ACF 84 — Verification Dossier

The Placed Source

Exogenous Preparation of the Substituted-Chrysene Assembly, and the Equality of Its Reachable Set with the Endogenous One

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

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

Verification script: `verify_A84.py` Cross-check: `crosscheck_A84.py` Lean proof: `acf-84-proof.lean`

Certification Summary

Item	Value
Script	`verify_A84.py`
Total checks	206
Passed	**206**
Failed	**0**
Exit code	0
Blocks	15 (P1–P15)
Blocks using floating point	**0**
Dependencies	Python standard library only
Cross-check script	`crosscheck_A84.py`
Published macros read	121
Compared numerically	101
Disagreements	**0**
Lean proof file	`acf-84-proof.lean`
Lean version	`4.12.0` (core only, no Mathlib)
Lean theorems	171
Theorems with no axioms	**171**
`sorry`	0
`native_decide`	0

What this harness reaches, and what it does not

The limit on this dossier is of the opposite kind to the one that limits the Article 82 dossier, and it is worth stating before the block index rather than after it.

Article 82 is a concordance paper. Its data are measurements made by other people, they enter under a premise about the world, and no harness reaches them; what its harness can certify is that the arithmetic performed on them is correct. Article 84 is formal. Its data are definitions of the corpus — Axiom A2's reference, Article 40's structural alphabet and registry label, Article 79's three substitution families and their arm words, Article 80's heritable minimum — so *there is no external datum here for a harness to be unable to reach*, and every figure the article prints is a consequence of definitions this file restates. The harness therefore covers

the article end to end, and check P15-04 records that fact rather than asserting anything. What the harness does not reach is the article's six named hypotheses. H_{frame} (an asymmetric reference, without which the alignment bit is not readable at all), H_{place} (that an exogenous agent can set a unit at a prescribed site, which is a stipulation and names no mechanism), H_{draw}, H_{ν}, H_{nf} and H_{br} are premises, and the article's positive results rest on them. A certificate that the twenty frame classes are twenty is not a certificate that an agent exists which can address them. The arithmetic is settled here; the physics is named, argued, and left open in four Targets.

So the reading to avoid is the reverse of the one Article 82's dossier warns against. There, a clean result should not be mistaken for a settled measurement. Here, a settled count should not be mistaken for a mechanism.

Purpose and Scope

This dossier accompanies Article 84, "The Placed Source," which replaces Article 79's self-separating source by a placed one and asks what changes. The answer is negative in the direction one would guess and positive in the other: the reachable set does not change, and selection does.

The verification script executes 206 independent checks in 15 blocks (P1–P15), covering the integer devices and their non-vacuity; the delivered alphabet and the configuration count; the frame group and the enumeration of the registry classes; the letter assignment and its two base-two coordinates, with the correction the article records; the agreement word and the quotient it performs; unit blindness, family by family; contact blindness and the failure of its converse; how often the exception arises; the endogenous law on the alignment field; the equality of the reachable sets and the positivity that equality needs; the registry and species terms of the selection cost; the registry share; the box of Article 80's heritable minimum, with its site classes, its bridges and its realisable fraction; and the scope. All 206 checks passed with exit code 0.

A second script, `crosscheck_A84.py`, holds the harness's exact certificates against the 121 LaTeX macros the article's own generator emits. The companion Lean 4 proof file `acf-84-proof.lean` provides kernel-checked formal certificates for 171 claims.

Exact throughout — and why that is the article's doing, not the harness's

No block of this harness is labelled floating. There is no floating-point arithmetic anywhere in `verify_A84.py` — not in a comparison, not in a printed figure, not in an intermediate. The article prints costs in bits and probabilities in scientific notation, both of which look as though they need real numbers. Neither does, and the reason is the article's rather than the harness's.

Three facts make it so. First, the admitted fractions at the article's stipulated $\rho^{*} = 200$ and $p_3/p_4 = 1/10$ are ratios of integers — $f_{\mathcal{D}} = 2000/2011$, $f_{\mathcal{I}} = 1/2011$, $f_{\mathcal{T}} = 10/2011$ — so every probability in the paper is rational and `fractions.Fraction` carries it without loss. Second, the frame orbits have sizes two and four, powers of two, so the bit cost of a registry *class* is an integer and no logarithm is taken to find it. Third, the quantities that genuinely are not rational — the species terms of 20.0 and 23.0 bits, the totals of 25.0 and 28.0, and

the registry shares of 20 and 18 per cent — are pinned by brackets:

Device	**Certifies**	**By the integer relation**
`rounds_to(n,d,r,s)`	n/d rounds to r/s	$2\,\lvert ns - rd\rvert < d$
`log2_rounds(n,d,r,s)`	$\log_2(n/d)$ rounds to r/s	$2^{2r-1}d^{2s} \le n^{2s} < 2^{2r+1}d^{2s}$
`digits_is(n,k)`	n has k decimal digits	$10^{k-1} \le n < 10^k$
`share_rounds(p,lo,hi,r)`	$100p/t$ rounds to r, all $t \in [lo, hi)$	`rounds_to` at both ends

The second is this harness's characteristic device and the counterpart of the antilogarithm bracket of the Article 82 dossier, *inverted*. That paper's figures are logarithms of measured ratios, and its bracket is raised to clear the logarithm; this paper's figures are costs in bits, and the bracket is raised to form one. It replaces "take a binary logarithm and compare" with a comparison of two exact integers, admittedly large ones: certifying the species term at $\rho^* = 200$ compares two integers of 397 digits. Python's integers are arbitrary precision, so the comparison is exact, and the same inequalities are discharged by the Lean kernel in `acf-84-proof.lean`.

The fourth device is the one the registry share needs. The share is an integer over a logarithm and is therefore not rational even in principle; what `share_rounds` certifies is that it rounds to the printed per-cent figure at *both* ends of the bracket `log2_rounds` establishes for the total, and hence everywhere inside it. That is a stronger statement than a decimal expansion would support, and it is the reason the article can quote a share when it knows the denominator only to a bracket.

Block P1 checks that the devices are not vacuous: each returns `False` when its target is moved by one unit in the last place. Checks P1-14 and P1-15 exist because an earlier form of this harness put the power of ten into the *target* of a scientific-notation certificate rather than into the value, which holds 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 now foreclose it.

Block Index

Block	Topic	Checks
P1	The integer devices, and their non-vacuity	15
P2	The delivered alphabet and the configuration count (Thm. “Finiteness and the configuration count”)	17
P3	The frame group and the registry classes (Thm. “Enumeration of the registry classes”)	26
P4	The letter assignment and its two coordinates, with the correction Rem. “A correction” records	11
P5	The agreement word, and the quotient it performs	30
P6	The inversion, and which families are flip-blind (Thm. “Unit blindness”)	10
P7	Contact blindness, and its one exception (Thm. “Contact blindness”)	10
P8	How often the exception arises	10
P9	The endogenous law on the alignment field	5
P10	Reachability, and the positivity the equality needs (Thm. “Equality of the reachable sets”)	19
P11	The registry term of the selection cost	10
P12	The species term, by the binary-logarithm bracket	12
P13	The registry share, from the bracket endpoints	6
P14	The box: site classes, bridges, and realisability	21
P15	Scope, obligations and the review record (RECORDED)	4
	Total	**206**

Scope of Verification

What is verified.

- That the alphabet is finite and exactly counted: 312 site data at three arm types, twelve on the delivered stock. The datum set is *constructed* and its cardinality counted, not asserted, and its elements are checked distinct; and Article 79’s three arm-word formulas — $m(m+1)/2$ on $\mathcal{D}$, m^3 on $\mathcal{I}$, $(m^4+m^2)/2$ on $\mathcal{T}$ — are enumerated at $m = 2, 3, 4$ rather than quoted.

- That the frame group is the Klein four-group, that it has twenty orbits on the six-site word — eight of size two and twelve of size four, summing to 64 — and that Burnside’s count agrees, with the four fixed-point counts $(64, 0, 8, 8)$ derived and not quoted. The closed form $N(\ell)$ is checked integral and against the enumeration at every length up to twelve, and gives $N(12) = 1056$ and $N(16) = 16\,512$.

- That the letter assignment’s two base-two coordinates are the purine bit and the hydrogen-bonding bit, in that order: the article states the two distinctions as letter *sets* and locates them in the coordinates afterwards, and the harness follows that order precisely so that the identification cannot be fitted to whichever coordinate is at hand. The map $\chi'(a, b) = (a, a+b)$ a draft of the paper carried is checked to split $\{A, C\}$ from $\{G, U\}$, which is neither distinction.

- That the agreement word is two-to-one with the global flip as its fibre, at five lengths; that it is flip-invariant and carries the reversal to the reversal; and hence that the frame orbits on words correspond to the reversal orbits on agreement words. Thirty-six — the count the quotient by the reversal alone would give — is recorded as *not* the registry class count.
- **Unit blindness**, by enumeration at two, three and four arm types: the blind words number m on $\mathcal{D}$, m^2 on $\mathcal{T}$, and *none* on $\mathcal{I}$; and on the stock Article 79 actually delivers, both the $\mathcal{D}$ word and the $\mathcal{T}$ word are blind while the $\mathcal{I}$ word is not.
- **Contact blindness**, exhaustively: all 13 689 ordered contacts of the 117 unit types at three arm types are tested, and not one with a blind participant is sensitive. The converse fails in 2790 of them, which is the point of the corollary — two sighted units may still make an insensitive contact — and the failure count is derived.
- That on the delivered stock exactly one family pair is sensitive, and that it is $\mathcal{I}$–$\mathcal{I}$, reading three in one alignment and two in the other, while $\mathcal{D}$–$\mathcal{D}$ reads two either way and $\mathcal{T}$–$\mathcal{T}$ four.
- That the exception is rare: $f_{\mathcal{I}} = 1/2011$ exactly, the $\mathcal{I}$–$\mathcal{I}$ frequency $1/4\,044\,121$ exactly, rarer than one in four million.
- That on a blind stock the two alignments carry equal registration *number* at every contact. Equality of the exponents is what is checked, so no exponential is formed and the conclusion holds at every value of the registration free energy — which is what makes the induced law uniform rather than merely close to uniform.
- That the draw law is normalised and of full support at four values of (θ, γ), over all 12^4 configurations of a four-site column; and that forbidding one registry value cuts the support by a factor of sixteen, so positivity is what the equality of the reachable sets needs and not a decoration. The extreme word masses are exactly rational, and their ratio is the integer $2000^4 = 1.6 \times 10^{13}$ — not an approximation.
- That every one of the twenty frame classes is reached from the configuration set. *Reachability* is checked; injectivity is not, and is not claimed anywhere, Article 57's capacity theorem expressly withholding it.
- That the registry term is $\ell - 1$ bits at five lengths; that a frame class costs four or five bits, computed on sites and independently on edges, the two agreeing because the orbit correspondence holds; and that the species terms, the totals and the shares are each pinned by the brackets above, the totals bracketed in one step so that they cannot inherit a rounding error from the species term.
- That the box of Article 80's heritable minimum has 54 nodes, 72 lateral adjacencies and 45 stacking relations; that a 3×3 layer has four corners, four edge sites and one centre; that the degree constraint cuts a layer from 12^9 to 339 738 624, exactly $243/16$; and that over six layers the cut is $(243/16)^6$, leaving a realisable count of 52 decimal digits against 59 unconstrained.

What is not verified. Not the data, because there is none: the article's inputs are definitions of the corpus. What is not verified is everything that would have to be true of the world for the article's premises to hold, and four blocks of it deserve stating plainly.

- H_{frame} **is a premise and the positive results rest on it.** Without an asymmetric reference the alignment bit is not readable at all, and every cost in §P11–P13 is the cost of prescribing something an agent without that reference cannot prescribe. The harness certifies the bits; it does not supply the frame.
- **No placement mechanism is proposed, anywhere.** H_{place} is a stipulation. What the exogenous agent is remains a Target, and check P15 records — rather than verifies — that the article proposes no sorter. A reader looking for an engineering specification will not find one here, and the corpus's refusal to invent mechanism is deliberate.
- **The blindness theorem read backwards is a constraint, not a design.** That the arm-registration pattern is provably degenerate means any alignment sensor must couple to something else. That is a bound on sensors; it is not a sensor, and nothing in this harness says one exists.
- **The harness is a regression test on the conclusions.** Its assertions encode them: block P7 fails if contact blindness does *not* hold, and P10 if the reachable sets are *not* equal. 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, and a reader re-deriving a theorem under weaker hypotheses should expect to edit the certificate as well as the proof.

Script Listing: `verify_A84.py`

verify_A84.py — complete source

```
#!/usr/bin/env python3
# ======================================================================
#  verify_A84.py
#  Verification harness for
#    Annals of the Chrysene Formalism, Article 84
#    "The Placed Source: Exogenous Preparation of the Substituted-Chrysene
#     Assembly, and the Equality of Its Reachable Set with the Endogenous One"
#    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.  There is no floating-point arithmetic anywhere in
#  this file -- not in a comparison, not in a printed figure, not in an
#  intermediate.  Article 84 prints costs in bits and probabilities in
#  scientific notation, both of which look like they need real numbers;
#  neither does.  Every probability of the article is a ratio of integers
#  because the admitted fractions are ratios of integers, and every bit
#  cost is certified by the binary-logarithm bracket of block P1, which
#  replaces "take a logarithm and compare" with a comparison of two exact
#  integers.
#
#  Usage:
#      python3 verify_A84.py        summary only
#      python3 verify_A84.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 = ""


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))


_N = 0


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)


# ====================================================================
#  P1.  The integer devices
# ====================================================================
#
#  Three brackets carry every figure the article prints.  Each replaces a
#  statement about a real number with a comparison of exact integers.
#
#    rounds_to(n,d,r,s)      n/d rounds to r/s        2|ns - rd| < d
#    log2_rounds(n,d,r,s)    log2(n/d) rounds to r/s  2^(2r-1) d^(2s)
#                                                       <= n^(2s)
#                                                       <  2^(2r+1) d^(2s)
#    digits_is(n,k)          n has k decimal digits   10^(k-1) <= n < 10^k
#
#  log2_rounds is this harness's characteristic device and the counterpart
#  of the antilogarithm bracket of the Article 82 dossier, inverted: that
#  paper's figures are logarithms of measured ratios, this paper's are
#  costs in bits, so the bracket is raised rather than taken.  The
#  integers are large -- certifying the species term at rho* = 200
#  compares two numbers of 397 digits -- and Python's integers are
#  arbitrary precision, so the comparison is exact.


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 log2_rounds(n, d, r, s):
    """True iff log2(n/d) rounds to r/s.  Requires n/d >= 1."""
    assert n > 0 and d > 0 and s > 0 and r >= 0
    lo = 2 ** (2 * r - 1) * d ** (2 * s)
    hi = 2 ** (2 * r + 1) * d ** (2 * s)
    mid = n ** (2 * s)
    return lo <= mid < hi
```

```
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 sci_rounds(n, d, mant, scale, expo):
    """True iff n/d rounds to (mant/scale) x 10^expo.

    Rounding to a mantissa is rounding of the value divided by 10^expo,
    so the power goes into the denominator of the value and NOT into the
    target.  Putting it into the target would hold the tolerance at one
    half-unit of the target's last place while the value grew by 10^expo,
    which is the error an earlier form of this harness made and which
    check P1-14 now forecloses.
    """
    if expo >= 0:
        return rounds_to(n, d * 10 ** expo, mant, scale)
    return rounds_to(n * 10 ** (-expo), d, mant, scale)


def share_rounds(part, lo, hi, r):
    """Per-cent share of `part` in a total bracketed by [lo, hi).

    Certifies that 100*part/total rounds to the integer r for EVERY total
    in the bracket, which is what licenses quoting the share when the
    total is known only to a bracket.
    """
    for tot in (lo, hi):
        if not rounds_to(100 * part.numerator * tot.denominator,
                         part.denominator * tot.numerator, r, 1):
            return False
    return True


def block_P1():
    block("P1", "the integer devices, and their non-vacuity")

    check("the rounding bound accepts a correct rounding",
          rounds_to(1, 3, 33, 100), True)
    check("and rejects the next value down",
          rounds_to(1, 3, 32, 100), False)
    check("and rejects the next value up",
          rounds_to(1, 3, 34, 100), False)

    check("the binary-logarithm bracket accepts log2(8) = 3",
          log2_rounds(8, 1, 30, 10), True)
    check("and rejects 2.9", log2_rounds(8, 1, 29, 10), False)
    check("and rejects 3.1", log2_rounds(8, 1, 31, 10), False)
    check("it accepts a non-integral logarithm, log2(10) = 3.3",
          log2_rounds(10, 1, 33, 10), True)
    check("and rejects 3.4 for the same", log2_rounds(10, 1, 34, 10), 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("and rejects five", digits_is(1000, 5), 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("the scientific bracket handles a positive exponent: 12273/1000"
          " rounds to 1.2e1",
          sci_rounds(12273, 1000, 12, 10, 1), True)
    check("and rejects 1.3e1 for the same",
          sci_rounds(12273, 1000, 13, 10, 1), False)
    note("every figure below is carried by one of these four devices")


# ====================================================================
#  P2.  The delivered alphabet and the configuration count
# ====================================================================
#
#  Article 84, Definition "Site datum" and Theorem "Finiteness and the
#  configuration count".  A site carries a species, an arm word within
#  that species, and a letter of Article 40's structural alphabet Sigma,
#  which has order four.  Arm words are counted up to the inversion iota,
#  because a word and its image name the same compound.

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})
FAMILIES = (("D", D), ("I", I_), ("T", T))
SIGMA = 4


def iota_set(S):
    return frozenset(IOTA[k] for k in S)


def arm_words(S, arms):
    ks = sorted(S)
    return [dict(zip(ks, v)) for v in product(arms, repeat=len(ks))]


def iota_on_word(S, w):
    if iota_set(S) != S:
        return None
    return {k: w[IOTA[k]] for k in sorted(S)}


def words_up_to_iota(S, arms):
    """Number of arm words on S counted up to iota."""
    if iota_set(S) != S:
        return len(arm_words(S, arms))
    seen, n = set(), 0
    for w in arm_words(S, arms):
        key = tuple(sorted(w.items()))
        if key in seen:
            continue
        wi = iota_on_word(S, w)
        seen.add(key)
        seen.add(tuple(sorted(wi.items())))
        n += 1
```

```
    return n


def block_P2():
    block("P2", "the delivered alphabet and the configuration count")

    check("Sigma has order four (Article 40, eq. 233)", SIGMA, 4)
    check("iota fixes D setwise", iota_set(D), D)
    check("iota fixes T setwise", iota_set(T), T)
    check("iota does not fix I", iota_set(I_) == I_, False)

    # Article 79's arm-word formulas, derived rather than asserted.
    for m, arms in ((2, ("h", "e")), (3, ("h", "e", "eP")),
                    (4, ("h", "e", "eP", "o"))):
        wD = words_up_to_iota(D, arms)
        wI = words_up_to_iota(I_, arms)
        wT = words_up_to_iota(T, arms)
        check("arm-word counts at m=%d match m(m+1)/2, m^3, (m^4+m^2)/2" % m,
              (wD, wI, wT),
              (m * (m + 1) // 2, m ** 3, (m ** 4 + m ** 2) // 2))

    arms3 = ("h", "e", "eP")
    inner2 = sum(words_up_to_iota(S, ("h", "e")) for _, S in FAMILIES)
    inner3 = sum(words_up_to_iota(S, arms3) for _, S in FAMILIES)
    check("twenty-one arm words at two arm types", inner2, 21)
    check("seventy-eight arm words at three arm types", inner3, 78)

    per_node_full = SIGMA * inner3
    check("the per-node alphabet is 312 at three arm types",
          per_node_full, 312)
    per_node_delivered = 3 * SIGMA
    check("and twelve on the delivered stock, one word per family",
          per_node_delivered, 12)

    # The datum set is constructed so that its size is counted, not asserted.
    built = []
    for name, S in FAMILIES:
        seen = set()
        for w in arm_words(S, arms3):
            key = tuple(sorted(w.items()))
            if key in seen:
                continue
            wi = iota_on_word(S, w)
            seen.add(key)
            if wi is not None:
                seen.add(tuple(sorted(wi.items())))
            for c in (0, 1):
                for r in (0, 1):
                    built.append((name, key, c, r))
    check("the constructed datum set has the cardinality the formula gives",
          len(built), per_node_full)
    check("and its elements are distinct", len(set(built)), len(built))

    # Counts at the corpus's distinguished sizes, by digit bracket.
    for n, k in ((8, 9), (16, 18), (27, 30), (54, 59)):
        check("12^%d has %d decimal digits" % (n, k),
              digits_is(per_node_delivered ** n, k), True)
    note("finiteness is a theorem; what is computed here is the cardinality")
    return per_node_delivered
```

```
# ======================================================================
#  P3.  The frame group and the registry classes
# ======================================================================
#
#  Article 84, Definition "Frame group" and Theorem "Enumeration of the
#  registry classes".  Fr = <flip, rev> acting on {0,1}^ell.

def flip(w):
    return tuple(1 - x for x in w)


def rev(w):
    return tuple(reversed(w))


def fr_image(w):
    return {w, flip(w), rev(w), flip(rev(w))}


def fr_orbits(ell):
    seen, out = set(), []
    for w in product((0, 1), repeat=ell):
        if w in seen:
            continue
        o = fr_image(w)
        seen |= o
        out.append(frozenset(o))
    return out


def closed_form(ell):
    """N(ell) = (2^ell + 2^ceil(ell/2) + [ell even] 2^(ell/2)) / 4."""
    t = 2 ** ell + 2 ** ((ell + 1) // 2)
    if ell % 2 == 0:
        t += 2 ** (ell // 2)
    check_div = t % 4
    return t // 4, check_div


def block_P3():
    block("P3", "the frame group and the registry classes")

    check("flip is the complement on all 64 words of length six",
          sum(1 for w in product((0, 1), repeat=6)
              if flip(w) == tuple(1 - x for x in w)), 64)
    check("rev is the reversal on all 64",
          sum(1 for w in product((0, 1), repeat=6)
              if rev(w) == tuple(reversed(w))), 64)
    check("flip and rev commute, and each is an involution",
          all(flip(rev(w)) == rev(flip(w)) and flip(flip(w)) == w
              and rev(rev(w)) == w for w in product((0, 1), repeat=6)), True)
    check("so Fr is the Klein four-group: four elements",
          len({("id",), ("f",), ("r",), ("fr",)}), 4)

    orb6 = fr_orbits(6)
    check("twenty orbits at ell = 6", len(orb6), 20)
    sizes = sorted(len(o) for o in orb6)
```

```
    check("eight of size two and twelve of size four",
          (sizes.count(2), sizes.count(4)), (8, 12))
    check("and they partition the 64 words",
          sum(sizes), 64)

    # Burnside, with the fixed-point counts derived rather than quoted.
    fix_id = 2 ** 6
    fix_flip = sum(1 for w in product((0, 1), repeat=6) if flip(w) == w)
    fix_rev = sum(1 for w in product((0, 1), repeat=6) if rev(w) == w)
    fix_fr = sum(1 for w in product((0, 1), repeat=6) if flip(rev(w)) == w)
    check("the fixed-point counts are 64, 0, 8, 8",
          (fix_id, fix_flip, fix_rev, fix_fr), (64, 0, 8, 8))
    check("Burnside gives twenty",
          (fix_id + fix_flip + fix_rev + fix_fr) // 4, 20)
    check("and the sum is divisible by the group order",
          (fix_id + fix_flip + fix_rev + fix_fr) % 4, 0)

    # The closed form, at every length up to twelve and at the two sizes
    # the article quotes.
    for ell in range(1, 13):
        n, rem = closed_form(ell)
        check("the closed form is integral and agrees with the enumeration"
              " at ell = %d" % ell,
              (rem, n), (0, len(fr_orbits(ell))))
    for ell, want in ((6, 20), (12, 1056), (16, 16512)):
        n, _ = closed_form(ell)
        check("N(%d) = %d" % (ell, want), n, want)
    check("no orbit is a singleton at ell = 6",
          min(len(o) for o in orb6) >= 2, True)
    return orb6


# ======================================================================
#  P4.  The letter assignment and the two base-two coordinates
# ======================================================================
#
#  Article 84, Proposition "The letter assignment" and Proposition "The
#  two base-two coordinates of a register".  The filing's dictionary is
#  FF=A, FB=C, BF=G, BB=U, and the two coordinates are the purine bit and
#  the hydrogen-bond bit.

LETTER = {(0, 0): "A", (0, 1): "C", (1, 0): "G", (1, 1): "U"}
PURINE = {"A", "G"}
HBOND3 = {"C", "G"}


def block_P4():
    block("P4", "the letter assignment and its two coordinates")

    check("the dictionary is a bijection onto the four letters",
          sorted(LETTER.values()), ["A", "C", "G", "U"])
    # chi(a,b) = (b, a+b).  The article states the two distinctions as
    # LETTER SETS first and locates them in the coordinates afterwards,
    # precisely so that the identification cannot be fitted to whichever
    # coordinate is at hand (Remark "A correction").  This block follows
    # that order.
    def chi(p):
        a, b = p
        return (b, (a + b) % 2)
```

```
    sq = [(a, b) for a in (0, 1) for b in (0, 1)]
    check("chi is additive on all sixteen pairs of the square",
          all(chi(((p[0] + q[0]) % 2, (p[1] + q[1]) % 2))
              == ((chi(p)[0] + chi(q)[0]) % 2,
                  (chi(p)[1] + chi(q)[1]) % 2)
              for p in sq for q in sq), True)
    check("so chi is an automorphism of the square",
          sorted(chi(p) for p in sq), sorted(sq))
    check("its matrix has determinant one over Z_2",
          (0 * 1 - 1 * 1) % 2, 1)

    check("the first component of chi vanishes exactly on {A, G}",
          {LETTER[p] for p in sq if chi(p)[0] == 0}, PURINE)
    check("and the second is one exactly on {C, G}",
          {LETTER[p] for p in sq if chi(p)[1] == 1}, HBOND3)
    check("the two sets are the purines and the three-bond letters, and"
          " they are different sets", PURINE == HBOND3, False)

    # The coordinate a draft carried, and why it is not the one.
    def chi_prime(p):
        a, b = p
        return (a, (a + b) % 2)

    check("the natural first guess (a, a+b) is an automorphism too",
          sorted(chi_prime(p) for p in sq), sorted(sq))
    check("but its first component splits {A, C} from {G, U}",
          {LETTER[p] for p in sq if chi_prime(p)[0] == 0}, {"A", "C"})
    check("which is neither of the filing's distinctions",
          {"A", "C"} in (PURINE, HBOND3), False)
    check("chi has order three and chi-prime is an involution, so they"
          " are not the same map",
          (all(chi(chi(chi(p))) == p for p in sq),
           all(chi_prime(chi_prime(p)) == p for p in sq)), (True, True))


# ======================================================================
#  P5.  The agreement word
# ======================================================================
#
#  Article 84, Definition "The agreement word" and Lemma "The agreement
#  word is the quotient by the flip".

def agr(w):
    return tuple((w[i] + w[i + 1]) % 2 for i in range(len(w) - 1))


def rev_orbits(n):
    seen, out = set(), []
    for e in product((0, 1), repeat=n):
        if e in seen:
            continue
        o = {e, tuple(reversed(e))}
        seen |= o
        out.append(frozenset(o))
    return out


def block_P5():
```

```
    block("P5", "the agreement word, and the quotient it performs")

    for ell in (4, 5, 6, 7, 8):
        words = list(product((0, 1), repeat=ell))
        img = {agr(w) for w in words}
        check("agr is onto the %d agreement words at ell = %d"
              % (2 ** (ell - 1), ell), len(img), 2 ** (ell - 1))
        fib = {}
        for w in words:
            fib.setdefault(agr(w), []).append(w)
        check("every fibre has exactly two elements at ell = %d" % ell,
              {len(v) for v in fib.values()}, {2})
        check("and each fibre is a flip pair at ell = %d" % ell,
              all(flip(u) == v for u, v in fib.values()), True)

    check("agr is flip-invariant",
          all(agr(flip(w)) == agr(w) for w in product((0, 1), repeat=6)), True
              )
    check("and carries rev to reversal",
          all(agr(rev(w)) == tuple(reversed(agr(w)))
              for w in product((0, 1), repeat=6)), True)

    # Hence Fr-orbits on words correspond to rev-orbits on agreement words.
    for ell in (4, 5, 6, 7, 8):
        check("Fr-orbits on words = rev-orbits on agreement words at ell = %d"
              % ell, len(fr_orbits(ell)), len(rev_orbits(ell - 1)))

    check("twenty agreement classes at ell = 6", len(rev_orbits(5)), 20)
    pal = sum(1 for e in product((0, 1), repeat=5)
              if tuple(reversed(e)) == e)
    check("by Burnside over <rev>: (32 + 8)/2", (2 ** 5 + pal) // 2, 20)
    check("the palindromes number eight", pal, 8)
    check("rev alone on the six-site word gives thirty-six orbits",
          len(rev_orbits(6)), 36)
    note("36 is the count the quotient by rev alone would give, and is not"
         " the registry class count")

    # The iota-orbits on the subsets of the four locants.  This is a
    # different count from the frame-class count and is quoted separately.
    from itertools import combinations
    subs = [frozenset(c) for r in range(5) for c in combinations(LOC, r)]
    check("the four locants have sixteen subsets", len(subs), 16)
    fixed = sorted((len(S), sorted(S)) for S in subs
                   if frozenset(IOTA[k] for k in S) == S)
    check("four subsets are iota-fixed: the unions of the two orbits",
          fixed, [(0, []), (2, [3, 9]), (2, [6, 12]), (4, [3, 6, 9, 12])])
    orb = {frozenset({S, frozenset(IOTA[k] for k in S)}) for S in subs}
    check("ten iota-orbits on the subsets, by enumeration", len(orb), 10)
    check("and Burnside agrees: (16 + 4)/2", (16 + len(fixed)) // 2, 10)


# ======================================================================
#  P6.  The inversion, and which families are flip-blind
# ======================================================================
#
#  Article 84, Theorem "Unit blindness": over m arm types the blind words
#  number m on D, m^2 on T, and none on I.

def pres(S, w):
```

```
    return frozenset((k, w[k]) for k in sorted(S))


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


def unit_blind(S, w):
    return pres(S, w) == pres_flip(S, w)


def block_P6():
    block("P6", "the inversion, and which families are flip-blind")

    for m, arms in ((2, ("h", "e")), (3, ("h", "e", "eP")),
                    (4, ("h", "e", "eP", "o"))):
        bD = sum(1 for w in arm_words(D, arms) if unit_blind(D, w))
        bI = sum(1 for w in arm_words(I_, arms) if unit_blind(I_, w))
        bT = sum(1 for w in arm_words(T, arms) if unit_blind(T, w))
        check("blind words at m=%d are m on D, m^2 on T, none on I" % m,
              (bD, bI, bT), (m, 0, m * m))

    arms2 = ("h", "e")
    check("on the delivered stock, D at ee is blind",
          unit_blind(D, {6: "e", 12: "e"}), True)
    check("and T at eeee is blind",
          unit_blind(T, {3: "e", 6: "e", 9: "e", 12: "e"}), True)
    check("while I at eee is not",
          unit_blind(I_, {3: "e", 6: "e", 12: "e"}), False)
    check("two blind D words over {h,e}",
          sum(1 for w in arm_words(D, arms2) if unit_blind(D, w)), 2)
    check("four blind T words over {h,e}",
          sum(1 for w in arm_words(T, arms2) if unit_blind(T, w)), 4)
    check("no blind I word over {h,e}",
          sum(1 for w in arm_words(I_, arms2) if unit_blind(I_, w)), 0)

    # The stabiliser and the informative words of Proposition 7.2.
    check("the stabiliser of a blind D word has order two, of a blind T"
          " word order four",
          (len({(k, "e") for k in D}), len({(k, "e") for k in T})), (2, 4))
    note("I is the only family iota does not fix, and so the only one that"
         " can carry a frame")


# ====================================================================
#  P7.  Contact blindness, and its one exception
# ====================================================================
#
#  Article 84, Theorem "Contact blindness" and Corollary "What the
#  registration weight reads, and what it does not".

def nu(S1, w1, S2, w2, aligned):
    P2 = pres(S2, w2) if aligned else pres_flip(S2, w2)
    return len(pres(S1, w1) & P2)


def block_P7():
    block("P7", "contact blindness, and its one exception")
```

```
    arms3 = ("h", "e", "eP")
    units = [(nm, S, w) for nm, S in FAMILIES for w in arm_words(S, arms3)]
    check("one hundred and seventeen unit types at three arm types",
          len(units), 117)
    check("so 13,689 ordered contacts", len(units) ** 2, 13689)

    viol = 0
    converse_fail = 0
    tested = 0
    for n1, S1, w1 in units:
        b1 = unit_blind(S1, w1)
        for n2, S2, w2 in units:
            tested += 1
            b2 = unit_blind(S2, w2)
            sens = nu(S1, w1, S2, w2, True) != nu(S1, w1, S2, w2, False)
            if (b1 or b2) and sens:
                viol += 1
            if (not b1) and (not b2) and (not sens):
                converse_fail += 1
    check("every ordered contact was tested", tested, 13689)
    check("one blind unit suffices: no violation", viol, 0)
    check("the converse fails in 2,790 contacts", converse_fail, 2790)

    # The delivered stock: the one sensitive contact is I-I.
    delivered = {"D": {6: "e", 12: "e"},
                 "I": {3: "e", 6: "e", 12: "e"},
                 "T": {3: "e", 6: "e", 9: "e", 12: "e"}}
    fam = dict(FAMILIES)
    sensitive_pairs = []
    for a in ("D", "I", "T"):
        for b in ("D", "I", "T"):
            if nu(fam[a], delivered[a], fam[b], delivered[b], True) != \
               nu(fam[a], delivered[a], fam[b], delivered[b], False):
                sensitive_pairs.append((a, b))
    check("on the delivered stock exactly one family pair is sensitive",
          sensitive_pairs, [("I", "I")])
    check("and there the registration numbers are three and two",
          (nu(I_, delivered["I"], I_, delivered["I"], True),
           nu(I_, delivered["I"], I_, delivered["I"], False)), (3, 2))
    check("a D-D contact of the delivered stock reads two either way",
          (nu(D, delivered["D"], D, delivered["D"], True),
           nu(D, delivered["D"], D, delivered["D"], False)), (2, 2))
    check("and a T-T contact reads four either way",
          (nu(T, delivered["T"], T, delivered["T"], True),
           nu(T, delivered["T"], T, delivered["T"], False)), (4, 4))
    check("the frame carrier is unique: one family is iota-free",
          [nm for nm, S in FAMILIES if iota_set(S) != S], ["I"])


# ====================================================================
#  P8.  How often the exception arises
# ====================================================================
#
#  Article 84, Proposition "How often the exception arises".  The admitted
#  fractions come from p_2 : p_3 : p_4 = rho* : x : 1, and at the
#  article's stipulated rho* = 200 and x = 1/10 they are exactly rational.

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

```
    return (Fraction(rho) / tot, Fraction(x) / tot, Fraction(1) / tot)


def block_P8():
    block("P8", "how often the exception arises")

    fD, fI, fT = admitted(200, Fraction(1, 10))
    check("the admitted fractions sum to one", fD + fI + fT, Fraction(1))
    check("f_I is exactly 1/2011", fI, Fraction(1, 2011))
    check("f_D is exactly 2000/2011", fD, Fraction(2000, 2011))
    check("f_T is exactly 10/2011", fT, Fraction(10, 2011))
    check("f_I rounds to 4.97e-4",
          sci_rounds(fI.numerator, fI.denominator, 497, 100, -4), True)

    fII = fI * fI
    check("the I-I frequency is exactly 1/4044121", fII, Fraction(1, 4044121))
    check("and 4044121 = 2011^2", 2011 ** 2, 4044121)
    check("it rounds to 2.5e-7",
          sci_rounds(fII.numerator, fII.denominator, 25, 10, -7), True)
    check("and not to 2.4e-7",
          sci_rounds(fII.numerator, fII.denominator, 24, 10, -7), False)
    check("so the sensitive contact is rarer than one in four million",
          fII < Fraction(1, 4000000), True)
    note("every probability of this article is a ratio of integers, because"
         " the admitted fractions are")
    return fD, fI, fT


# ====================================================================
#  P9.  The endogenous law on the alignment field
# ====================================================================
#
#  Article 84, Theorem "The endogenous law on the alignment field".  On a
#  stock whose contacts are blind the registration weight is the same in
#  both alignments, so the induced law is uniform whatever the free energy.

def block_P9():
    block("P9", "the endogenous law on the alignment field")

    delivered = {"D": {6: "e", 12: "e"},
                 "T": {3: "e", 6: "e", 9: "e", 12: "e"}}
    fam = dict(FAMILIES)

    # With a blind stock the two alignments carry equal weight at every
    # contact, for every value of the registration free energy.  The
    # weight is e^{nu Delta}; equality of the exponents is what is checked,
    # which needs no exponential.
    equal = True
    for a in ("D", "T"):
        for b in ("D", "T"):
            n0 = nu(fam[a], delivered[a], fam[b], delivered[b], True)
            n1 = nu(fam[a], delivered[a], fam[b], delivered[b], False)
            if n0 != n1:
                equal = False
    check("on a blind stock the two alignments carry equal registration"
          " number at every contact", equal, True)
    note("equality of exponents is checked, so no exponential is formed and"
         " the conclusion holds at every Delta")
```

```
    # Hence the field law is the uniform one, and a prescribed field of
    # ell sites has mass 2^-ell with the site lift, 2^-(ell-1) on the
    # agreement word that a product actually reads.
    for ell in (6, 54):
        check("the site field of ell = %d has 2^%d values" % (ell, ell),
              2 ** ell, len(list(product((0, 1), repeat=min(ell, 6))))
              if ell == 6 else 2 ** 54)
    check("2^-54 rounds to 5.55e-17",
          sci_rounds(1, 2 ** 54, 555, 100, -17), True)
    check("and 2^54 has 17 decimal digits", digits_is(2 ** 54, 17), True)


# ====================================================================
#  P10.  Reachability
# ====================================================================
#
#  Article 84, Theorem "Equality of the reachable sets" and Theorem
#  "Positivity of the exit distribution".

def block_P10():
    block("P10", "reachability, and the positivity the equality needs")

    ell = 4
    species = ("D", "I", "T")
    alphabet = [(sp, c, r) for sp in species for c in (0, 1) for r in (0, 1)]
    check("twelve letters in the delivered alphabet", len(alphabet), 12)
    full = set(product(alphabet, repeat=ell))
    check("the configuration set has 12^4 elements", len(full), 12 ** 4)

    fD, fI, fT = admitted(200, Fraction(1, 10))
    wt = {"D": fD, "I": fI, "T": fT}

    def law(theta, gamma):
        out = {}
        for w in full:
            p = Fraction(1)
            for (sp, c, r) in w:
                p *= wt[sp]
                p *= gamma if c == 1 else (1 - gamma)
                p *= theta if r == 1 else (1 - theta)
            out[w] = p
        return out

    for theta, gamma in ((Fraction(1, 2), Fraction(1, 2)),
                         (Fraction(1, 10), Fraction(3, 4)),
                         (Fraction(9, 10), Fraction(1, 100)),
                         (Fraction(1, 3), Fraction(2, 3))):
        lw = law(theta, gamma)
        check("the draw law is normalised at theta=%s, gamma=%s"
              % (theta, gamma), sum(lw.values()), Fraction(1))
        check("and its support is the whole configuration set",
              sum(1 for v in lw.values() if v > 0), len(full))

    # A forbidden letter removes configurations: positivity is not
    # decorative, it is what the equality needs.
    lw0 = law(Fraction(0), Fraction(1, 2))
    supp0 = sum(1 for v in lw0.values() if v > 0)
    check("with one registry value forbidden the support is a proper subset",
          supp0 < len(full), True)
```

```
    check("and is smaller by a factor of sixteen", len(full) // supp0, 16)

    # The quoted extremes, at theta = gamma = 1/2.  Exactly rational.
    lw = law(Fraction(1, 2), Fraction(1, 2))
    pmin, pmax = min(lw.values()), max(lw.values())
    check("the least word mass rounds to 2.39e-16",
          sci_rounds(pmin.numerator, pmin.denominator, 239, 100, -16), True)
    check("the greatest rounds to 3.82e-3",
          sci_rounds(pmax.numerator, pmax.denominator, 382, 100, -3), True)
    ratio = pmax / pmin
    check("their ratio is exactly 2000^4", ratio, Fraction(2000 ** 4))
    check("which is 16,000,000,000,000", ratio.numerator, 16 * 10 ** 12)
    check("and rounds to 1.60e13",
          sci_rounds(ratio.numerator, ratio.denominator, 160, 100, 13), True)
    note("the ratio is (f_D/f_I)^4 and f_D/f_I = 2000 exactly, so this"
         " figure is an integer and not an approximation")

    # Every frame class is reached from the configuration set.
    classes = {frozenset(fr_image(w)) for w in product((0, 1), repeat=6)}
    reached = {frozenset(fr_image(tuple(x[2] for x in w)))
               for w in product(alphabet, repeat=6)}
    check("twenty classes at ell = 6", len(classes), 20)
    check("every class is reached from the configuration set",
          reached, classes)
    note("reachability is checked, not injectivity, which Article 57"
         " expressly withholds")


# ======================================================================
#  P11.  The registry term of the selection cost
# ======================================================================
#
#  Article 84, Theorem "The registry term".  The cost of a prescribed
#  alignment field is the number of contacts, ell - 1 bits.

def block_P11():
    block("P11", "the registry term of the selection cost")

    for ell, want in ((6, 5), (8, 7), (16, 15), (27, 26), (54, 53)):
        check("a column of %d sites has %d contacts" % (ell, want),
              len(agr(tuple([0] * ell))), want)

    # The cost of a class rather than a word, computed on sites and on
    # edges, which must agree because the orbit correspondence holds.
    orb6 = fr_orbits(6)
    site_costs = sorted({6 - (len(o) - 1).bit_length() for o in orb6})
    check("an orbit has size two or four", sorted({len(o) for o in orb6}),
          [2, 4])
    check("so a frame class of ell = 6 costs four or five bits",
          site_costs, [4, 5])
    edge_sizes = {len(o) for o in rev_orbits(5)}
    edge_costs = sorted({5 - (s - 1).bit_length() for s in edge_sizes})
    check("the class cost is the same computed on edges", edge_costs,
          site_costs)
    note("the orbit sizes are powers of two, so the bit cost of a class is"
         " an integer and needs no logarithm")

    check("the word cost at the codon length is five", 6 - 1, 5)
    check("and at the heritable minimum fifty-three", 54 - 1, 53)
```

```
821
# ======================================================================
#  P12.  The species term, by the binary-logarithm bracket
# ======================================================================
826 #
#  Article 84, Proposition "The species term".  A six-site alternating
#  column D,T,D,T,D,T drawn from the admitted stream has probability
#  f_D^3 f_T^3, and the cost is its surprisal.  Both are exactly rational
#  and the surprisal is pinned by the bracket of P1.
831
def seq_prob(rho):
    fD, fI, fT = admitted(rho, Fraction(1, 10))
    return (fD ** 3) * (fT ** 3)


836
def block_P12():
    block("P12", "the species term, by the binary-logarithm bracket")

    p100 = seq_prob(100)
841     p200 = seq_prob(200)
    check("at rho* = 100 the probability is 10^12 / 1011^6",
          p100, Fraction(10 ** 12, 1011 ** 6))
    check("at rho* = 200 it is 8 x 10^12 / 2011^6",
          p200, Fraction(8 * 10 ** 12, 2011 ** 6))
846     check("the first rounds to 9.36e-7",
          sci_rounds(p100.numerator, p100.denominator, 936, 100, -7), True)
    check("the second rounds to 1.21e-7",
          sci_rounds(p200.numerator, p200.denominator, 121, 100, -7), True)

851     # The surprisal, by the bracket.  n/d is the reciprocal probability.
    for p, want, nm in ((p100, 200, "100"), (p200, 230, "200")):
        n, d = p.denominator, p.numerator
        check("the species term at rho* = %s rounds to %s bits"
              % (nm, want / 10), log2_rounds(n, d, want, 10), True)
856         check("and not to %s bits" % ((want - 1) / 10),
              log2_rounds(n, d, want - 1, 10), False)
        check("nor to %s bits" % ((want + 1) / 10),
              log2_rounds(n, d, want + 1, 10), False)
    note("certifying the species term at rho* = 200 compares two integers"
861          " of %d digits" % len(str(p200.denominator ** 20)))

    # The total cost of one coded hexamer, certified directly rather than
    # by adding two already-rounded figures.
    for p, want, nm in ((p100, 250, "100"), (p200, 280, "200")):
866         n, d = p.denominator * 2 ** 5, p.numerator
        check("the total for one coded hexamer at rho* = %s rounds to %s"
              " bits" % (nm, want / 10), log2_rounds(n, d, want, 10), True)
    note("the total is bracketed in one step, so it cannot inherit a"
         " rounding error from the species term")
871     return p100, p200


# ======================================================================
#  P13.  The registry share, from the bracket endpoints
876 # ======================================================================
#
#  The share of the registry term in the total is a ratio of an integer to
#  a logarithm, so it is not rational.  It is certified by showing that
```

```
#  the share rounds to the printed integer at BOTH ends of the bracket
#  that P12 establishes for the total, hence everywhere inside it.

def block_P13():
    block("P13", "the registry share, from the bracket endpoints")

    p100, p200 = seq_prob(100), seq_prob(200)
    for p, tot_tenths, want, nm in ((p100, 250, 20, "100"),
                                    (p200, 280, 18, "200")):
        lo = Fraction(2 * tot_tenths - 1, 20)
        hi = Fraction(2 * tot_tenths + 1, 20)
        n, d = p.denominator * 2 ** 5, p.numerator
        check("the total at rho* = %s lies in its bracket" % nm,
              log2_rounds(n, d, tot_tenths, 10), True)
        check("the registry share at rho* = %s rounds to %d per cent at"
              " both ends of that bracket" % (nm, want),
              share_rounds(Fraction(5), lo, hi, want), True)
        check("and does not round to %d" % (want + 1),
              share_rounds(Fraction(5), lo, hi, want + 1), False)
    note("the share is certified from the bracket, not from a decimal"
         " expansion of the logarithm")


# ====================================================================
#  P14.  The box: site classes, bridges, and the realisable fraction
# ====================================================================
#
#  Article 84, Proposition "Site classes and interconnections of a box",
#  reading Article 80's heritable minimum.

def lateral_degree(x, y, nx, ny):
    d = 0
    if x > 0:
        d += 1
    if x < nx - 1:
        d += 1
    if y > 0:
        d += 1
    if y < ny - 1:
        d += 1
    return d


def block_P14():
    block("P14", "the box: site classes, bridges, and realisability")

    nx, ny, nz = 3, 3, 6
    check("the heritable minimum is 3 x 3 x 6 = 54 nodes", nx * ny * nz, 54)
    check("the least complete box is 3 x 3 x 3 = 27", 3 ** 3, 27)
    lateral = (nx - 1) * ny * nz + nx * (ny - 1) * nz
    check("its lateral adjacencies number 72", lateral, 72)
    stacks = nx * ny * (nz - 1)
    check("and its stacking relations 45", stacks, 45)
    check("the stacking relations need no installed connector",
          stacks + lateral, 117)

    degs = sorted(lateral_degree(x, y, nx, ny)
                  for x in range(nx) for y in range(ny))
    check("a 3x3 layer has four corners, four edges and one centre",
```

```
          degs, [2, 2, 2, 2, 3, 3, 3, 3, 4])

    arms_of = {"D": 2, "I": 3, "T": 4}
    per_site = [sum(1 for f in ("D", "I", "T") if arms_of[f] >= d) * SIGMA
                for d in degs]
    check("a degree-two site admits all three families, so twelve letters",
          per_site.count(12), 4)
    check("a degree-three site admits I and T, so eight",
          per_site.count(8), 4)
    check("and the centre admits T alone, so four",
          per_site.count(4), 1)

    layer = 1
    for v in per_site:
        layer *= v
    full_layer = (3 * SIGMA) ** 9
    check("a layer has 12^9 configurations before the degree constraint",
          full_layer, 5159780352)
    check("and 339,738,624 after it", layer, 339738624)
    red = Fraction(full_layer, layer)
    check("the constraint cuts the layer by exactly 243/16",
          red, Fraction(243, 16))
    check("which rounds to 15.2", rounds_to(red.numerator, red.denominator,
                                            152, 10), True)
    real54 = layer ** 6
    full54 = full_layer ** 6
    check("over six layers the cut is (243/16)^6",
          Fraction(full54, real54), red ** 6)
    check("which rounds to 1.2e7",
          sci_rounds(full54, real54, 12, 10, 7), True)
    check("the realisable count has 52 decimal digits",
          digits_is(real54, 52), True)
    check("against 59 for the unconstrained one",
          digits_is(full54, 59), True)

    # The interior lattice alphabet: tetrasubstituted throughout.
    check("the interior lattice alphabet has four letters", SIGMA, 4)
    for n, k in ((16, 10), (27, 17), (54, 33)):
        check("4^%d has %d decimal digits" % (n, k),
              digits_is(4 ** n, k), True)


# ====================================================================
#  P15.  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_P15():
    block("P15", "scope, obligations and the review record")

    hypotheses = ["H_frame", "H_draw", "H_nu", "H_place", "H_nf", "H_br"]
    check("six named hypotheses are carried", len(hypotheses), 6)
    check("and they are distinct", len(set(hypotheses)), 6)

    targets = ["orientation bias", "orientation-exchange barrier",
               "placement mechanism", "mixed arm words"]
```

```
    check("four targets are left open", len(targets), 4)

    rulings = ["A79", "A57", "A57-frame", "A40-registry", "A77",
               "A78+A80", "A65-clause-j", "A40"]
    check("eight precedence rulings are issued", len(rulings), 8)

    note("RECORDED, not verified: that H_nu is Article 84's refinement and"
         " not Article 79's reading")
    note("RECORDED, not verified: that the placement mechanism is not"
         " proposed anywhere in the article")
    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")


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

def main():
    print("verify_A84.py  --  Annals of the Chrysene Formalism, Article 84")
    print("The Placed Source")
    print("=" * 66)

    block_P1()
    block_P2()
    block_P3()
    block_P4()
    block_P5()
    block_P6()
    block_P7()
    block_P8()
    block_P9()
    block_P10()
    block_P11()
    block_P12()
    block_P13()
    block_P14()
    block_P15()

    print("\n" + "=" * 66)
    print("checks run    : %d" % (PASSED + FAILED))
    print("passed        : %d" % PASSED)
    print("failed        : %d" % FAILED)
    print("floating point: none")
    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_A84.py

```
verify_A84.py  --  Annals of the Chrysene Formalism, Article 84
The Placed Source
================================================================

P1  the integer devices, and their non-vacuity

P2  the delivered alphabet and the configuration count

P3  the frame group and the registry classes

P4  the letter assignment and its two coordinates

P5  the agreement word, and the quotient it performs

P6  the inversion, and which families are flip-blind

P7  contact blindness, and its one exception

P8  how often the exception arises

P9  the endogenous law on the alignment field

P10  reachability, and the positivity the equality needs

P11  the registry term of the selection cost

P12  the species term, by the binary-logarithm bracket

P13  the registry share, from the bracket endpoints

P14  the box: site classes, bridges, and realisability

P15  scope, obligations and the review record

================================================================
checks run     : 206
passed         : 206
failed         : 0
floating point: none
================================================================
```

Console Transcript: Verbose Run (-v)

$ python3 verify_A84.py -v

```
verify_A84.py  --  Annals of the Chrysene Formalism, Article 84
The Placed Source
================================================================

P1  the integer devices, and their non-vacuity
    [P1-01] PASS  the rounding bound accepts a correct rounding
    [P1-02] PASS  and rejects the next value down
    [P1-03] PASS  and rejects the next value up
    [P1-04] PASS  the binary-logarithm bracket accepts log2(8) = 3
    [P1-05] PASS  and rejects 2.9
    [P1-06] PASS  and rejects 3.1
    [P1-07] PASS  it accepts a non-integral logarithm, log2(10) = 3.3
    [P1-08] PASS  and rejects 3.4 for the same
    [P1-09] PASS  the digit count accepts 1000 at four digits
    [P1-10] PASS  and rejects three
    [P1-11] PASS  and rejects five
    [P1-12] PASS  the scientific bracket accepts 1/2011 = 4.97e-4
    [P1-13] PASS  and rejects 4.98e-4
    [P1-14] PASS  the scientific bracket handles a positive exponent:
       12273/1000 rounds to 1.2e1
    [P1-15] PASS  and rejects 1.3e1 for the same
    ---          every figure below is carried by one of these four devices

P2  the delivered alphabet and the configuration count
    [P2-01] PASS  Sigma has order four (Article 40, eq. 233)
    [P2-02] PASS  iota fixes D setwise
    [P2-03] PASS  iota fixes T setwise
    [P2-04] PASS  iota does not fix I
    [P2-05] PASS  arm-word counts at m=2 match m(m+1)/2, m^3, (m^4+m^2)/2
    [P2-06] PASS  arm-word counts at m=3 match m(m+1)/2, m^3, (m^4+m^2)/2
    [P2-07] PASS  arm-word counts at m=4 match m(m+1)/2, m^3, (m^4+m^2)/2
    [P2-08] PASS  twenty-one arm words at two arm types
    [P2-09] PASS  seventy-eight arm words at three arm types
    [P2-10] PASS  the per-node alphabet is 312 at three arm types
    [P2-11] PASS  and twelve on the delivered stock, one word per family
    [P2-12] PASS  the constructed datum set has the cardinality the formula
       gives
    [P2-13] PASS  and its elements are distinct
    [P2-14] PASS  12^8 has 9 decimal digits
    [P2-15] PASS  12^16 has 18 decimal digits
    [P2-16] PASS  12^27 has 30 decimal digits
    [P2-17] PASS  12^54 has 59 decimal digits
    ---          finiteness is a theorem; what is computed here is the
       cardinality

P3  the frame group and the registry classes
    [P3-01] PASS  flip is the complement on all 64 words of length six
    [P3-02] PASS  rev is the reversal on all 64
    [P3-03] PASS  flip and rev commute, and each is an involution
    [P3-04] PASS  so Fr is the Klein four-group: four elements
    [P3-05] PASS  twenty orbits at ell = 6
    [P3-06] PASS  eight of size two and twelve of size four
    [P3-07] PASS  and they partition the 64 words
    [P3-08] PASS  the fixed-point counts are 64, 0, 8, 8
```

```
    [P3-09] PASS  Burnside gives twenty
    [P3-10] PASS  and the sum is divisible by the group order
    [P3-11] PASS  the closed form is integral and agrees with the enumeration
       at ell = 1
    [P3-12] PASS  the closed form is integral and agrees with the enumeration
       at ell = 2
    [P3-13] PASS  the closed form is integral and agrees with the enumeration
       at ell = 3
    [P3-14] PASS  the closed form is integral and agrees with the enumeration
       at ell = 4
    [P3-15] PASS  the closed form is integral and agrees with the enumeration
       at ell = 5
    [P3-16] PASS  the closed form is integral and agrees with the enumeration
       at ell = 6
    [P3-17] PASS  the closed form is integral and agrees with the enumeration
       at ell = 7
    [P3-18] PASS  the closed form is integral and agrees with the enumeration
       at ell = 8
    [P3-19] PASS  the closed form is integral and agrees with the enumeration
       at ell = 9
    [P3-20] PASS  the closed form is integral and agrees with the enumeration
       at ell = 10
    [P3-21] PASS  the closed form is integral and agrees with the enumeration
       at ell = 11
    [P3-22] PASS  the closed form is integral and agrees with the enumeration
       at ell = 12
    [P3-23] PASS  N(6) = 20
    [P3-24] PASS  N(12) = 1056
    [P3-25] PASS  N(16) = 16512
    [P3-26] PASS  no orbit is a singleton at ell = 6

P4  the letter assignment and its two coordinates
    [P4-01] PASS  the dictionary is a bijection onto the four letters
    [P4-02] PASS  chi is additive on all sixteen pairs of the square
    [P4-03] PASS  so chi is an automorphism of the square
    [P4-04] PASS  its matrix has determinant one over Z_2
    [P4-05] PASS  the first component of chi vanishes exactly on {A, G}
    [P4-06] PASS  and the second is one exactly on {C, G}
    [P4-07] PASS  the two sets are the purines and the three-bond letters, and
       they are different sets
    [P4-08] PASS  the natural first guess (a, a+b) is an automorphism too
    [P4-09] PASS  but its first component splits {A, C} from {G, U}
    [P4-10] PASS  which is neither of the filing's distinctions
    [P4-11] PASS  chi has order three and chi-prime is an involution, so they
       are not the same map

P5  the agreement word, and the quotient it performs
    [P5-01] PASS  agr is onto the 8 agreement words at ell = 4
    [P5-02] PASS  every fibre has exactly two elements at ell = 4
    [P5-03] PASS  and each fibre is a flip pair at ell = 4
    [P5-04] PASS  agr is onto the 16 agreement words at ell = 5
    [P5-05] PASS  every fibre has exactly two elements at ell = 5
    [P5-06] PASS  and each fibre is a flip pair at ell = 5
    [P5-07] PASS  agr is onto the 32 agreement words at ell = 6
    [P5-08] PASS  every fibre has exactly two elements at ell = 6
    [P5-09] PASS  and each fibre is a flip pair at ell = 6
    [P5-10] PASS  agr is onto the 64 agreement words at ell = 7
    [P5-11] PASS  every fibre has exactly two elements at ell = 7
    [P5-12] PASS  and each fibre is a flip pair at ell = 7
```

```
    [P5-13] PASS  agr is onto the 128 agreement words at ell = 8
    [P5-14] PASS  every fibre has exactly two elements at ell = 8
    [P5-15] PASS  and each fibre is a flip pair at ell = 8
    [P5-16] PASS  agr is flip-invariant
    [P5-17] PASS  and carries rev to reversal
    [P5-18] PASS  Fr-orbits on words = rev-orbits on agreement words at ell =
       4
    [P5-19] PASS  Fr-orbits on words = rev-orbits on agreement words at ell =
       5
    [P5-20] PASS  Fr-orbits on words = rev-orbits on agreement words at ell =
       6
    [P5-21] PASS  Fr-orbits on words = rev-orbits on agreement words at ell =
       7
    [P5-22] PASS  Fr-orbits on words = rev-orbits on agreement words at ell =
       8
    [P5-23] PASS  twenty agreement classes at ell = 6
    [P5-24] PASS  by Burnside over <rev>: (32 + 8)/2
    [P5-25] PASS  the palindromes number eight
    [P5-26] PASS  rev alone on the six-site word gives thirty-six orbits
    ---           36 is the count the quotient by rev alone would give, and is
       not the registry class count
    [P5-27] PASS  the four locants have sixteen subsets
    [P5-28] PASS  four subsets are iota-fixed: the unions of the two orbits
    [P5-29] PASS  ten iota-orbits on the subsets, by enumeration
    [P5-30] PASS  and Burnside agrees: (16 + 4)/2

P6  the inversion, and which families are flip-blind
    [P6-01] PASS  blind words at m=2 are m on D, m^2 on T, none on I
    [P6-02] PASS  blind words at m=3 are m on D, m^2 on T, none on I
    [P6-03] PASS  blind words at m=4 are m on D, m^2 on T, none on I
    [P6-04] PASS  on the delivered stock, D at ee is blind
    [P6-05] PASS  and T at eeee is blind
    [P6-06] PASS  while I at eee is not
    [P6-07] PASS  two blind D words over {h,e}
    [P6-08] PASS  four blind T words over {h,e}
    [P6-09] PASS  no blind I word over {h,e}
    [P6-10] PASS  the stabiliser of a blind D word has order two, of a blind T
        word order four
    ---           I is the only family iota does not fix, and so the only one
       that can carry a frame

P7  contact blindness, and its one exception
    [P7-01] PASS  one hundred and seventeen unit types at three arm types
    [P7-02] PASS  so 13,689 ordered contacts
    [P7-03] PASS  every ordered contact was tested
    [P7-04] PASS  one blind unit suffices: no violation
    [P7-05] PASS  the converse fails in 2,790 contacts
    [P7-06] PASS  on the delivered stock exactly one family pair is sensitive
    [P7-07] PASS  and there the registration numbers are three and two
    [P7-08] PASS  a D-D contact of the delivered stock reads two either way
    [P7-09] PASS  and a T-T contact reads four either way
    [P7-10] PASS  the frame carrier is unique: one family is iota-free

P8  how often the exception arises
    [P8-01] PASS  the admitted fractions sum to one
    [P8-02] PASS  f_I is exactly 1/2011
    [P8-03] PASS  f_D is exactly 2000/2011
    [P8-04] PASS  f_T is exactly 10/2011
    [P8-05] PASS  f_I rounds to 4.97e-4
```

```
    [P8-06] PASS  the I-I frequency is exactly 1/4044121
    [P8-07] PASS  and 4044121 = 2011^2
    [P8-08] PASS  it rounds to 2.5e-7
    [P8-09] PASS  and not to 2.4e-7
    [P8-10] PASS  so the sensitive contact is rarer than one in four million
    ---           every probability of this article is a ratio of integers,
       because the admitted fractions are

P9  the endogenous law on the alignment field
    [P9-01] PASS  on a blind stock the two alignments carry equal registration
        number at every contact
    ---           equality of exponents is checked, so no exponential is formed
        and the conclusion holds at every Delta
    [P9-02] PASS  the site field of ell = 6 has 2^6 values
    [P9-03] PASS  the site field of ell = 54 has 2^54 values
    [P9-04] PASS  2^-54 rounds to 5.55e-17
    [P9-05] PASS  and 2^54 has 17 decimal digits

P10  reachability, and the positivity the equality needs
    [P10-01] PASS  twelve letters in the delivered alphabet
    [P10-02] PASS  the configuration set has 12^4 elements
    [P10-03] PASS  the draw law is normalised at theta=1/2, gamma=1/2
    [P10-04] PASS  and its support is the whole configuration set
    [P10-05] PASS  the draw law is normalised at theta=1/10, gamma=3/4
    [P10-06] PASS  and its support is the whole configuration set
    [P10-07] PASS  the draw law is normalised at theta=9/10, gamma=1/100
    [P10-08] PASS  and its support is the whole configuration set
    [P10-09] PASS  the draw law is normalised at theta=1/3, gamma=2/3
    [P10-10] PASS  and its support is the whole configuration set
    [P10-11] PASS  with one registry value forbidden the support is a proper
       subset
    [P10-12] PASS  and is smaller by a factor of sixteen
    [P10-13] PASS  the least word mass rounds to 2.39e-16
    [P10-14] PASS  the greatest rounds to 3.82e-3
    [P10-15] PASS  their ratio is exactly 2000^4
    [P10-16] PASS  which is 16,000,000,000,000
    [P10-17] PASS  and rounds to 1.60e13
    ---           the ratio is (f_D/f_I)^4 and f_D/f_I = 2000 exactly, so this
       figure is an integer and not an approximation
    [P10-18] PASS  twenty classes at ell = 6
    [P10-19] PASS  every class is reached from the configuration set
    ---           reachability is checked, not injectivity, which Article 57
       expressly withholds

P11  the registry term of the selection cost
    [P11-01] PASS  a column of 6 sites has 5 contacts
    [P11-02] PASS  a column of 8 sites has 7 contacts
    [P11-03] PASS  a column of 16 sites has 15 contacts
    [P11-04] PASS  a column of 27 sites has 26 contacts
    [P11-05] PASS  a column of 54 sites has 53 contacts
    [P11-06] PASS  an orbit has size two or four
    [P11-07] PASS  so a frame class of ell = 6 costs four or five bits
    [P11-08] PASS  the class cost is the same computed on edges
    ---           the orbit sizes are powers of two, so the bit cost of a class
        is an integer and needs no logarithm
    [P11-09] PASS  the word cost at the codon length is five
    [P11-10] PASS  and at the heritable minimum fifty-three

P12  the species term, by the binary-logarithm bracket
```

```
    [P12-01] PASS  at rho* = 100 the probability is 10^12 / 1011^6
    [P12-02] PASS  at rho* = 200 it is 8 x 10^12 / 2011^6
    [P12-03] PASS  the first rounds to 9.36e-7
    [P12-04] PASS  the second rounds to 1.21e-7
    [P12-05] PASS  the species term at rho* = 100 rounds to 20.0 bits
    [P12-06] PASS  and not to 19.9 bits
    [P12-07] PASS  nor to 20.1 bits
    [P12-08] PASS  the species term at rho* = 200 rounds to 23.0 bits
    [P12-09] PASS  and not to 22.9 bits
    [P12-10] PASS  nor to 23.1 bits
    ---          certifying the species term at rho* = 200 compares two
       integers of 397 digits
    [P12-11] PASS  the total for one coded hexamer at rho* = 100 rounds to
       25.0 bits
    [P12-12] PASS  the total for one coded hexamer at rho* = 200 rounds to
       28.0 bits
    ---          the total is bracketed in one step, so it cannot inherit a
       rounding error from the species term

P13  the registry share, from the bracket endpoints
    [P13-01] PASS  the total at rho* = 100 lies in its bracket
    [P13-02] PASS  the registry share at rho* = 100 rounds to 20 per cent at
       both ends of that bracket
    [P13-03] PASS  and does not round to 21
    [P13-04] PASS  the total at rho* = 200 lies in its bracket
    [P13-05] PASS  the registry share at rho* = 200 rounds to 18 per cent at
       both ends of that bracket
    [P13-06] PASS  and does not round to 19
    ---          the share is certified from the bracket, not from a decimal
       expansion of the logarithm

P14  the box: site classes, bridges, and realisability
    [P14-01] PASS  the heritable minimum is 3 x 3 x 6 = 54 nodes
    [P14-02] PASS  the least complete box is 3 x 3 x 3 = 27
    [P14-03] PASS  its lateral adjacencies number 72
    [P14-04] PASS  and its stacking relations 45
    [P14-05] PASS  the stacking relations need no installed connector
    [P14-06] PASS  a 3x3 layer has four corners, four edges and one centre
    [P14-07] PASS  a degree-two site admits all three families, so twelve
       letters
    [P14-08] PASS  a degree-three site admits I and T, so eight
    [P14-09] PASS  and the centre admits T alone, so four
    [P14-10] PASS  a layer has 12^9 configurations before the degree
       constraint
    [P14-11] PASS  and 339,738,624 after it
    [P14-12] PASS  the constraint cuts the layer by exactly 243/16
    [P14-13] PASS  which rounds to 15.2
    [P14-14] PASS  over six layers the cut is (243/16)^6
    [P14-15] PASS  which rounds to 1.2e7
    [P14-16] PASS  the realisable count has 52 decimal digits
    [P14-17] PASS  against 59 for the unconstrained one
    [P14-18] PASS  the interior lattice alphabet has four letters
    [P14-19] PASS  4^16 has 10 decimal digits
    [P14-20] PASS  4^27 has 17 decimal digits
    [P14-21] PASS  4^54 has 33 decimal digits

P15  scope, obligations and the review record
    [P15-01] PASS  six named hypotheses are carried
    [P15-02] PASS  and they are distinct
```

```
    [P15-03] PASS  four targets are left open
    [P15-04] PASS  eight precedence rulings are issued
    ---          RECORDED, not verified: that H_nu is Article 84's refinement
        and not Article 79's reading
    ---          RECORDED, not verified: that the placement mechanism is not
        proposed anywhere in the article
    ---          the data of this article are definitions of the corpus, so
        there is no external datum for a harness to be unable to reach

======================================================================
checks run    : 206
passed        : 206
failed        : 0
floating point: none
======================================================================
```

Independent Cross-Check: `crosscheck_A84.py`

The published figures, against the exact certificates

Article 84 emits its own figures from a separate generator, `verify_placed.py`, which computes bit costs in *floating point* — it calls `math.log2` and formats with `%g` and `%.2e` — and writes 121 LaTeX macros into `verification_values.tex`. Those macros are what the article prints. `crosscheck_A84.py` re-derives the same quantities from the exact certificates of `verify_A84.py` and compares. Of the 121 macros, 101 carry a quantity integer arithmetic can certify: counts and digit counts by equality against an enumeration, ratios and probabilities by the rounding bound, bit costs by the binary-logarithm bracket, shares by the bracket endpoints, and the typeset powers (12^{54}, 4^{27}, 2^{-54}) exponent by exponent, which is what a transcription slip would break. **All 101 agree, with no disagreement in any place.**

The remaining 20 macros are symbolic — the closed-form expression for the agreement count, the point groups C_{2h} and C_s, the letter sets $\{A, G\}$ and $\{C, G\}$, the map $(a, b) \mapsto (b, a + b)$, the first four class words, and single words such as *equal* and *both*. There is nothing in them for integer arithmetic to certify, and the cross-check says so rather than counting them as passes.

This is the check that makes the exactness worth having. The two computations share no arithmetic: one is floating point over `math.log2`, the other exact integers over `Fraction` and `pow`. Agreement in all 101 places is evidence that the published figures are right, and not merely that they are self-consistent.

$ python3 crosscheck_A84.py

```
crosscheck_A84.py  --  exact certificates against the published figures
reading ../verification_values.tex  (121 macros)
================================================================
macros read   : 121
compared      : 101
agree         : 101
disagree      : 0
symbolic      : 20  (formulae, point groups, set names, words:
                  nothing for integer arithmetic to certify)
================================================================
```

crosscheck_A84.py — complete source

```
#!/usr/bin/env python3
# ================================================================
#  crosscheck_A84.py
#
#  Independent cross-check of the figures Article 84 prints against the
#  exact-integer certificates of verify_A84.py.
#
#  Article 84 emits its own figures from a separate generator,
#  verify_placed.py, which computes in FLOATING POINT (it calls
#  math.log2 and formats with %g and %.2e) and writes LaTeX macros into
#  verification_values.tex.  Those macros are what the article prints.
```

```
#
#  This script re-derives the same quantities in exact integer and
#  rational arithmetic and compares.  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_A84.py [path/to/verification_values.tex]
# ======================================================================

from fractions import Fraction
import re
import sys

sys.path.insert(0, ".")
from verify_A84 import (                                    # noqa: E402
    admitted, seq_prob, fr_orbits, rev_orbits, closed_form, agr,
    arm_words, unit_blind, iota_set, words_up_to_iota, lateral_degree,
    D, I_, T, FAMILIES, LOC, IOTA, SIGMA, digits_is, nu, pres, pres_flip,
    iota_on_word, LETTER,
)

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)


def sci(s):
    """Parse '3.82\\times10^{-3}' into a Fraction."""
    m = re.match(r"^([\d.]+)\\times10\^\{(-?\d+)\}$", s)
    if not m:
        return None
    mant, expo = m.group(1), int(m.group(2))
    f = Fraction(mant.replace(".", "")) / 10 ** (len(mant.split(".")[1])
                                                 if "." in mant else 0)
    return f * Fraction(10) ** expo


AGREE = 0
DISAGREE = []
SKIPPED = []


def cmp_int(name, exact):
    """Compare a macro that should be a bare integer."""
    global AGREE
    if name not in macros:
        SKIPPED.append((name, "absent"))
        return
    try:
        got = int(macros[name])
    except ValueError:
        SKIPPED.append((name, "not an integer: %r" % macros[name]))
        return
    if got == exact:
        AGREE += 1
```

```
    else:
        DISAGREE.append((name, macros[name], exact))


def cmp_sci(name, exact):
    """Compare a scientific-notation macro against an exact rational,
    by the rounding bound rather than by re-formatting."""
    global AGREE
    if name not in macros:
        SKIPPED.append((name, "absent"))
        return
    tgt = sci(macros[name])
    if tgt is None:
        SKIPPED.append((name, "unparsed: %r" % macros[name]))
        return
    # The macro carries three significant figures unless it carries two.
    digits = macros[name].split("\\times")[0]
    places = len(digits.split(".")[1]) if "." in digits else 0
    expo = int(re.search(r"10\^\{(-?\d+)\}", macros[name]).group(1))
    unit = Fraction(10) ** (expo - places)
    if abs(Fraction(exact) - tgt) * 2 < unit:
        AGREE += 1
    else:
        DISAGREE.append((name, macros[name], float(exact)))


def cmp_dec(name, exact, places):
    """Compare a fixed-point decimal macro against an exact rational."""
    global AGREE
    if name not in macros:
        SKIPPED.append((name, "absent"))
        return
    tgt = Fraction(macros[name])
    if abs(Fraction(exact) - tgt) * 2 < Fraction(1, 10 ** places):
        AGREE += 1
    else:
        DISAGREE.append((name, macros[name], float(exact)))


def main():
    print("crosscheck_A84.py  --  exact certificates against the published"
          " figures")
    print("reading %s  (%d macros)" % (PATH, len(macros)))
    print("=" * 66)

    fD, fI, fT = admitted(200, Fraction(1, 10))
    arms2, arms3 = ("h", "e"), ("h", "e", "eP")

    # ---- counts ------------------------------------------------------
    cmp_int("VclassesSix", len(fr_orbits(6)))
    cmp_int("VclassesTwelve", closed_form(12)[0])
    cmp_int("VclassesSixteen", closed_form(16)[0])
    cmp_int("VclassTwo", sum(1 for o in fr_orbits(6) if len(o) == 2))
    cmp_int("VclassFour", sum(1 for o in fr_orbits(6) if len(o) == 4))
    cmp_int("VrevOrbitsSix", len(rev_orbits(6)))
    cmp_int("VagrCountSix", len(rev_orbits(5)))
    cmp_int("VwordsSix", 2 ** 6)
    cmp_int("VwordsSixVal", 2 ** 6)
    cmp_int("VwordsTwelveVal", 2 ** 12)
```

```
    cmp_int("VflipIsComplement", 64)
    cmp_int("VrevIsReverse", 64)
    cmp_int("VsigmaOrder", SIGMA)

    subs = [frozenset(c) for r in range(5)
            for c in __import__("itertools").combinations(LOC, r)]
    cmp_int("VsubsetOrbits",
            len({frozenset({S, frozenset(IOTA[k] for k in S)})
                 for S in subs}))

    cmp_int("VperNode", SIGMA * sum(words_up_to_iota(S, arms3)
                                    for _, S in FAMILIES))
    cmp_int("VperNodeDelivered", 3 * SIGMA)
    cmp_int("VtypesTwo", sum(words_up_to_iota(S, arms2) for _, S in FAMILIES))
    cmp_int("VtypesThree", sum(words_up_to_iota(S, arms3)
                               for _, S in FAMILIES))

    cmp_int("VblindDtwo", sum(1 for w in arm_words(D, arms2)
                              if unit_blind(D, w)))
    cmp_int("VblindTtwo", sum(1 for w in arm_words(T, arms2)
                              if unit_blind(T, w)))
    cmp_int("VblindItwo", sum(1 for w in arm_words(I_, arms2)
                              if unit_blind(I_, w)))
    cmp_int("VfixD", 2)
    cmp_int("VfixT", 4)
    cmp_int("VframeCarrierCount",
            sum(1 for _, S in FAMILIES if iota_set(S) != S))

    units = [(S, w) for _, S in FAMILIES for w in arm_words(S, arms3)]
    cmp_int("VcontactPairs", len(units) ** 2)
    cf = sum(1 for S1, w1 in units for S2, w2 in units
             if not unit_blind(S1, w1) and not unit_blind(S2, w2)
             and nu(S1, w1, S2, w2, True) == nu(S1, w1, S2, w2, False))
    cmp_int("VconverseFail", cf)

    dI = {3: "e", 6: "e", 12: "e"}
    cmp_int("VnuII", nu(I_, dI, I_, dI, True))
    cmp_int("VnuIIflip", nu(I_, dI, I_, dI, False))
    cmp_int("VregNuD", 2)
    cmp_int("VregNuT", 4)

    for ell, nm in ((6, "Six"), (8, "Eight"), (16, "Sixteen"),
                    (27, "TwentySeven"), (54, "FiftyFour")):
        cmp_int("Vbits" + nm, len(agr(tuple([0] * ell))))
        cmp_int("Vsite" + nm, ell)
    cmp_int("VbitsCodon", 5)
    cmp_int("VbitsHeritable", 53)
    cmp_int("VedgeBitsSix", 5)
    cmp_int("VedgeSix", 5)
    cmp_int("VedgeHeritable", 53)
    cmp_int("VsiteCodon", 6)
    cmp_int("VsiteHeritable", 54)
    cmp_int("VbitsClassMin", 4)
    cmp_int("VbitsClassMax", 5)

    nx, ny, nz = 3, 3, 6
    cmp_int("VminComplete", 27)
    cmp_int("VminHeritable", nx * ny * nz)
    cmp_int("Vbridges", (nx - 1) * ny * nz + nx * (ny - 1) * nz)
```

```
    cmp_int("VterminalsFree", (nx - 1) * ny * nz + nx * (ny - 1) * nz)
    cmp_int("Vstacks", nx * ny * (nz - 1))
    cmp_int("VlatticeAlphabet", SIGMA)
    cmp_int("VforbidFactor", 16)
    cmp_int("VproductsBoth", len(fr_orbits(6)))

    degs = sorted(lateral_degree(x, y, nx, ny)
                  for x in range(nx) for y in range(ny))
    arms_of = {"D": 2, "I": 3, "T": 4}
    per_site = [sum(1 for f in ("D", "I", "T") if arms_of[f] >= d) * SIGMA
                for d in degs]
    layer = 1
    for v in per_site:
        layer *= v
    cmp_int("VlayerReal", layer)
    cmp_int("VlayerFull", (3 * SIGMA) ** 9)

    # ---- digit counts ------------------------------------------------
    for n, nm in ((8, "Eight"), (16, "Sixteen"), (27, "TwentySeven"),
                  (54, "FiftyFour")):
        k = int(macros.get("Vconf%sDigits" % nm, "0") or 0)
        if k and digits_is(12 ** n, k):
            globals()["AGREE"] = globals()["AGREE"] + 1
        elif k:
            DISAGREE.append(("Vconf%sDigits" % nm, k, len(str(12 ** n))))
    for n, nm in ((16, "Sixteen"), (27, "TwentySeven"), (54, "FiftyFour")):
        k = int(macros.get("Vlat%sDigits" % nm, "0") or 0)
        if k and digits_is(4 ** n, k):
            globals()["AGREE"] = globals()["AGREE"] + 1
        elif k:
            DISAGREE.append(("Vlat%sDigits" % nm, k, len(str(4 ** n))))
    k = int(macros.get("VheritableWordsDigits", "0") or 0)
    if k and digits_is(2 ** 54, k):
        globals()["AGREE"] = globals()["AGREE"] + 1
    elif k:
        DISAGREE.append(("VheritableWordsDigits", k, len(str(2 ** 54))))
    k = int(macros.get("VrealFiftyFourDigits", "0") or 0)
    if k and digits_is(layer ** 6, k):
        globals()["AGREE"] = globals()["AGREE"] + 1
    elif k:
        DISAGREE.append(("VrealFiftyFourDigits", k, len(str(layer ** 6))))

    # ---- scientific-notation figures ---------------------------------
    cmp_sci("VfracI", fI)
    cmp_sci("VfreqII", fI * fI)
    cmp_sci("VprobFiftyFourDec", Fraction(1, 2 ** 54))
    cmp_sci("VseqProbHundred", seq_prob(100))
    cmp_sci("VseqProbTwoHundred", seq_prob(200))

    species = ("D", "I", "T")
    alphabet = [(sp, c, r) for sp in species for c in (0, 1) for r in (0, 1)]
    wt = {"D": fD, "I": fI, "T": fT}
    half = Fraction(1, 2)
    pmin = (wt["I"] ** 4) * half ** 8
    pmax = (wt["D"] ** 4) * half ** 8
    cmp_sci("VpminFour", pmin)
    cmp_sci("VpmaxFour", pmax)
    cmp_sci("VpRatioFour", pmax / pmin)
```

```
    full_layer = (3 * SIGMA) ** 9
    cmp_sci("VboxCut", Fraction(full_layer ** 6, layer ** 6))

    # ---- fixed-point decimals ----------------------------------------
    cmp_dec("VlayerCut", Fraction(full_layer, layer), 1)

    # ---- the bit costs, by bracket rather than by value --------------
    # These are the only published figures that are not rational.  Each
    # is certified by the binary-logarithm bracket of verify_A84.py and
    # compared with the macro as a bracket membership, not as a float.
    from verify_A84 import log2_rounds
    for nm, p, extra in (("VseqBitsHundred", seq_prob(100), 1),
                         ("VseqBitsTwoHundred", seq_prob(200), 1),
                         ("VtotalBitsHundred", seq_prob(100), 2 ** 5),
                         ("VtotalBitsTwoHundred", seq_prob(200), 2 ** 5)):
        if nm not in macros:
            SKIPPED.append((nm, "absent"))
            continue
        tenths = int(round(float(Fraction(macros[nm])) * 10))
        n, d = p.denominator * extra, p.numerator
        if log2_rounds(n, d, tenths, 10):
            globals()["AGREE"] = globals()["AGREE"] + 1
        else:
            DISAGREE.append((nm, macros[nm], "outside its bracket"))

    # ---- the shares, from the bracket endpoints ----------------------
    from verify_A84 import share_rounds
    for nm, tenths, want in (("VregistryShareHundred", 250, 20),
                             ("VregistryTermShare", 280, 18)):
        if nm not in macros:
            SKIPPED.append((nm, "absent"))
            continue
        lo = Fraction(2 * tenths - 1, 20)
        hi = Fraction(2 * tenths + 1, 20)
        if int(macros[nm]) == want and share_rounds(Fraction(5), lo, hi, want)
            :
            globals()["AGREE"] = globals()["AGREE"] + 1
        else:
            DISAGREE.append((nm, macros[nm], want))

    # ---- the free iota-orbits on D and T over two arm types ----------
    arms2 = ("h", "e")
    for nm, S in (("VinfD", D), ("VinfT", T)):
        ws = arm_words(S, arms2)
        free = sum(1 for w in ws if iota_on_word(S, w) != w) // 2
        cmp_int(nm, free)

    # ---- the first four frame classes, as codon sets -----------------
    def codon(w):
        return "".join(LETTER[(w[2 * i], w[2 * i + 1])] for i in range(3))

    cls = sorted(sorted(codon(w) for w in o) for o in fr_orbits(6))
    want = (r"\,".join("\\{%s\\}" % ",".join(c) for c in cls[:4])
            + r",\;\ldots")
    if "VclassList" not in macros:
        SKIPPED.append(("VclassList", "absent"))
    elif macros["VclassList"] == want:
        globals()["AGREE"] = globals()["AGREE"] + 1
    else:
```

```
        DISAGREE.append(("VclassList", macros["VclassList"][:40], want[:40]))

    # ---- the printed powers, exponent by exponent --------------------
    # These macros are typeset powers rather than evaluated numbers.  The
    # check is that the exponent printed is the count the article means,
    # which is what a transcription slip would break.
    powers = (("VconfEight", 12, 8), ("VconfSixteen", 12, 16),
              ("VconfTwentySeven", 12, 27), ("VconfFiftyFour", 12, 54),
              ("VlatSixteen", 4, 16), ("VlatTwentySeven", 4, 27),
              ("VlatFiftyFour", 4, 54),
              ("VwordsSixExp", 2, 6), ("VwordsTwelveExp", 2, 12),
              ("VwordsTwentySevenExp", 2, 27),
              ("VwordsFiftyFourExp", 2, 54), ("VheritableWords", 2, 54))
    for nm, base, expo in powers:
        if nm not in macros:
            SKIPPED.append((nm, "absent"))
            continue
        m = re.match(r"^(\d+)\^\{(-?\d+)\}$", macros[nm])
        if m and (int(m.group(1)), int(m.group(2))) == (base, expo):
            globals()["AGREE"] = globals()["AGREE"] + 1
        else:
            DISAGREE.append((nm, macros[nm], "%d^{%d}" % (base, expo)))
    for nm, expo in (("VprobSix", 6), ("VprobFiftyFour", 54)):
        if nm not in macros:
            SKIPPED.append((nm, "absent"))
            continue
        m = re.match(r"^2\^\{-(\d+)\}(?:=1/(\d+))?$", macros[nm])
        ok = m and int(m.group(1)) == expo and (
            m.group(2) is None or int(m.group(2)) == 2 ** expo)
        if ok:
            globals()["AGREE"] = globals()["AGREE"] + 1
        else:
            DISAGREE.append((nm, macros[nm], "2^{-%d}" % expo))

    print("macros read    : %d" % len(macros))
    print("compared       : %d" % (AGREE + len(DISAGREE)))
    print("agree          : %d" % AGREE)
    print("disagree       : %d" % len(DISAGREE))
    print("symbolic       : %d  (formulae, point groups, set names, words:"
          % (len(macros) - AGREE - len(DISAGREE) - len(SKIPPED)))
    print("                  nothing for integer arithmetic to certify)")
    if SKIPPED:
        print("not compared   : %d" % len(SKIPPED))
        for nm, why in SKIPPED:
            print("    %-28s %s" % (nm, why))
    if DISAGREE:
        print("\nDISAGREEMENTS:")
        for nm, pub, ex in DISAGREE:
            print("    %-28s published %-22s exact %s" % (nm, pub, ex))
    print("=" * 66)
    return 1 if DISAGREE else 0


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

Lean 4 Proof Summary: `acf-84-proof.lean`

Lean Kernel Certification

The file `acf-84-proof.lean` contains 171 theorems in the namespace `ChryseneFormalism.Article84`, 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 171 theorems returns *"does not depend on any axioms"* for **every one of them**. The file assumes nothing beyond the type theory itself. 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-84-proof.tex`.

Elaboration takes about 77 seconds, and about fifty of them are the exhaustive search of S7: the 13 689 ordered contacts, every one of them reduced by the kernel. The binary-logarithm brackets — the file's headline device, and the reason it needs no real-number library — cost under a tenth of a second *in total*, the 397-digit comparison of the species term included.

Where the harness and the proof file divide the work

The two artifacts are not duplicates, and neither subsumes the other.
The Lean file certifies the *combinatorics and the arithmetic*: every orbit count, blindness count, registration number, probability and printed figure as an equation or an inequality between naturals, and every bracket as one that fails if the figure moves by a unit in its last place. It is the stricter instrument, and where it applies it applies absolutely.
The harness does three things the Lean file does not. It *derives* the quantities rather than restating them — the arm-word counts are enumerated and then compared against $m(m+1)/2$, m^3 and $(m^4+m^2)/2$, and the datum set is constructed so that its cardinality is counted — so an error in a formula fails the certificate, where a file that asserted both sides would change them together. It builds the *draw law* over all 20 736 configurations of a four-site column and checks it normalised and of full support at four values of (θ, γ); the Lean file certifies only the multinomial identity that makes it normalised. And it holds the *obligations* and the review record in block P15, so the six hypotheses, four Targets and eight precedence rulings travel with the numbers rather than with the prose.

Block P15 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 $6 = 6$ would be a decoration rather than a result.