of Delta" % lbl, over, 0)
    for pe, r in ((Fraction(90, 100), 1620), (Fraction(95, 100), 451),
                  (Fraction(99, 100), 20)):
        cap = 5 * sig(1 - pe)
        check("and 5 sigma rounds to 0.%04d at pi_e = %s" % (r, pe),
              rounds_to(cap.numerator, cap.denominator, r, 10000),
              True)

    # What the submitted version called a reduced cost is the conditional
    # entropy, and the surprisal RANGE belongs with it: the conditional
    # cost of a PARTICULAR prescribed field is field-dependent and can
    # exceed ell - 1.
    want_s = {(1, 1): (452, 1895), (1, 2): (183, 3069),
              (2, 1): (183, 3069), (2, 2): (26, 5797)}
    for (g, delta), (lo, hi) in sorted(want_s.items()):
        th = theta_iv(g, delta)
        high = -log2_of_iv(th)
        low = -log2_of_iv(IV1 - th)
        check("the surprisal at gap %d, Delta = %d runs from %s to %s"
              " bits" % (g, delta, dec(Fraction(lo, 1000), 3),
                         dec(Fraction(hi, 1000), 3)),
              (rounds_to_iv(low, lo, 1000),
               rounds_to_iv(high, hi, 1000)), (True, True))
    check("so the high end exceeds one bit, and a particular prescribed"
          " field can cost more than ell - 1",
          all((-log2_of_iv(theta_iv(g, d))).lo > 1
              for g in (1, 2) for d in (1, 2)), True)

    # And the unconditional form, which needs no unlucky field.  Averaged
    # over a uniformly chosen prescribed field the conditional cost is
    # sum_j (1/2)(-log2 theta_j - log2(1 - theta_j)) >= ell - 1 by Gibbs,
    # with equality only at Delta = 0.  The inequality is algebraic:
    # theta(1 - theta) <= 1/4 for every theta, so the bracketed term is
    # at least two.
    for pe, lbl in ((Fraction(95, 100), "pi_e = 0.95"),
                    (Fraction(90, 100), "pi_e = 0.90")):
        ph = 1 - pe
        full = gap_dist(ph, sensitive_only=False)
        for delta, r in ((1, 50282 if pe == Fraction(95, 100) else 51014),
                         (2, 50863 if pe == Fraction(95, 100) else 53097)):
            acc = IV0
            for g, p in full.items():
                th = theta_iv(g, delta) if g else Iv(Fraction(1, 2))
                term = -log2_of_iv(th) - log2_of_iv(IV1 - th)
                acc = acc + Iv(p) * Iv(Fraction(1, 2)) * term
            tot = Iv(5) * acc
            check("the Gibbs mean of the conditional cost at %s, Delta ="
                  " %d is at least five" % (lbl, delta), tot.lo >= 5, True)
            check("and rounds to %s" % dec(Fraction(r, 10000), 4),
```

```
                    rounds_to_iv(tot, r, 10000), True)
    check("the Gibbs inequality is algebraic: 4 theta (1 - theta) <= 1"
          " for every rational theta tested",
          all(4 * Fraction(a, 97) * (1 - Fraction(a, 97)) <= 1
              for a in range(1, 97)), True)
    note("so the conditional cost EXCEEDS Article 84's figure on average,"
         " and not only for an unlucky field")


# ======================================================================
#  C11.  How much of H_frame cleavage refines
# ======================================================================
#
#  Article 84's H_frame asks for an iota-free reference, because at a
#  blind contact the two alignments are one configuration described
#  twice.  At a SENSITIVE contact they are not: the two carry different
#  registration energies, so the contact distinguishes its two states
#  without any external frame.  It does not LABEL them, so what is
#  established is a refinement of H_frame and not its discharge.
#
#  The contacts of a column are NOT independent: contact i and contact
#  i+1 share unit i+1, and sensitivity needs both units sighted, so the
#  indicators are positively correlated.  Every distributional figure is
#  computed by transfer matrix, exactly.

def blind_matrix_int(ph):
    """(N, den, B) with integer numerators: P[j] = N[j]/den, and B the
    blind-contact adjacency.  Working in integers keeps the transfer
    matrix exact out to column lengths of a few thousand, where the
    denominators of a Fraction implementation would serve no purpose."""
    tys = stream(ph)
    den = 1
    for _, _, _, p in tys:
        den = den * p.denominator // _gcd(den, p.denominator)
    N = [p.numerator * (den // p.denominator) for _, _, _, p in tys]
    n = len(tys)
    B = [[0] * n for _ in range(n)]
    for i, (_, S1, a1, _) in enumerate(tys):
        for j, (_, S2, a2, _) in enumerate(tys):
            B[i][j] = 0 if sensitive(S1, a1, S2, a2) else 1
    return N, den, B


def _gcd(a, b):
    while b:
        a, b = b, a % b
    return a


def p_no_sensitive_seq(ph, upto):
    """Yield (ell, exact P(no contact of an ell-site column is
    sensitive)) for ell = 2 .. upto, in one pass."""
    N, den, B = blind_matrix_int(ph)
    n = len(N)
    u = [1] * n
    denpow = 1
    for ell in range(2, upto + 1):
        u = [sum(N[j] * u[j] for j in range(n) if B[i][j]) for i in range(n)]
        denpow *= den
```

```
        yield ell, Fraction(sum(N[i] * u[i] for i in range(n)),
                            denpow * den)


def p_no_sensitive(ph, ell):
    for e, v in p_no_sensitive_seq(ph, ell):
        if e == ell:
            return v
    raise AssertionError


def block_C11(sig):
    block("C11", "how much of H_frame cleavage refines")

    # (a) The dependence, measured against the binomial prediction.
    for pe, lbl, r in ((Fraction(99, 100), "pi_e = 0.99", 504),
                       (Fraction(95, 100), "pi_e = 0.95", 105),
                       (Fraction(90, 100), "pi_e = 0.90", 56)):
        ph = 1 - pe
        tys = stream(ph)
        both = Fraction(0)
        for _, S1, a1, p1 in tys:
            for _, S2, a2, p2 in tys:
                if not sensitive(S1, a1, S2, a2):
                    continue
                for _, S3, a3, p3 in tys:
                    if sensitive(S2, a2, S3, a3):
                        both += p1 * p2 * p3
        s = sig(ph)
        ratio = both / (s * s)
        check("adjacent contacts are positively correlated at %s" % lbl,
              ratio > 1, True)
        check("and the ratio to the binomial figure rounds to %s"
              % dec(Fraction(r, 10), 1),
              rounds_to(ratio.numerator, ratio.denominator, r, 10),
              True)
    note("so the binomial count does not apply, and every distributional"
         " figure below is a transfer-matrix computation")

    # (b) P(at least one sensitive contact), exactly, against the
    # binomial figure the submitted version reported.
    want = {(6, Fraction(1, 1)): ("sci", 124, 100, -6),
            (6, Fraction(99, 100)): ("dec", 19, 10000),
            (6, Fraction(95, 100)): ("dec", 415, 10000),
            (6, Fraction(90, 100)): ("dec", 1363, 10000),
            (54, Fraction(1, 1)): ("sci", 131, 100, -5),
            (54, Fraction(99, 100)): ("dec", 202, 10000),
            (54, Fraction(95, 100)): ("dec", 3577, 10000),
            (54, Fraction(90, 100)): ("dec", 7804, 10000)}
    for ell in (6, 54):
        for pe in (Fraction(1, 1), Fraction(99, 100), Fraction(95, 100),
                   Fraction(90, 100)):
            ph = 1 - pe
            p_any = 1 - p_no_sensitive(ph, ell)
            binom = 1 - (1 - sig(ph)) ** (ell - 1)
            check("the exact figure never exceeds the binomial one at"
                  " ell = %d, pi_e = %s" % (ell, pe), p_any <= binom, True)
            spec = want[(ell, pe)]
            if spec[0] == "sci":
```

```
                ok = sci_rounds(p_any.numerator, p_any.denominator,
                                spec[1], spec[2], spec[3])
            else:
                ok = rounds_to(p_any.numerator,
                               p_any.denominator, spec[1], spec[2])
            check("and the exact figure matches the printed one",
                  ok, True)
        show("ell = %-2d : P(at least one sensitive contact) = %s"
             % (ell, ", ".join(
                 expo(1 - p_no_sensitive(1 - pe, ell))
                 for pe in (Fraction(1, 1), Fraction(99, 100),
                            Fraction(95, 100), Fraction(90, 100)))))
    b54 = 1 - (1 - sig(Fraction(1, 10))) ** 53
    check("at ell = 54, pi_e = 0.90 the binomial figure is 0.8255 against"
          " the exact 0.7804, so the dependence is not a rounding effect",
          rounds_to(b54.numerator, b54.denominator, 8255, 10000),
          True)

    # (c) The distinction that matters.  A sensitive contact is
    # intrinsically oriented; a blind one still needs the reference.  So
    # the FRACTION of a column's contacts on which H_frame is discharged
    # is sigma, not the probability that the column carries one.
    for ell, nm in ((6, "six"), (54, "fifty-four")):
        for pe, r in ((Fraction(99, 100), 2 if ell == 6 else 21),
                      (Fraction(95, 100), 45 if ell == 6 else 478),
                      (Fraction(90, 100), 162 if ell == 6 else 1717)):
            m = (ell - 1) * sig(1 - pe)
            check("the expected count of intrinsically oriented contacts"
                  " at ell = %d, pi_e = %s rounds to %s"
                  % (ell, pe, dec(Fraction(r, 1000), 3)),
                  rounds_to(m.numerator, m.denominator, r, 1000),
                  True)
    for pe, r in ((Fraction(99, 100), 9996), (Fraction(95, 100), 9910),
                  (Fraction(90, 100), 9676)):
        still = 100 * (1 - sig(1 - pe))
        check("the share of contacts still needing the reference at"
              " pi_e = %s rounds to %s per cent"
              % (pe, dec(Fraction(r, 100), 2)),
              rounds_to(still.numerator, still.denominator, r, 100),
              True)
    check("so even at the heaviest cleavage over ninety-six per cent of"
          " contacts still need the reference: a refinement of H_frame"
          " and not its discharge",
          100 * (1 - sig(Fraction(1, 10))) > 96, True)

    # (d) The length at which a column is more likely than not to carry a
    # sensitive contact, from the exact law, with the binomial figure
    # beside it.
    for pe, half, nb in ((Fraction(90, 100), 26, 23),
                         (Fraction(95, 100), 85, 78),
                         (Fraction(99, 100), 1802, 1768)):
        ph = 1 - pe
        got = None
        for ell, pno in p_no_sensitive_seq(ph, half + 1):
            if 1 - pno >= Fraction(1, 2):
                got = ell
                break
        check("half of columns carry a sensitive contact by %d sites at"
              " pi_e = %s, exactly" % (half, pe), got, half)
```

```
        s = sig(ph)
        k = 1
        while 1 - (1 - s) ** k < Fraction(1, 2):
            k += 1
        check("against the binomial figure of %d sites" % nb, k + 1, nb)
        show("pi_e = %-6s half-length %d sites (binomial %d)"
             % (pe, got, k + 1))
    check("the binomial figure understates the length at every operating"
          " point, the dependence working against the agent", True, True)

    # (e) The lattice length.  Article 80's 54 is the NODE COUNT of a
    # 3x3x6 box, not a column length: such a box is nine columns of six
    # along the stacking axis, hence 45 stacking contacts, not 53.
    check("a 3x3x6 box has 54 nodes", 3 * 3 * 6, 54)
    check("it is nine columns of six along the stacking axis",
          (3 * 3, 6), (9, 6))
    check("so it carries 45 stacking contacts", 9 * (6 - 1), 45)
    check("and not the 53 a single column of 54 sites would carry",
          54 - 1, 53)
    check("the two figures are different objects and the article says"
          " which is which", 45 == 53, False)


# ======================================================================
#  C12.  Scope, obligations and the review record
# ======================================================================
#
#  These checks assert the bookkeeping of the article, not its
#  mathematics.  Each is marked RECORDED: the harness confirms that the
#  article's own counts are internally consistent, and nothing more.

def block_C12():
    block("C12", "scope, obligations and the review record")

    hyps = ["H_cleave", "H_rgy", "H_nu", "H_frame", "H_draw"]
    check("five named hypotheses are carried", len(hyps), 5)
    check("and they are distinct", len(set(hyps)), 5)
    corrections = ["the direction of the channel",
                   "the two lemmas of the D/T proof",
                   "the dependence of adjacent contacts",
                   "the conditional entropy, named correctly",
                   "the box contact count"]
    check("five corrections are recorded rather than absorbed",
          len(corrections), 5)
    note("RECORDED, not verified: that Delta is nowhere measured in the"
         " corpus, which is why the ceiling is the operative statement")
    note("RECORDED, not verified: that cleavage is a chemical process"
         " this article does not propose a mechanism for")
    note("the data of this article are definitions of the corpus, so"
         " there is no external datum for a harness to be unable to"
         " reach; what it cannot reach is the value of Delta")


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

def main():
    print("verify_A85.py  --  Annals of the Chrysene Formalism, Article 85")
```

```
    print("The Partially Sighted Source")
    print("=" * 66)

    block_C1()
    block_C2()
    block_C3()
    block_C4()
    sig = block_C5()
    gd = block_C6(sig)
    ipc = block_C7(sig)
    block_C8(sig)
    block_C9(sig, ipc)
    block_C10(sig, ipc)
    block_C11(sig)
    block_C12()

    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_A85.py**

```
verify_A85.py  --  Annals of the Chrysene Formalism, Article 85
The Partially Sighted Source
================================================================

C1  the devices: integer brackets and certified enclosures
    e      = 2.71828182845904523536  (width 1.67e-100)
    ln 2   = 0.69314718055994530942  (width 1.05e-118)

C2  the stream: units, arm words and the admitted fractions

C3  the separation criterion, and its symmetry

C4  why D behaves differently from I and T

C5  the sensitivity polynomial, derived by enumeration
    pi_e = 0.99 : sigma = 3.92e-04  (lift 1587 over the pure limit)
    pi_e = 0.95 : sigma = 9.03e-03  (lift 36502 over the pure limit)
    pi_e = 0.90 : sigma = 3.24e-02  (lift 131037 over the pure limit)

C6  the registration gap at a sensitive contact
    pi_e = 0.99 : P(g=1|sens) = 0.001, P(g=2|sens) = 0.99913, mean gap 1.999
    pi_e = 0.95 : P(g=1|sens) = 0.000, P(g=2|sens) = 0.99992, mean gap 2.000
    pi_e = 0.90 : P(g=1|sens) = 0.000, P(g=2|sens) = 0.99997, mean gap 2.000

C7  the bias, and the information the source supplies
    pi_e = 1      Delta = 1 : 3.96e-08 bits per contact
    pi_e = 1      Delta = 2 : 1.17e-07 bits per contact
    pi_e = 99/100 Delta = 1 : 1.85e-04 bits per contact
    pi_e = 99/100 Delta = 2 : 3.41e-04 bits per contact
    pi_e = 19/20  Delta = 1 : 4.27e-03 bits per contact
    pi_e = 19/20  Delta = 2 : 7.85e-03 bits per contact
    pi_e = 9/10   Delta = 1 : 1.53e-02 bits per contact
    pi_e = 9/10   Delta = 2 : 2.82e-02 bits per contact

C8  the channel has no preferred direction
    worst per-contact cost deviation 2.83e-07, i.e. one part in 3.54e+06

C9  the trade against Article 79's arm efficiency
    pi_e = 0.99 Delta = 1 : 0.61 bits gained per unit efficiency lost
    pi_e = 0.99 Delta = 2 : 1.12 bits gained per unit efficiency lost
    pi_e = 0.95 Delta = 1 : 1.18 bits gained per unit efficiency lost
    pi_e = 0.95 Delta = 2 : 2.17 bits gained per unit efficiency lost
    pi_e = 0.90 Delta = 1 : 1.23 bits gained per unit efficiency lost
    pi_e = 0.90 Delta = 2 : 2.26 bits gained per unit efficiency lost

C10  the capacity of the channel, and what it is not
    ell = 6  pi_e = 1      I(Delta=1) = 0.0000, I(Delta=2) = 0.0000
    ell = 6  pi_e = 19/20  I(Delta=1) = 0.0213, I(Delta=2) = 0.0393
    ell = 6  pi_e = 9/10   I(Delta=1) = 0.0766, I(Delta=2) = 0.1409
    ell = 54 pi_e = 1      I(Delta=1) = 0.0000, I(Delta=2) = 0.0000
    ell = 54 pi_e = 19/20  I(Delta=1) = 0.2262, I(Delta=2) = 0.4162
    ell = 54 pi_e = 9/10   I(Delta=1) = 0.8122, I(Delta=2) = 1.4941

C11  how much of H_frame cleavage refines
```

```
    ell = 6  : P(at least one sensitive contact) = 1.24e-06, 1.93e-03, 4.15e
        -02, 1.36e-01
    ell = 54 : P(at least one sensitive contact) = 1.31e-05, 2.02e-02, 3.58e
        -01, 7.80e-01
    pi_e = 9/10   half-length 26 sites (binomial 23)
    pi_e = 19/20  half-length 85 sites (binomial 78)
    pi_e = 99/100 half-length 1802 sites (binomial 1768)

C12  scope, obligations and the review record

================================================================
checks run     : 283
passed         : 283
failed         : 0
floating point: none
grid           : outward rounding onto multiples of 2^-400
================================================================
```

## Console Transcript: Verbose Run (-v)

**$ python3 verify_A85.py -v**

```
verify_A85.py  --  Annals of the Chrysene Formalism, Article 85
The Partially Sighted Source
================================================================

C1  the devices: integer brackets and certified enclosures
    [C1-01] PASS  the rounding bound accepts a correct rounding
    [C1-02] PASS  and rejects the next value down
    [C1-03] PASS  and rejects the next value up
    [C1-04] PASS  the digit count accepts 1000 at four digits
    [C1-05] PASS  and rejects three
    [C1-06] PASS  the scientific bracket accepts 1/2011 = 4.97e-4
    [C1-07] PASS  and rejects 4.98e-4
    [C1-08] PASS  it puts a positive exponent into the value, not the target
    [C1-09] PASS  every interval contains its own midpoint
    [C1-10] PASS  e lies strictly inside the decimal bracket of its first
       fifteen places
    [C1-11] PASS  ln 2 lies strictly inside the bracket of its first fifteen
    [C1-12] PASS  the enclosures of e and of ln 2 are narrower than 10^-80
    [C1-13] PASS  log2(2^1) encloses 1 exactly
    [C1-14] PASS  log2(2^3) encloses 3 exactly
    [C1-15] PASS  log2(2^10) encloses 10 exactly
    [C1-16] PASS  log2(2^20) encloses 20 exactly
    [C1-17] PASS  log2(10) encloses 3.3219280948873623
    [C1-18] PASS  log2 is strictly increasing on the arguments tested
    [C1-19] PASS  e^3 is enclosed by e * e * e
    [C1-20] PASS  e^12 is bracketed by the two decimals 162754.7914190039 and
       162754.7914190040
    [C1-21] PASS  H(1/2) encloses one exactly
    [C1-22] PASS  H(1/4) is bracketed by the two decimals of 2 - (3/4) log2 3
    [C1-23] PASS  H is symmetric about one half, to within the enclosures
    [C1-24] PASS  the enclosure device accepts the printed 0.160
    [C1-25] PASS  and rejects 0.161
    [C1-26] PASS  and rejects 0.159
    [C1-27] PASS  an enclosure too wide to decide a rounding is rejected
    e      = 2.71828182845904523536  (width 1.67e-100)
    ln 2   = 0.69314718055994530942  (width 1.05e-118)
    ---          every transcendental figure below is certified by an
       enclosure of this kind, never by a floating-point comparison

C2  the stream: units, arm words and the admitted fractions
    [C2-01] PASS  iota is an involution of the four locants
    [C2-02] PASS  iota fixes D and T setwise but not I
    [C2-03] PASS  the admitted fractions sum to one exactly
    [C2-04] PASS  f_D, f_I, f_T are 2000/2011, 1/2011, 10/2011
    [C2-05] PASS  f_I rounds to 4.97e-4
    [C2-06] PASS  f_I squared is exactly 1/2011^2 and rounds to 2.47e-7
    [C2-07] PASS  the arm words over {h,e} number 4 on D, 8 on I, 16 on T
    [C2-08] PASS  so the stream has twenty-eight unit types
    [C2-09] PASS  and nine over a single arm letter
    [C2-10] PASS  the stream is normalised exactly at pi_h = 0
    [C2-11] PASS  the stream is normalised exactly at pi_h = 1/100
    [C2-12] PASS  the stream is normalised exactly at pi_h = 1/20
    [C2-13] PASS  the stream is normalised exactly at pi_h = 1/10
    [C2-14] PASS  the stream is normalised exactly at pi_h = 1/2
```

```
    [C2-15] PASS  and with the initiator removed it is still normalised
    [C2-16] PASS  removing the initiator leaves twenty unit types
    [C2-17] PASS  at perfect arming only the all-e word of each family
       survives
    ---           the alphabet is {h, e}: an e_P arm is downstream of admission
56
C3  the separation criterion, and its symmetry
    [C3-01] PASS  the criterion agrees with sensitivity on all 784 ordered
       pairs over 2 arm types
    [C3-02] PASS  and the pair count over {h,e} is 784
    [C3-03] PASS  the criterion agrees with sensitivity on all 13689 ordered
       pairs over 3 arm types
61  [C3-04] PASS  and over three arm types it is 13,689
    [C3-05] PASS  sensitivity is symmetric in the two units, 2 arm types
    [C3-06] PASS  sensitivity is symmetric in the two units, 3 arm types
    [C3-07] PASS  the two cross-overlaps are equal, which is an identity
    [C3-08] PASS  one blind unit suffices for a blind contact, 2 arm types
66  [C3-09] PASS  one blind unit suffices for a blind contact, 3 arm types
    ---           the criterion is exact in both directions, so a one-unit test
        decides a two-unit property

C4  why D behaves differently from I and T
    [C4-01] PASS  on D-D two sighted units always give a sensitive contact, 2
       arm types
71  [C4-02] PASS  on D-D two sighted units always give a sensitive contact, 3
       arm types
    [C4-03] PASS  and there are four such D-D contacts over {h,e}
    [C4-04] PASS  on T-T forty of the both-sighted contacts are blind
    [C4-05] PASS  and one hundred and four are sensitive
    [C4-06] PASS  so the sensitive share on T-T rounds to 0.722
76  [C4-07] PASS  over three arm types the same share rounds to 0.806
    [C4-08] PASS  on I-I twenty-four both-sighted contacts are blind, so I
       behaves like T and not like D
    [C4-09] PASS  the dichotomy is D against the rest, not D and I against T
    [C4-10] PASS  the difference formula holds on every sighted D-D pair, 2
       arm types
    [C4-11] PASS  and over {h,e} its values are exactly -2 and +2
81  [C4-12] PASS  the difference formula holds on every sighted D-D pair, 3
       arm types
    [C4-13] PASS  and over three arm types it is still never zero
    [C4-14] PASS  a sighted D-D contact is blind for no alphabet of two or
       three letters and for every larger one
    [C4-15] PASS  at four letters there are 24 blind pairs of 144
    [C4-16] PASS  so the proposition is a statement about alphabets of at most
        three letters
86  [C4-17] PASS  the first lemma of the submitted proof has 108
       counterexamples
    [C4-18] PASS  and the second has 54
    [C4-19] PASS  so neither lemma is available and the difference formula is
       what carries the proposition
    [C4-20] PASS  every sighted D word is disjoint from its flip
    [C4-21] PASS  but eight of the twelve sighted T words meet their flips
91  ---           D carries one iota-orbit, T carries two, and that is the
       whole of the difference

C5  the sensitivity polynomial, derived by enumeration
    [C5-01] PASS  sigma has degree exactly eight, two tetrasubstituted units
    [C5-02] PASS  and its coefficients are rational, as a sum of products of
       rational weights
```

```
    [C5-03] PASS  sigma(0) is exactly f_I^2, which is Article 84's figure
    [C5-04] PASS  so sigma(0) = 1/4044121 and rounds to 2.47e-7
    [C5-05] PASS  the polynomial is a probability on [0,1]: every value tested
        lies in the unit interval
    [C5-06] PASS  the first correction is linear and its coefficient rounds to
        9.40e-6
    [C5-07] PASS  the quadratic coefficient is exactly 16176842/4044121
    [C5-08] PASS  which is 4044121 = 2011^2 in the denominator
    [C5-09] PASS  and rounds to 4.00009, not to the integer four
    [C5-10] PASS  the linear coefficient is smaller than the quadratic by more
        than two orders
    [C5-11] PASS  at one per cent cleavage the quadratic term alone carries
       the value to within fifteen per cent
    [C5-12] PASS  and the share rounds to 1.02
    ---           sensitivity needs TWO sighted units, so at small pi_h the
       cheapest route is one cleaved arm on each: hence quadratic
    [C5-13] PASS  sigma is exactly rational at pi_e = 0.99
    [C5-14] PASS  sigma is exactly rational at pi_e = 0.95
    [C5-15] PASS  sigma is exactly rational at pi_e = 0.90
    [C5-16] PASS  sigma at pi_e = 0.90 rounds to 3.24e-2
    [C5-17] PASS  at pi_e = 0.95 to 9.03e-3
    [C5-18] PASS  at pi_e = 0.99 to 3.92e-4
    [C5-19] PASS  the lift over the pure limit is exactly 131037 at pi_e =
       0.90
    [C5-20] PASS  36502 at pi_e = 0.95
    [C5-21] PASS  and 1587 at pi_e = 0.99
    [C5-22] PASS  so one per cent cleavage lifts sensitivity by over three
       orders
    pi_e = 0.99 : sigma = 3.92e-04  (lift 1587 over the pure limit)
    pi_e = 0.95 : sigma = 9.03e-03  (lift 36502 over the pure limit)
    pi_e = 0.90 : sigma = 3.24e-02  (lift 131037 over the pure limit)

C6  the registration gap at a sensitive contact
    [C6-01] PASS  the gap is never zero on a sensitive contact, by definition
    [C6-02] PASS  it is at least one and at most four
    [C6-03] PASS  and the attained gaps are exactly 1, 2, 3, 4
    [C6-04] PASS  four is two tetrasubstituted units whose patterns are
       disjoint from their flips
    [C6-05] PASS  at perfect arming every sensitive contact has gap one
    [C6-06] PASS  and its mass is exactly f_I^2
    [C6-07] PASS  the gap distribution is normalised to sigma at pi_e = 0.99
    [C6-08] PASS  the conditional distribution concentrates on gap two at pi_e
        = 0.99, share above 0.99
    pi_e = 0.99 : P(g=1|sens) = 0.001, P(g=2|sens) = 0.99913, mean gap 1.999
    [C6-09] PASS  at pi_e = 0.99 the gap-one share rounds to 0.001
    [C6-10] PASS  and the mean gap rounds to 1.999
    [C6-11] PASS  the gap distribution is normalised to sigma at pi_e = 0.95
    [C6-12] PASS  the conditional distribution concentrates on gap two at pi_e
        = 0.95, share above 0.99
    pi_e = 0.95 : P(g=1|sens) = 0.000, P(g=2|sens) = 0.99992, mean gap 2.000
    [C6-13] PASS  the gap distribution is normalised to sigma at pi_e = 0.90
    [C6-14] PASS  the conditional distribution concentrates on gap two at pi_e
        = 0.90, share above 0.99
    pi_e = 0.90 : P(g=1|sens) = 0.000, P(g=2|sens) = 0.99997, mean gap 2.000
    [C6-15] PASS  at pi_e = 0.90 the gap-two share rounds to 0.99997
    [C6-16] PASS  the gap-one share rounds to 0.000
    [C6-17] PASS  the mean gap rounds to 2.000
    [C6-18] PASS  and the gap-1 share rounds to 3.32e-5
    [C6-19] PASS  and the gap-3 share rounds to 5.46e-7
```

```
    [C6-20] PASS  and the gap-4 share rounds to 4.01e-7
    ---          a mean of 2.000 here is a near point mass, not a balance of
       small and large gaps, and the article says so

C7  the bias, and the information the source supplies
    [C7-01] PASS  at a blind contact the two alignments carry equal weight, so
        theta is exactly one half
    [C7-02] PASS  and a blind contact therefore carries exactly no information
    [C7-03] PASS  at every positive gap and free energy theta is strictly
       below one half
    [C7-04] PASS  and the information per sensitive contact is strictly
       between zero and one
    [C7-05] PASS  information at gap 1, Delta = 1 rounds to 0.160
    [C7-06] PASS  information at gap 1, Delta = 2 rounds to 0.473
    [C7-07] PASS  information at gap 1, Delta = 3 rounds to 0.725
    [C7-08] PASS  information at gap 2, Delta = 1 rounds to 0.473
    [C7-09] PASS  information at gap 2, Delta = 2 rounds to 0.870
    [C7-10] PASS  information at gap 2, Delta = 3 rounds to 0.975
    [C7-11] PASS  information at gap 4, Delta = 1 rounds to 0.870
    [C7-12] PASS  information at gap 4, Delta = 2 rounds to 0.996
    [C7-13] PASS  information at gap 4, Delta = 3 rounds to 0.1000
    [C7-14] PASS  information is increasing in the gap at fixed Delta
    [C7-15] PASS  and increasing in Delta at fixed gap
    [C7-16] PASS  the full gap distribution is normalised at pi_e = 0.99
    [C7-17] PASS  and its blind mass is exactly 1 - sigma at pi_e = 0.99
    [C7-18] PASS  the information per contact never exceeds sigma at pi_e =
       0.99, over six values of Delta
    [C7-19] PASS  the full gap distribution is normalised at pi_e = 0.95
    [C7-20] PASS  and its blind mass is exactly 1 - sigma at pi_e = 0.95
    [C7-21] PASS  the information per contact never exceeds sigma at pi_e =
       0.95, over six values of Delta
    [C7-22] PASS  the full gap distribution is normalised at pi_e = 0.90
    [C7-23] PASS  and its blind mass is exactly 1 - sigma at pi_e = 0.90
    [C7-24] PASS  the information per contact never exceeds sigma at pi_e =
       0.90, over six values of Delta
    [C7-25] PASS  and the ceiling is approached: at pi_e = 0.90, Delta = 24
       the information is within 10^-9 of sigma
    [C7-26] PASS  the information per contact at pi_e = 1, Delta = 1 rounds to
        3.96e-8
    pi_e = 1      Delta = 1 : 3.96e-08 bits per contact
    [C7-27] PASS  the information per contact at pi_e = 1, Delta = 2 rounds to
        1.17e-7
    pi_e = 1      Delta = 2 : 1.17e-07 bits per contact
    [C7-28] PASS  the information per contact at pi_e = 99/100, Delta = 1
       rounds to 1.85e-4
    pi_e = 99/100 Delta = 1 : 1.85e-04 bits per contact
    [C7-29] PASS  the information per contact at pi_e = 99/100, Delta = 2
       rounds to 3.41e-4
    pi_e = 99/100 Delta = 2 : 3.41e-04 bits per contact
    [C7-30] PASS  the information per contact at pi_e = 19/20, Delta = 1
       rounds to 0.0043
    pi_e = 19/20  Delta = 1 : 4.27e-03 bits per contact
    [C7-31] PASS  the information per contact at pi_e = 19/20, Delta = 2
       rounds to 0.0079
    pi_e = 19/20  Delta = 2 : 7.85e-03 bits per contact
    [C7-32] PASS  the information per contact at pi_e = 9/10, Delta = 1 rounds
        to 0.0153
    pi_e = 9/10   Delta = 1 : 1.53e-02 bits per contact
    [C7-33] PASS  the information per contact at pi_e = 9/10, Delta = 2 rounds
```

```
        to 0.0282
    pi_e = 9/10   Delta = 2 : 2.82e-02 bits per contact
    ---          the ceiling is the Delta-free statement, and it is the one
       the article rests on

C8  the channel has no preferred direction
    [C8-01] PASS  iota closes D: True
    [C8-02] PASS  iota closes I: False
    [C8-03] PASS  iota closes T: True
    [C8-04] PASS  the relabelling is measure-preserving on D and T, at every
       cleavage level tested
    [C8-05] PASS  and it reverses the presented pattern
    [C8-06] PASS  it is a bijection of each family's words
    [C8-07] PASS  and an involution
    [C8-08] PASS  on I it is not even defined, iota I being {6,9,12}
    [C8-09] PASS  without the initiator the signed bias is EXACTLY zero at
       pi_e = 0.90
    [C8-10] PASS  with the initiator it is nonzero but below f_I^2 at pi_e =
       0.90
    [C8-11] PASS  and rounds to 2.03e-7
    [C8-12] PASS  without the initiator the signed bias is EXACTLY zero at
       pi_e = 0.95
    [C8-13] PASS  with the initiator it is nonzero but below f_I^2 at pi_e =
       0.95
    [C8-14] PASS  without the initiator the field law is EXACTLY uniform, at
       every pi_h, Delta and column length tested
    [C8-15] PASS  with the initiator it departs from uniform by at most 8.06e
       -8
    [C8-16] PASS  the cost of a prescribed field is exactly ell - 1 bits at
       pi_h > 0, Article 84's figure unamended
    [C8-17] PASS  the cost of a prescribed field is (ell-1)(1 +/- 2.83e-7)
       bits, uniformly in ell, pi_h and Delta
    [C8-18] PASS  which is one part in 3.54e6
    worst per-contact cost deviation 2.83e-07, i.e. one part in 3.54e+06
    ---          the channel is open and carries no direction: that is the
       correction this block makes to the submitted version

C9  the trade against Article 79's arm efficiency
    [C9-01] PASS  arm efficiency is exactly one at perfect arming
    [C9-02] PASS  arm efficiency is exactly rational at pi_e = 0.99
    [C9-03] PASS  arm efficiency is exactly rational at pi_e = 0.95
    [C9-04] PASS  arm efficiency is exactly rational at pi_e = 0.90
    [C9-05] PASS  it rounds to 0.9997 at pi_e = 99/100
    [C9-06] PASS  it rounds to 0.9964 at pi_e = 19/20
    [C9-07] PASS  it rounds to 0.9875 at pi_e = 9/10
    [C9-08] PASS  every Bernstein coefficient of sigma' on [0, 1/5] is
       positive, so sigma is strictly increasing there
    [C9-09] PASS  there are eight of them, sigma' having degree seven
    [C9-10] PASS  the least is sigma'(0), which rounds to 9.40e-6
    [C9-11] PASS  and the least equals the linear coefficient of sigma, as it
       must
    [C9-12] PASS  both accepted fractions are strictly positive on [0, 1/5]
    [C9-13] PASS  arm efficiency is strictly decreasing in pi_h over [0, 1/5],
        both accepted fractions being strictly decreasing there
    [C9-14] PASS  so sensitivity rises and efficiency falls over the same
       range: the trade is genuine
    [C9-15] PASS  the trade at pi_e = 0.99, Delta = 1 rounds to 0.61 bits per
       unit of efficiency
    pi_e = 0.99 Delta = 1 : 0.61 bits gained per unit efficiency lost
```

```
    [C9-16] PASS  the trade at pi_e = 0.99, Delta = 2 rounds to 1.12 bits per
        unit of efficiency
    pi_e = 0.99 Delta = 2 : 1.12 bits gained per unit efficiency lost
    [C9-17] PASS  the trade at pi_e = 0.95, Delta = 1 rounds to 1.18 bits per
        unit of efficiency
    pi_e = 0.95 Delta = 1 : 1.18 bits gained per unit efficiency lost
    [C9-18] PASS  the trade at pi_e = 0.95, Delta = 2 rounds to 2.17 bits per
        unit of efficiency
    pi_e = 0.95 Delta = 2 : 2.17 bits gained per unit efficiency lost
    [C9-19] PASS  the trade at pi_e = 0.90, Delta = 1 rounds to 1.23 bits per
        unit of efficiency
    pi_e = 0.90 Delta = 1 : 1.23 bits gained per unit efficiency lost
    [C9-20] PASS  the trade at pi_e = 0.90, Delta = 2 rounds to 2.26 bits per
        unit of efficiency
    pi_e = 0.90 Delta = 2 : 2.26 bits gained per unit efficiency lost
    ---          the trade is worst at the lightest cleavage, which is the
        opposite of what a linear reading would suggest

C10  the capacity of the channel, and what it is not
    [C10-01] PASS  a column of six sites has 5 contacts
    [C10-02] PASS  and the unconditional cost of a prescribed field is exactly
         that, to four places
    [C10-03] PASS  a column of fifty-four sites has 53 contacts
    [C10-04] PASS  and the unconditional cost of a prescribed field is exactly
         that, to four places
    [C10-05] PASS  the mutual information at ell = 6, pi_e = 1, Delta = 1
        matches the printed figure
    [C10-06] PASS  and the conditional entropy rounds to 5.0000
    [C10-07] PASS  the mutual information at ell = 6, pi_e = 1, Delta = 2
        matches the printed figure
    [C10-08] PASS  and the conditional entropy rounds to 5.0000
    ell = 6  pi_e = 1      I(Delta=1) = 0.0000, I(Delta=2) = 0.0000
    [C10-09] PASS  the mutual information at ell = 6, pi_e = 19/20, Delta = 1
        matches the printed figure
    [C10-10] PASS  and the conditional entropy rounds to 4.9787
    [C10-11] PASS  the mutual information at ell = 6, pi_e = 19/20, Delta = 2
        matches the printed figure
    [C10-12] PASS  and the conditional entropy rounds to 4.9607
    ell = 6  pi_e = 19/20  I(Delta=1) = 0.0213, I(Delta=2) = 0.0393
    [C10-13] PASS  the mutual information at ell = 6, pi_e = 9/10, Delta = 1
        matches the printed figure
    [C10-14] PASS  and the conditional entropy rounds to 4.9234
    [C10-15] PASS  the mutual information at ell = 6, pi_e = 9/10, Delta = 2
        matches the printed figure
    [C10-16] PASS  and the conditional entropy rounds to 4.8591
    ell = 6  pi_e = 9/10   I(Delta=1) = 0.0766, I(Delta=2) = 0.1409
    [C10-17] PASS  the mutual information at ell = 54, pi_e = 1, Delta = 1
        matches the printed figure
    [C10-18] PASS  and the conditional entropy rounds to 53.0000
    [C10-19] PASS  the mutual information at ell = 54, pi_e = 1, Delta = 2
        matches the printed figure
    [C10-20] PASS  and the conditional entropy rounds to 53.0000
    ell = 54 pi_e = 1      I(Delta=1) = 0.0000, I(Delta=2) = 0.0000
    [C10-21] PASS  the mutual information at ell = 54, pi_e = 19/20, Delta = 1
         matches the printed figure
    [C10-22] PASS  and the conditional entropy rounds to 52.7738
    [C10-23] PASS  the mutual information at ell = 54, pi_e = 19/20, Delta = 2
         matches the printed figure
    [C10-24] PASS  and the conditional entropy rounds to 52.5838
```

```
    ell = 54 pi_e = 19/20  I(Delta=1) = 0.2262, I(Delta=2) = 0.4162
    [C10-25] PASS  the mutual information at ell = 54, pi_e = 9/10, Delta = 1
       matches the printed figure
    [C10-26] PASS  and the conditional entropy rounds to 52.1878
    [C10-27] PASS  the mutual information at ell = 54, pi_e = 9/10, Delta = 2
       matches the printed figure
    [C10-28] PASS  and the conditional entropy rounds to 51.5059
    ell = 54 pi_e = 9/10   I(Delta=1) = 0.8122, I(Delta=2) = 1.4941
    [C10-29] PASS  the capacity of a six-site column never exceeds 5 sigma at
       pi_e = 0.99, over six values of Delta
    [C10-30] PASS  the capacity of a six-site column never exceeds 5 sigma at
       pi_e = 0.95, over six values of Delta
    [C10-31] PASS  the capacity of a six-site column never exceeds 5 sigma at
       pi_e = 0.90, over six values of Delta
    [C10-32] PASS  and 5 sigma rounds to 0.1620 at pi_e = 9/10
    [C10-33] PASS  and 5 sigma rounds to 0.0451 at pi_e = 19/20
    [C10-34] PASS  and 5 sigma rounds to 0.0020 at pi_e = 99/100
    [C10-35] PASS  the surprisal at gap 1, Delta = 1 runs from 0.452 to 1.895
       bits
    [C10-36] PASS  the surprisal at gap 1, Delta = 2 runs from 0.183 to 3.069
       bits
    [C10-37] PASS  the surprisal at gap 2, Delta = 1 runs from 0.183 to 3.069
       bits
    [C10-38] PASS  the surprisal at gap 2, Delta = 2 runs from 0.026 to 5.797
       bits
    [C10-39] PASS  so the high end exceeds one bit, and a particular
       prescribed field can cost more than ell - 1
    [C10-40] PASS  the Gibbs mean of the conditional cost at pi_e = 0.95,
       Delta = 1 is at least five
    [C10-41] PASS  and rounds to 5.0282
    [C10-42] PASS  the Gibbs mean of the conditional cost at pi_e = 0.95,
       Delta = 2 is at least five
    [C10-43] PASS  and rounds to 5.0863
    [C10-44] PASS  the Gibbs mean of the conditional cost at pi_e = 0.90,
       Delta = 1 is at least five
    [C10-45] PASS  and rounds to 5.1014
    [C10-46] PASS  the Gibbs mean of the conditional cost at pi_e = 0.90,
       Delta = 2 is at least five
    [C10-47] PASS  and rounds to 5.3097
    [C10-48] PASS  the Gibbs inequality is algebraic: 4 theta (1 - theta) <= 1
        for every rational theta tested
    ---          so the conditional cost EXCEEDS Article 84's figure on
       average, and not only for an unlucky field

C11  how much of H_frame cleavage refines
    [C11-01] PASS  adjacent contacts are positively correlated at pi_e = 0.99
    [C11-02] PASS  and the ratio to the binomial figure rounds to 50.4
    [C11-03] PASS  adjacent contacts are positively correlated at pi_e = 0.95
    [C11-04] PASS  and the ratio to the binomial figure rounds to 10.5
    [C11-05] PASS  adjacent contacts are positively correlated at pi_e = 0.90
    [C11-06] PASS  and the ratio to the binomial figure rounds to 5.6
    ---          so the binomial count does not apply, and every
       distributional figure below is a transfer-matrix computation
    [C11-07] PASS  the exact figure never exceeds the binomial one at ell = 6,
        pi_e = 1
    [C11-08] PASS  and the exact figure matches the printed one
    [C11-09] PASS  the exact figure never exceeds the binomial one at ell = 6,
        pi_e = 99/100
    [C11-10] PASS  and the exact figure matches the printed one
```

```
    [C11-11] PASS  the exact figure never exceeds the binomial one at ell = 6,
        pi_e = 19/20
    [C11-12] PASS  and the exact figure matches the printed one
    [C11-13] PASS  the exact figure never exceeds the binomial one at ell = 6,
        pi_e = 9/10
    [C11-14] PASS  and the exact figure matches the printed one
    ell = 6  : P(at least one sensitive contact) = 1.24e-06, 1.93e-03, 4.15e
        -02, 1.36e-01
    [C11-15] PASS  the exact figure never exceeds the binomial one at ell =
        54, pi_e = 1
    [C11-16] PASS  and the exact figure matches the printed one
    [C11-17] PASS  the exact figure never exceeds the binomial one at ell =
        54, pi_e = 99/100
    [C11-18] PASS  and the exact figure matches the printed one
    [C11-19] PASS  the exact figure never exceeds the binomial one at ell =
        54, pi_e = 19/20
    [C11-20] PASS  and the exact figure matches the printed one
    [C11-21] PASS  the exact figure never exceeds the binomial one at ell =
        54, pi_e = 9/10
    [C11-22] PASS  and the exact figure matches the printed one
    ell = 54 : P(at least one sensitive contact) = 1.31e-05, 2.02e-02, 3.58e
        -01, 7.80e-01
    [C11-23] PASS  at ell = 54, pi_e = 0.90 the binomial figure is 0.8255
        against the exact 0.7804, so the dependence is not a rounding effect
    [C11-24] PASS  the expected count of intrinsically oriented contacts at
        ell = 6, pi_e = 99/100 rounds to 0.002
    [C11-25] PASS  the expected count of intrinsically oriented contacts at
        ell = 6, pi_e = 19/20 rounds to 0.045
    [C11-26] PASS  the expected count of intrinsically oriented contacts at
        ell = 6, pi_e = 9/10 rounds to 0.162
    [C11-27] PASS  the expected count of intrinsically oriented contacts at
        ell = 54, pi_e = 99/100 rounds to 0.021
    [C11-28] PASS  the expected count of intrinsically oriented contacts at
        ell = 54, pi_e = 19/20 rounds to 0.478
    [C11-29] PASS  the expected count of intrinsically oriented contacts at
        ell = 54, pi_e = 9/10 rounds to 1.717
    [C11-30] PASS  the share of contacts still needing the reference at pi_e =
        99/100 rounds to 99.96 per cent
    [C11-31] PASS  the share of contacts still needing the reference at pi_e =
        19/20 rounds to 99.10 per cent
    [C11-32] PASS  the share of contacts still needing the reference at pi_e =
        9/10 rounds to 96.76 per cent
    [C11-33] PASS  so even at the heaviest cleavage over ninety-six per cent
        of contacts still need the reference: a refinement of H_frame and not
        its discharge
    [C11-34] PASS  half of columns carry a sensitive contact by 26 sites at
        pi_e = 9/10, exactly
    [C11-35] PASS  against the binomial figure of 23 sites
    pi_e = 9/10   half-length 26 sites (binomial 23)
    [C11-36] PASS  half of columns carry a sensitive contact by 85 sites at
        pi_e = 19/20, exactly
    [C11-37] PASS  against the binomial figure of 78 sites
    pi_e = 19/20  half-length 85 sites (binomial 78)
    [C11-38] PASS  half of columns carry a sensitive contact by 1802 sites at
        pi_e = 99/100, exactly
    [C11-39] PASS  against the binomial figure of 1768 sites
    pi_e = 99/100 half-length 1802 sites (binomial 1768)
    [C11-40] PASS  the binomial figure understates the length at every
        operating point, the dependence working against the agent
```

```
    [C11-41] PASS  a 3x3x6 box has 54 nodes
    [C11-42] PASS  it is nine columns of six along the stacking axis
    [C11-43] PASS  so it carries 45 stacking contacts
    [C11-44] PASS  and not the 53 a single column of 54 sites would carry
    [C11-45] PASS  the two figures are different objects and the article says
       which is which

C12  scope, obligations and the review record
    [C12-01] PASS  five named hypotheses are carried
    [C12-02] PASS  and they are distinct
    [C12-03] PASS  five corrections are recorded rather than absorbed
    ---          RECORDED, not verified: that Delta is nowhere measured in the
        corpus, which is why the ceiling is the operative statement
    ---          RECORDED, not verified: that cleavage is a chemical process
       this article does not propose a mechanism for
    ---          the data of this article are definitions of the corpus, so
       there is no external datum for a harness to be unable to reach; what
       it cannot reach is the value of Delta

================================================================
checks run    : 283
passed        : 283
failed        : 0
floating point: none
grid          : outward rounding onto multiples of 2^-400
================================================================
```

## Independent Cross-Check: `crosscheck_A85.py`

**The published figures, against the exact values and the enclosures**

Article 85 emits its own figures from a separate generator, `verify_cleave.py`, which is exact in the rationals through its blocks Y1–Y3 and Y5 but computes the entropies and exponentials of blocks Y4–Y7 in *floating point* — it imports `math.log2` and `math.exp` — and writes 153 LaTeX macros into `verification_values.tex`. Those macros are what the article prints. `crosscheck_A85.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, a transcendental against its certified enclosure, and the two symbolic macros — the stipulated ratio $p_3/p_4$ and the exact quadratic coefficient $16\,176\,842/4\,044\,121$ — against strings the harness derives. **All 153 of the 153 agree, with no disagreement in any place.**

This is the check that makes the enclosure machinery worth having. The two computations share no arithmetic: one is floating point over `math.log2` and `math.exp`, the other exact integers over `Fraction` with outward-rounded intervals built from series with explicit remainder bounds. Agreement in all 153 places is evidence that the published figures are right, and not merely that they are self-consistent.

It is also the check that would have caught a floating-point figure that had drifted in its last printed place, and that is not a hypothetical worry for this article: nineteen of its macros are four-place decimals computed from an exponential, and a four-place decimal is exactly the precision at which double arithmetic begins to be a question rather than an answer.

**$ python3 crosscheck_A85.py**

```
crosscheck_A85.py  --  exact values and certified enclosures against
                       the figures Article 85 prints
reading ../verification_values.tex  (153 macros)
======================================================================
macros read    : 153
compared       : 153
agree          : 153
disagree       : 0
======================================================================
```

**`crosscheck_A85.py` — complete source**

```
#!/usr/bin/env python3
# ======================================================================
#  crosscheck_A85.py
#
#  Independent cross-check of the figures Article 85 prints against the
#  exact values and certified enclosures of verify_A85.py.
#
#  Article 85 emits its own figures from a separate generator,
#  verify_cleave.py, which computes the entropies and exponentials of its
#  blocks Y4 to Y7 in FLOATING POINT -- it imports math.log2 and math.exp
#  and formats with %.4f and %.2e -- and writes the result as LaTeX
```

```
#  macros into verification_values.tex.  Those macros are what the
#  article prints.
#
#  This script re-derives the same quantities exactly, or inside a
#  certified rational enclosure, and checks the printed figure against
#  the certificate by the device appropriate to it.  The two
#  computations share no arithmetic, so agreement is evidence that the
#  published figures are right and not merely that they are
#  self-consistent.
#
#  Usage:  python3 crosscheck_A85.py [path/to/verification_values.tex]
# ======================================================================

from fractions import Fraction
import re
import sys

sys.path.insert(0, ".")
import verify_A85 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 = []
SYMBOLIC = []


# ----------------------------------------------------------------------
#  Parsing the printed form, so that the device matches the precision
#  the article actually prints.
# ----------------------------------------------------------------------
def parse(s):
    s = s.strip()
    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+)\}$", s)
    if m:
        whole, frac, e = m.group(1), m.group(2) or "", int(m.group(3))
        scale = 10 ** len(frac)
        return ("sci", int(whole) * scale + int(frac or 0), scale, e)
    m = re.match(r"^(\d+)/(\d+)$", s)
    if m:
        return ("frac", int(m.group(1)), int(m.group(2)))
    return ("sym", s)
```

```
71
def cmp_exact(name, value):
    """Check a macro against an exact Fraction (or int)."""
    global AGREE
    if name not in macros:
76      SKIPPED.append((name, "absent"))
        return
    spec = parse(macros[name])
    x = Fraction(value)
    if spec[0] == "dec":
81      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])
    elif spec[0] == "frac":
86      ok = (x == Fraction(spec[1], spec[2]))
    else:
        SYMBOLIC.append(name)
        return
    if ok:
91      AGREE += 1
    else:
        DISAGREE.append((name, macros[name], str(x)[:40]))


96  def cmp_iv(name, value):
    """Check a macro against a certified enclosure."""
    global AGREE
    if name not in macros:
        SKIPPED.append((name, "absent"))
101     return
    spec = parse(macros[name])
    w = V.iv(value)
    if spec[0] == "dec":
        ok = V.rounds_to_iv(w, spec[1], spec[2])
106 elif spec[0] == "sci":
        ok = V.sci_rounds_iv(w, spec[1], spec[2], spec[3])
    else:
        SYMBOLIC.append(name)
        return
111 if ok:
        AGREE += 1
    else:
        DISAGREE.append((name, macros[name], V.dec(w.mid(), 8)))


116
def cmp_sym(name, want):
    """Check a macro that carries a fixed symbol rather than a number."""
    global AGREE
    if name not in macros:
121     SKIPPED.append((name, "absent"))
        return
    if macros[name] == want:
        AGREE += 1
    else:
126     DISAGREE.append((name, macros[name], want))


# ---------------------------------------------------------------------
```

```
def main():
    print("crosscheck_A85.py  --  exact values and certified enclosures"
          " against")
    print("                       the figures Article 85 prints")
    print("reading %s  (%d macros)" % (PATH, len(macros)))
    print("=" * 66)

    F = Fraction
    PE = {"NineNine": F(99, 100), "NineFive": F(95, 100), "Nine": F(90, 100)}
    DN = {"One": 1, "Two": 2, "Three": 3}

    # ---- the stipulated parameters --------------------------------
    cmp_exact("VrhoStar", V.RHO)
    cmp_sym("VpThreeOverFour", "1/10")

    # ---- C3, C4: the counts ---------------------------------------
    n = 0
    for _, S1 in V.FAMS:
        for _, S2 in V.FAMS:
            n += len(V.arm_words(S1, V.ARMS2)) * len(V.arm_words(S2, V.ARMS2))
    cmp_exact("VpairsChecked", n)

    tab2 = V.both_sighted_table(V.ARMS2)
    tab3 = V.both_sighted_table(V.ARMS3)
    cmp_exact("VddSens", tab2[("D", "D")][1])
    cmp_exact("VttBlind", tab2[("T", "T")][0])
    cmp_exact("VttSens", tab2[("T", "T")][1])
    tt = tab2[("T", "T")]
    cmp_exact("VttSensFrac", F(tt[1], tt[0] + tt[1]))
    tt3 = tab3[("T", "T")]
    cmp_exact("VttSensFracThree", F(tt3[1], tt3[0] + tt3[1]))
    cmp_exact("VIIblind", tab2[("I", "I")][0])

    nb = tot = 0
    for a in V.arm_words(V.D, ("a", "b", "c", "d")):
        if V.unit_blind(V.D, a):
            continue
        for b in V.arm_words(V.D, ("a", "b", "c", "d")):
            if V.unit_blind(V.D, b):
                continue
            tot += 1
            if not V.sensitive(V.D, a, V.D, b):
                nb += 1
    cmp_exact("VddBlindFour", nb)
    cmp_exact("VddPairsFour", tot)
    cmp_exact("VddAlphaMax", 3)

    fail_a = fail_b = 0
    for _, S2 in V.FAMS:
        for a in V.arm_words(V.D, V.ARMS3):
            if V.unit_blind(V.D, a):
                continue
            for b in V.arm_words(S2, V.ARMS3):
                if V.unit_blind(S2, b):
                    continue
                m = V.nu(V.D, a, S2, b, True)
                if m == V.nu(V.D, a, S2, b, False):
                    if m > 0:
                        fail_a += 1
```

```
                    else:
                        fail_b += 1
    cmp_exact("VlemmaFailA", fail_a)
    cmp_exact("VlemmaFailB", fail_b)

    meet = sum(1 for a in V.arm_words(V.T, V.ARMS2)
               if not V.unit_blind(V.T, a)
               and (V.pres(V.T, a) & V.pres_flip(V.T, a)))
    sight = sum(1 for a in V.arm_words(V.T, V.ARMS2)
                if not V.unit_blind(V.T, a))
    cmp_exact("VtMeet", meet)
    cmp_exact("VtSighted", sight)

    # ---- C5: the polynomial ------------------------------------------
    sig = V.sensitivity_poly()
    cmp_exact("VsigDegree", sig.degree())
    cmp_exact("VsigZero", sig(0))
    cmp_exact("VfracI", V.FI)
    cmp_exact("VfIsq", V.FI * V.FI)
    cmp_exact("VsigLinCoef", sig.coef(1))
    cmp_exact("VsigQuadCoef", sig.coef(2))
    cmp_sym("VsigQuadExact", "%d/%d" % (sig.coef(2).numerator,
                                        sig.coef(2).denominator))
    x = F(1, 100)
    cmp_exact("VquadShare", sig.coef(2) * x * x / sig(x))
    for nm, pe in PE.items():
        cmp_exact("Vsigma" + nm, sig(1 - pe))
        cmp_exact("Vceil" + nm, sig(1 - pe))
        cmp_exact("Vlift" + nm, sig(1 - pe) / sig(0))

    # ---- C6: the gap -------------------------------------------------
    cmp_exact("VgapMin", 1)
    cmp_exact("VgapMax", 4)
    for nm, pe in PE.items():
        d = V.gap_dist(1 - pe)
        t = sum(d.values())
        cmp_exact("VgapOneShare" + nm, d.get(1, F(0)) / t)
        cmp_exact("VgapTwoShare" + nm, d.get(2, F(0)) / t)
        cmp_exact("VgapMean" + nm, sum(g * p for g, p in d.items()) / t)
    d9 = V.gap_dist(F(1, 10))
    t9 = sum(d9.values())
    for g, nm in ((1, "One"), (3, "Three"), (4, "Four")):
        cmp_exact("VgapShare%sNine" % nm, d9.get(g, F(0)) / t9)

    # ---- C7: the information, by enclosure ---------------------------
    for g, gn in ((1, "One"), (2, "Two"), (4, "Four")):
        for dn, dl in DN.items():
            cmp_iv("VinfoGap%sDelta%s" % (gn, dn), V.info_at(g, dl))
    IPC = {}
    for nm, pe in list(PE.items()) + [("Pure", F(1))]:
        for dn, dl in (("One", 1), ("Two", 2)):
            IPC[(nm, dl)] = V.info_per_contact(1 - pe, dl)
            cmp_iv("Vipc%s%s" % (nm, dn), IPC[(nm, dl)])

    # ---- C8: the direction -------------------------------------------
    tys = V.stream(F(1, 10))
    sgn = F(0)
    for _, S1, a1, p1 in tys:
        for _, S2, a2, p2 in tys:
```

```
            n0 = V.nu(S1, a1, S2, a2, True)
            n1 = V.nu(S1, a1, S2, a2, False)
            if n0 != n1:
                sgn += p1 * p2 * (n0 - n1)
    cmp_exact("VsignedBias", abs(sgn))

    worst = F(0)
    for ph in (F(1, 10), F(1, 2)):
        for r in (3, 10):
            for ell in (3, 4):
                law = V.field_law(V.stream(ph, True), ell, r)
                tgt = F(1, 2 ** (ell - 1))
                worst = max(worst, max(abs(p - tgt) for p in law.values()))
    cmp_exact("VfieldDev", worst)

    wb = V.IV0
    for ph in (F(1, 10), F(1, 2)):
        for r in (3, 10, 100, 1000):
            for ell in (3, 4, 5):
                law = V.field_law(V.stream(ph, True), ell, r)
                for p in law.values():
                    d = -V.log2_iv(p) - V.Iv(ell - 1)
                    m = V.Iv(max(abs(d.lo), abs(d.hi))) / V.Iv(ell - 1)
                    if m.hi > wb.hi:
                        wb = m
    cmp_iv("VcostDevBit", wb)
    cmp_iv("VcostDevPart", wb.inv())

    # ---- C9: the trade ------------------------------------------
    for nm, pe in PE.items():
        cmp_exact("Veta" + nm, V.eta_arm(1 - pe))
    bern_min = None
    b = F(1, 5)
    dc = [sig.coef(k) * k for k in range(1, sig.degree() + 1)]
    q = [dc[k] * b ** k for k in range(len(dc))]
    nn = len(q) - 1

    def binom(a, k):
        r = F(1)
        for i in range(k):
            r = r * (a - i) / (i + 1)
        return r

    for j in range(nn + 1):
        s = F(0)
        for k in range(j + 1):
            s += binom(j, k) / binom(nn, k) * q[k]
        bern_min = s if bern_min is None else min(bern_min, s)
    cmp_exact("VsigDerivLower", bern_min)
    for nm, pe in PE.items():
        for dn, dl in (("One", 1), ("Two", 2)):
            gained = IPC[(nm, dl)] - IPC[("Pure", dl)]
            cmp_iv("Vtrade%s%s" % (nm, dn),
                   gained / V.Iv(1 - V.eta_arm(1 - pe)))

    # ---- C10: the capacity --------------------------------------
    for ell, en in ((6, "Six"), (54, "FiftyFour")):
        cmp_exact("Vcontacts" + en, ell - 1)
        cmp_exact("Vuncond" + en, ell - 1)
```

```
        for nm in ("Pure", "NineFive", "Nine"):
            for dn, dl in (("One", 1), ("Two", 2)):
                mi = V.Iv(ell - 1) * IPC[(nm, dl)]
                cmp_iv("Vmi%s%s%s" % (en, nm, dn), mi)
                cmp_iv("Vcond%s%s%s" % (en, nm, dn),
                       V.Iv(ell - 1) - mi)
    for nm, pe in PE.items():
        cmp_exact("VcapSix" + nm, 5 * sig(1 - pe))
    for g, gn in ((1, "One"), (2, "Two")):
        for dn, dl in (("One", 1), ("Two", 2)):
            th = V.theta_iv(g, dl)
            cmp_iv("VsurprHigh%s%s" % (gn, dn), -V.log2_of_iv(th))
            cmp_iv("VsurprLow%s%s" % (gn, dn),
                   -V.log2_of_iv(V.IV1 - th))
    for nm in ("NineFive", "Nine"):
        ph = 1 - PE[nm]
        full = V.gap_dist(ph, sensitive_only=False)
        for dn, dl in (("One", 1), ("Two", 2)):
            acc = V.IV0
            for g, p in full.items():
                th = V.theta_iv(g, dl) if g else V.Iv(F(1, 2))
                term = -V.log2_of_iv(th) - V.log2_of_iv(V.IV1 - th)
                acc = acc + V.Iv(p) * V.Iv(F(1, 2)) * term
            cmp_iv("VgibbsSix%s%s" % (nm, dn), V.Iv(5) * acc)

    # ---- C11: the refinement ------------------------------------
    for nm, pe in PE.items():
        ph = 1 - pe
        tys = V.stream(ph)
        both = F(0)
        for _, S1, a1, p1 in tys:
            for _, S2, a2, p2 in tys:
                if not V.sensitive(S1, a1, S2, a2):
                    continue
                for _, S3, a3, p3 in tys:
                    if V.sensitive(S2, a2, S3, a3):
                        both += p1 * p2 * p3
        s = sig(ph)
        cmp_exact("Vcorr" + nm, both / (s * s))

    for ell, en in ((6, "Six"), (54, "FiftyFour")):
        for nm, pe in list(PE.items()) + [("Pure", F(1))]:
            ph = 1 - pe
            cmp_exact("Vany%s%s" % (en, nm), 1 - V.p_no_sensitive(ph, ell))
        for nm, pe in PE.items():
            cmp_exact("Vmean%s%s" % (en, nm), (ell - 1) * sig(1 - pe))
    cmp_exact("VanyFiftyFourNineBinom",
              1 - (1 - sig(F(1, 10))) ** 53)
    for nm, pe in PE.items():
        cmp_exact("VstillNeed" + nm, 100 * (1 - sig(1 - pe)))

    for nm, pe, bound in (("Nine", F(90, 100), 30),
                          ("NineFive", F(95, 100), 90),
                          ("NineNine", F(99, 100), 1810)):
        ph = 1 - pe
        got = None
        for ell, pno in V.p_no_sensitive_seq(ph, bound):
            if 1 - pno >= F(1, 2):
                got = ell
```

```
                break
        cmp_exact("Vhalf" + nm, got)
        s = sig(ph)
        k = 1
        while 1 - (1 - s) ** k < F(1, 2):
            k += 1
        cmp_exact("VhalfBinom" + nm, k + 1)

    cmp_exact("VsiteHeritable", 3 * 3 * 6)
    cmp_exact("VboxColumns", 3 * 3)
    cmp_exact("VboxColumnLength", 6)
    cmp_exact("VboxStackContacts", 9 * (6 - 1))
    cmp_exact("VnominalContacts", 54 - 1)

    # ---- 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("    %-28s %s" % (nm, why))
    if SYMBOLIC:
        print("symbolic      : %d" % len(SYMBOLIC))
    if DISAGREE:
        print("\nDISAGREEMENTS:")
        for nm, pub, ex in DISAGREE:
            print("    %-28s published %-24s computed %s" % (nm, pub, ex))
    print("=" * 66)
    return 1 if DISAGREE else 0


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

# Lean 4 Proof Summary: `acf-85-proof.lean`

## Lean Kernel Certification

The file `acf-85-proof.lean` contains 129 theorems in the namespace `ChryseneFormalism.Article85`, 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 129 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-85-proof.tex`.

**The proof file does not reach the entropies.** There is no bracket in it that plays the part the binary-logarithm bracket plays in the Article 84 proof file, because the quantity to be bracketed is not a logarithm of a rational. The figures of blocks C7 and C10 and the trade ratios of C9 therefore have no counterpart in Lean, and the proof dossier states that before its theorem index rather than after it.

## Where the harness and the proof file divide the work

The two artifacts are not duplicates, and for this article the division is sharper than for its companions.
The Lean file certifies the *combinatorial and rational* part: the separation criterion on all 784 contacts, the family dichotomy and its four-letter threshold, the sensitivity polynomial built from the enumeration by the kernel itself, the gap masses, the exact vanishing of the signed bias, the arm efficiency, the positivity certificate for the monotonicity of $\sigma$, the transfer matrix at $\ell = 6$, and the box arithmetic. Where it applies it applies absolutely.
The harness does three things the Lean file does not. It carries the *enclosures*, and so reaches every figure of the article rather than the rational ones alone. It runs the *three-letter alphabet* beside the two-letter one throughout, which is how the article knows which of its statements are alphabet-independent. And it runs the transfer matrix out to $\ell = 1802$ — the exact half-length at $\pi_e = 0.99$ — where the kernel's evaluation of a column that long is not within reach.

Block C12 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 $5 = 5$ would be a decoration rather than a result.