

Computational Verification Code: Article 45

The Twenty Registry Classes of a Chrysene Stack

Frame Orbits, Descent of a Letter Transformation, and a Variational Criterion

Author: Charles D. Schaper, Ph.D. | **Artifact:** `verify_twenty_classes.py` → console report (750 lines, 45 checks)

Verification Certificate & Reproducibility Protocol

Every enumerable assertion of Article 45 — each orbit count, stabiliser type, group order, fixed-point count, equivalence of conditions, printed table row, and worked numerical example — is recomputed by the standalone script `verify_twenty_classes.py` listed in Section 3. The script imports nothing from the manuscript: the alphabet, the frame group, the registry sequence and the two transformations are rebuilt from their definitions, so that agreement is evidence rather than restatement.

- **Zero External Dependencies:** Python 3.8 or later, standard library only (`itertools`, `random`, `sys`, `math.comb`); no third-party package, which secures long-term reproducibility.
- **Determinism:** The two pseudorandom weight vectors used in Block C5 are drawn after `random.seed(20260913)`; every other computation is an exhaustive enumeration. Repeated runs on any platform produce identical output.
- **Execution Command:** Run `python3 verify_twenty_classes.py` for the summary, or with the `-v` flag for the per-check transcript reproduced in Section 4. The process exits 0 if every check passes and 1 otherwise, so the script serves as a build-time gate.
- **Verification Status: PASSED** — 45 of 45 checks in Blocks C1–C6; No discrepancy found.; exit code 0.

On the absence of a generated macro file. The companion script of several articles in this corpus emits a `.tex` file of macro definitions which the manuscript then inputs, so that a printed constant cannot drift from the computation that produced it. No such file is emitted here, and none is wanted. The quantities of this article are structural rather than numerical — an orbit count, a stabiliser type, how many of twenty-four bijections descend — and they occur inside the statements of theorems, where a macro would obscure the statement. The script therefore *asserts* the printed values and reports pass or fail, and Section 4 carries the console transcript in place of a macro listing. Appendix A of the article is the one place where a table of computed data is printed, and Block C6 compares that table row by row against a recomputation, which is the same guarantee reached by a different route.

1 Index of Verification Blocks (C1–C6)

The six blocks are the six verification routines of the script, which follow the sections of the article. Table 1 maps each block to the numbered results it checks.

Table 1: Mapping between verification blocks in `verify_twenty_classes.py` and numbered results in Article 45.

Block	Manuscript Reference	Checks	Verified Mathematical Identity / Output
C1	Lem. 2.15, Rem. 2.17	4	Exact enumeration. The two frame actions agree: W is a bijection $\{0, 1\}^{2N} \rightarrow \Sigma^N$ with $W \circ \text{flip} = \rho^{\times N} \circ W$ and $W \circ \text{rev} = \text{reversal} \circ \tau^{\times N} \circ W$; the offset-0 convention is exhibited on the witness FFFBBF.
C2	Lem. 3.1, Thm. 3.3, Prop. 3.6, Cor. 3.7, Prop. 3.15	13	Exact enumeration. Orbit count $2^{2N-2} + 2^{N-1}$ for $2N = 2, \dots, 14$, hence twenty at $2N = 6$; the fixed-point vector $(2^{2N}, 0, 2^N, 2^N)$ underlying Proof 1, and the disjointness used by Proofs 2 and 3; $\bar{D}$ constant on classes and separating them; no class of size one, size 2 or 4 according to stabiliser type, with the central entry D_N distinguishing the two size-2 families; type counts $2^{N-1}, 2^{N-1}, 2^{2N-2} - 2^{N-1}$; and Trap 4, namely that classes agree with multisets only at $N = 3$ (72 against 35 at $N = 4$) and that the fibre profile $\{2:8, 4:12\}$ differs from that of the standard code table.
C3	Lem. 4.4, Lem. 4.5, Prop. 4.6, Thm. 4.9, Prop. 4.11, Rem. 4.13, Prop. 4.14, Ex. 4.15	11	Exact enumeration. flip is the unique fixed-point-free element of Fr ; eight isomorphisms, of which four preserve and four exchange the partition classes; conditions (i)–(v) of the descent theorem are equivalent across all 24 bijections for $N = 1, \dots, 4$, with exactly eight descending; (iv') is strictly stronger, holding for precisely the four class-preserving maps when $N \geq 2$, with witness $T = (02)(13)$ satisfying $T\tau T^{-1} = \tau\rho$; and the transformation of Ex. 4.15 carries one class onto a set meeting four classes, of sizes two, two, four and four.
C4	Prop. 5.3, Prop. 5.5, Thm. 5.8	7	Exact enumeration. The kernel of the induced action is $\{\text{id}, \rho\}$ for $N \geq 2$ and all of G at $N = 1$, so that Ψ is two-to-one onto four induced maps with fibres the $\langle \rho \rangle$ -cosets; κ and λ are well defined modulo reversal; λ acts freely for every N — verified rather than asserted, $\text{Fix}(\lambda) = \emptyset$ at $N = 2, \dots, 5$ against $\text{Fix}(\kappa) = 2^N$ and $\text{Fix}(\kappa\lambda)$ nonempty for even N ; κ preserves D_N on reversal-symmetric classes always and on all classes exactly when N is odd; and $T \circ \rho = c' \circ T$ for every admissible T .
C5	Cor. 6.2, Prop. 6.4, Prop. 6.11, Ex. 6.14–6.16, Rem. 6.18	8	Exact dyadic arithmetic, with one seeded pseudorandom weight. The four values $\frac{1}{2}, 1, 2, 4$ of Ex. 6.14 with unique maximiser $\kappa\lambda$; a λ -invariant weight tying them in two pairs; a point of the degeneracy locus Z at which neither factor carries a symmetry and yet the maximiser is unique, which is the witness for $Z \supsetneq Z_{\max}$; each of the six pair conditions a nonzero bilinear form; the entropy of the pushforward constant on the eight admissible maps; the word-level gap identity $\Lambda(T) - \Lambda(T\rho) = \langle v - \text{flip}_* v, v \circ T \rangle$ on a seeded random weight, and at most four word-level values for a frame-invariant weight; and ten orbits of $\langle \lambda \rangle$ on the twenty classes.
C6	Table A.1 of Appendix A	2	Exact enumeration. All twenty printed rows — representative, $\bar{D}$, class size, stabiliser type, D_N , and the images under κ and λ — agree with a recomputation from the definitions, and exactly eight classes carry the asterisk marking them κ -fixed.

2 Scope of the Computation

A verification certificate is informative only if its boundary is stated. This script decides statements that quantify over a finite set, and nothing else. Three classes of claim in Article 45 lie outside it, and are discharged in the manuscript by citation, by hypothesis, or by reading.

1. **Statements about encoded stacks rather than binary strings.** Everything computed here is a statement about $\{0, 1\}^{2N}$ and about words over a four-letter alphabet. The passage from these to an encoded chrysene stack is the content of the realization hypotheses named where used in the article; the script neither assumes nor tests them.
2. **The genetic-code datum.** The fibre profile of the standard code table is an external empirical datum, cited. The script checks only its internal arithmetic (twenty amino acids; $61 + 3 = 64$ codons) and the inequality of the two profiles on which Prop. 3.15(ii) turns. That the quoted table is the correct one is a matter for the cited source.
3. **Provenance, and negative claims about usage.** Which axioms a proposition invokes, the status label a statement carries, and any claim of the form “no statement of this paper uses X ” are properties of the text. They were established by reading, and are recorded in the concordance and the defect ledger of Appendix B; no computation can confirm them.

Within that boundary the coverage is exhaustive rather than sampled. Where the article states a result for general N , the script confirms it for every N in a range — to $2N = 14$ for the orbit count, $N \leq 4$ for the twenty-four bijections, $N \leq 5$ for the free action of λ — and in each case the general proof is in the article. The computation is a check against transcription and arithmetic error, not a substitute for a proof.

3 Complete Verification Script (verify_twenty_classes.py)

Copy-Paste Usage: Select the code listing below directly from this PDF, save it as `verify_twenty_classes.py`, and run `python3 verify_twenty_classes.py -v`.

```

1 #!/usr/bin/env python3
2 """
3 verify_twenty_classes.py
4 =====
5
6 Independent verification harness for
7
8     C. D. Schaper, "The Twenty Registry Classes of a Chrysene Stack:
9     Frame Orbits, Descent of a Letter Transformation, and a Variational
10     Criterion", Annals of the Chrysene Formalism, 1(2), 676-704, 2026.
11     Article 45. DOI: https://doi.org/10.68366/vfjz1403
12
13 Every enumerable claim of that paper is recomputed here from its own
14 definitions. Nothing is imported from the manuscript: the alphabet, the
15 frame group, the registry sequence and the transformations are rebuilt from
16 scratch below, so that agreement is evidence and not a restatement.
17
18 Scope and limits
19 -----
20 This script checks statements that quantify over a finite set. It does not
21 check, and cannot check:
22
23 * statements about encoded stacks as opposed to binary strings, which
24   depend on the realization hypotheses named in the manuscript;
25 * the genetic-code fibre profile, which is an external datum cited to
26   [Cri68]; only its internal arithmetic is checked here;
27 * the provenance of the axioms, the status labels, or any claim of the
28   form "no statement of this paper uses  $X$ ".
29

```

```

30 Requirements: Python 3.8+, standard library only. Deterministic; the one
31 randomised check uses a fixed seed.
32
33 Usage: python3 verify_twenty_classes.py [-v]
34 Exit code 0 if every check passes, 1 otherwise.
35 """
36
37 import itertools
38 import random
39 import sys
40 from math import comb
41
42 VERBOSE = "-v" in sys.argv[1:]
43 RESULTS = []
44
45
46 def check(tag, statement, ok, detail=""):
47     RESULTS.append((tag, statement, bool(ok), detail))
48     if VERBOSE or not ok:
49         mark = "PASS" if ok else "FAIL"
50         print(f" [{mark}] {tag:<12} {statement}")
51         if detail:
52             print(f"           {detail}")
53
54
55 # -----
56 # 1. Markers, registry sequences, and the frame group on  $\{0,1\}^{2N}$ 
57 # -----
58
59 def flip_m(m):
60     return tuple(1 - x for x in m)
61
62
63 def rev_m(m):
64     return tuple(reversed(m))
65
66
67 def Dseq(m):
68     """Registry sequence, Definition 2.11:  $D_n = m_n + m_{n+1}$  in  $\mathbb{Z}/2$ ."""
69     return tuple((m[i] + m[i + 1]) % 2 for i in range(len(m) - 1))
70
71
72 def reverse(t):
73     return tuple(reversed(t))
74
75
76 def Dbar(d):
77     """Class of a registry sequence modulo reversal, Definition 2.11."""
78     return min(d, reverse(d))
79
80
81 def frame_orbit(m):
82     return frozenset({m, flip_m(m), rev_m(m), flip_m(rev_m(m))})
83
84
85 def marker_classes(twoN):
86     seen, out = set(), []
87     for m in itertools.product((0, 1), repeat=twoN):

```

```

88     if m in seen:
89         continue
90     O = frame_orbit(m)
91     seen |= O
92     out.append(0)
93     return out
94
95
96 def stabiliser_type(m):
97     """'r' for {id,rev}, 'fr' for {id,flip rev}, '1' for trivial."""
98     if rev_m(m) == m:
99         return "r"
100     if flip_m(rev_m(m)) == m:
101         return "fr"
102     return "1"
103
104
105 # -----
106 # 2. Section 3: the classification
107 # -----
108
109 def section3():
110     print("Section 3 - the orbit classification")
111
112     # Lemma 3.1, equation (1), and the fixed-point counts of Proof 1.
113     ok_count, ok_fix, detail = True, True, []
114     for N in range(1, 8):
115         twoN = 2 * N
116         cls = marker_classes(twoN)
117         predicted = 2 ** (twoN - 2) + 2 ** (N - 1)
118         ok_count &= len(cls) == predicted
119         words = list(itertools.product((0, 1), repeat=twoN))
120         f_id = len(words)
121         f_flip = sum(1 for m in words if flip_m(m) == m)
122         f_rev = sum(1 for m in words if rev_m(m) == m)
123         f_fr = sum(1 for m in words if flip_m(rev_m(m)) == m)
124         ok_fix &= (f_id, f_flip, f_rev, f_fr) == (2 ** twoN, 0, 2 ** N, 2 ** N)
125         detail.append(f"2N={twoN}: {len(cls)}")
126     check("Lem 3.1", "orbit count  $2^{(2N-2)+2^{(N-1)}}$  for  $2N=2..14$ ",
127           ok_count, "; ".join(detail))
128     check("Lem 3.1", "fixed-point counts ( $2^{2N}$ , 0,  $2^N$ ,  $2^N$ ) of Proof 1", ok_fix)
129
130     # Proof 2: palindromes and antipalindromes are disjoint and orbit-closed.
131     ok = True
132     for N in range(1, 6):
133         twoN = 2 * N
134         pal = {m for m in itertools.product((0, 1), repeat=twoN) if rev_m(m) == m}
135         anti = {m for m in itertools.product((0, 1), repeat=twoN)
136                if flip_m(rev_m(m)) == m}
137         ok &= not (pal & anti)
138         ok &= len(pal) == len(anti) == 2 ** N
139         ok &= all(flip_m(m) in pal for m in pal)
140         ok &= all(flip_m(m) in anti for m in anti)
141     check("Lem 3.1", "Proof 2: palindromes/antipalindromes disjoint, flip-closed", ok)
142
143     # Proof 3: D is a homomorphism with kernel the constants, and is onto.
144     ok = True
145     for N in range(1, 6):

```

```

146     twoN = 2 * N
147     img = {Dseq(m) for m in itertools.product((0, 1), repeat=twoN)}
148     ok &= len(img) == 2 ** (twoN - 1)
149     ker = [m for m in itertools.product((0, 1), repeat=twoN)
150            if Dseq(m) == tuple([0] * (twoN - 1))]
151     ok &= len(ker) == 2
152     ok &= all(Dseq(flip_m(m)) == Dseq(m)
153              for m in itertools.product((0, 1), repeat=twoN))
154     ok &= all(Dseq(rev_m(m)) == reverse(Dseq(m))
155              for m in itertools.product((0, 1), repeat=twoN))
156     check("Lem 3.1", "Proof 3: D onto, kernel the constants, flip/rev behaviour", ok)
157
158     # Theorem 3.3(i)-(ii): Dbar is a complete invariant of the class.
159     ok = True
160     for N in range(1, 6):
161         twoN = 2 * N
162         cls = marker_classes(twoN)
163         vals = {Dbar(Dseq(next(iter(0)))) for 0 in cls}
164         ok &= len(vals) == len(cls)
165         for 0 in cls:
166             ok &= len({Dbar(Dseq(m)) for m in 0}) == 1
167     check("Thm 3.3", "Dbar is constant on classes and separates them", ok)
168
169     # Proposition 3.6: stabiliser types, sizes, central-entry criterion.
170     ok_size, ok_crit, ok_counts = True, True, True
171     for N in range(1, 7):
172         twoN = 2 * N
173         cls = marker_classes(twoN)
174         n_r = n_fr = n_gen = 0
175         for 0 in cls:
176             m = min(0)
177             st = stabiliser_type(m)
178             size = len(0)
179             ok_size &= size in (2, 4)
180             ok_size &= (size == 2) == (st in ("r", "fr"))
181             d = Dseq(m)
182             # (iii) size two iff the registry sequence is reversal-symmetric
183             ok_crit &= (size == 2) == (d == reverse(d))
184             # (iv) central entry separates the two types
185             if size == 2:
186                 ok_crit &= (d[N - 1] == 0) == (st == "r")
187                 ok_crit &= (d[N - 1] == 1) == (st == "fr")
188             # (v) the central entry is a class function
189             ok_crit &= len({Dseq(x)[N - 1] for x in 0}) == 1
190             n_r += st == "r"
191             n_fr += st == "fr"
192             n_gen += st == "1"
193         ok_counts &= (n_r, n_fr, n_gen) == (
194             2 ** (N - 1), 2 ** (N - 1), 2 ** (2 * N - 2) - 2 ** (N - 1))
195     check("Prop 3.6", "no class of size one; sizes 2 or 4 iff stabiliser type", ok_size)
196     check("Prop 3.6", "size two iff D reversal-symmetric; central entry criterion", ok_crit)
197     check("Prop 3.6", "type counts 2^(N-1), 2^(N-1), 2^(2N-2)-2^(N-1)", ok_counts)
198
199     # Corollary 3.7 at 2N = 6.
200     cls6 = marker_classes(6)
201     sizes = sorted(len(0) for 0 in cls6)
202     n_r = sum(1 for 0 in cls6 if stabiliser_type(min(0)) == "r")
203     n_fr = sum(1 for 0 in cls6 if stabiliser_type(min(0)) == "fr")

```

```

204 check("Cor 3.7", "at 2N=6: twenty classes, eight of size two (4 + 4), twelve of size four",
205       len(cls6) == 20 and sizes.count(2) == 8 and sizes.count(4) == 12
206       and n_r == 4 and n_fr == 4)
207
208 # Proposition 3.15(i): the multiset coincidence holds at N = 3 only.
209 agree = [N for N in range(1, 12)
210          if 2 ** (2 * N - 2) + 2 ** (N - 1) == comb(N + 3, 3)]
211 check("Prop 3.15", "classes = multisets only at N=3 (72 vs 35 at N=4)",
212       agree == [3] and (2 ** 6 + 2 ** 3, comb(7, 3)) == (72, 35),
213       f"agreement at N in {agree}")
214
215 # Proposition 3.15(ii): fibre profile of q at N = 3, against the quoted code.
216 prof_q = {}
217 for 0 in cls6:
218     prof_q[len(0)] = prof_q.get(len(0), 0) + 1
219 code = {1: 2, 2: 9, 3: 1, 4: 5, 6: 3}
220 coding = sum(k * v for k, v in code.items())
221 check("Prop 3.15", "fibre profile of q is {2: 8, 4: 12}", prof_q == {2: 8, 4: 12})
222 check("Prop 3.15", "quoted code datum is internally consistent: 20 amino acids, 61+3=64 codons",
223       sum(code.values()) == 20 and coding == 61 and coding + 3 == 64)
224 check("Prop 3.15", "the two fibre profiles differ, so the surjections are inequivalent",
225       prof_q != code)
226
227
228 # -----
229 # 3. The structured alphabet, and Lemma 2.15
230 # -----
231
232 LETTERS = ("FF", "FB", "BF", "BB")
233 CLASSOF = {"FF": 0, "BB": 0, "FB": 1, "BF": 1} # registry partition
234 RHO = {"FF": "BB", "BB": "FF", "FB": "BF", "BF": "FB"} # fixed-point-free
235 TAU = {"FF": "FF", "BB": "BB", "FB": "BF", "BF": "FB"} # fixes the aligned class
236
237
238 def word_of(m, f=0):
239     """Offset-0 marker-to-word map W of Lemma 2.15, in marker coordinates."""
240     out = []
241     for i in range(len(m) // 2):
242         b1 = (m[2 * i] + m[2 * i + 1]) % 2
243         b2 = (m[2 * i + 1] + f) % 2
244         out.append(({0, 0}: "FF", (0, 1): "BB", (1, 1): "FB", (1, 0): "BF"}[(b1, b2)])
245     return tuple(out)
246
247
248 def flip_w(w, rho=RHO):
249     return tuple(rho[x] for x in w)
250
251
252 def rev_w(w, tau=TAU):
253     return tuple(reversed([tau[x] for x in w]))
254
255
256 def word_orbit(w, rho=RHO, tau=TAU):
257     r = rev_w(w, tau)
258     return frozenset({w, flip_w(w, rho), r, flip_w(r, rho)})
259
260
261 def word_classes(N, rho=RHO, tau=TAU):

```

```

262 seen, out = set(), []
263 for w in itertools.product(LETTERS, repeat=N):
264     if w in seen:
265         continue
266     0 = word_orbit(w, rho, tau)
267     seen |= 0
268     out.append(0)
269 return out
270
271
272 def section2():
273     print("Section 2 - the offset convention and the two frame groups")
274
275     ok_bij = ok_flip = ok_rev = True
276     for N in range(1, 6):
277         twoN = 2 * N
278         ms = list(itertools.product((0, 1), repeat=twoN))
279         img = {word_of(m) for m in ms}
280         ok_bij &= len(img) == len(ms) == 4 ** N
281         for m in ms:
282             ok_flip &= word_of(flip_m(m)) == flip_w(word_of(m))
283             ok_rev &= word_of(rev_m(m)) == rev_w(word_of(m))
284         check("Lem 2.15", "W is a bijection  $\{0,1\}^{2N} \rightarrow \Sigma^N$ ", ok_bij)
285         check("Lem 2.15", "W intertwines marker flip with letterwise rho", ok_flip)
286         check("Lem 2.15", "W intertwines marker reversal with reversal o tau", ok_rev)
287
288     # Remark 2.17: the witness of the predecessor.
289     m = (0, 0, 0, 1, 1, 0)
290     check("Rem 2.17", "witness FFFBBF: word (FF,FB,BF); images under flip and rev",
291           word_of(m) == ("FF", "FB", "BF")
292           and word_of(flip_m(m)) == ("BB", "BF", "FB")
293           and word_of(rev_m(m)) == ("FB", "BF", "FF"))
294
295
296 # -----
297 # 4. Section 4: descent
298 # -----
299
300 def perms_of_letters():
301     return [dict(zip(LETTERS, p)) for p in itertools.permutations(LETTERS)]
302
303
304 def preserves_partition(T):
305     """Carries the registry partition to itself (either class order)."""
306     img0 = {T[x] for x in LETTERS if CLASSOF[x] == 0}
307     return img0 in ({ "FF", "BB" }, { "FB", "BF" })
308
309
310 def preserves_each_class(T):
311     return all(CLASSOF[T[x]] == CLASSOF[x] for x in LETTERS)
312
313
314 def conj(T, g):
315     """T g T-1 as a permutation of the target letters."""
316     Tinv = {v: k for k, v in T.items()}
317     return {y: T[g[Tinv[y]]] for y in LETTERS}
318
319

```

```

320 def section4():
321     print("Section 4 - descent")
322
323     # Lemma 4.4(ii): fixed-point counts of the frame group on Sigma^N.
324     ok = True
325     for N in range(1, 7):
326         ws = list(itertools.product(LETTERS, repeat=N))
327         a = sum(1 for w in ws if flip_w(w) == w)
328         b = sum(1 for w in ws if rev_w(w) == w)
329         c = sum(1 for w in ws if flip_w(rev_w(w)) == w)
330         ok &= (a, b, c) == (0, 2 ** N, 2 ** N)
331     check("Lem 4.4", "flip is the unique fixed-point-free element of Fr (0, 2^N, 2^N)", ok)
332
333     # Lemma 4.4(i): Fr does not depend on which non-fpf element is named tau.
334     TAU2 = {"FF": "BB", "BB": "FF", "FB": "FB", "BF": "BF"} # = tau rho
335     ok = True
336     for N in range(1, 5):
337         for w in itertools.product(LETTERS, repeat=N):
338             ok &= word_orbit(w, RHO, TAU) == word_orbit(w, RHO, TAU2)
339     check("Lem 4.4", "the group Fr is unchanged by the substitution tau -> tau rho", ok)
340
341     # Lemma 4.5: exactly eight isomorphisms of structured alphabets.
342     isos = [T for T in perms_of_letters() if preserves_partition(T)]
343     check("Lem 4.5", "eight isomorphisms; four preserve each class, four exchange them",
344           len(isos) == 8 and sum(1 for T in isos if preserves_each_class(T)) == 4)
345
346     # Proposition 4.6 / Lemma 4.5: the target class count is independent of T.
347     ok = True
348     for N in (1, 2, 3):
349         base = len(word_classes(N))
350         for T in isos:
351             rho2 = conj(T, RHO)
352             tau2 = conj(T, TAU)
353             ok &= len(word_classes(N, rho2, tau2)) == base
354     check("Prop 4.6", "the target classification has the same count for every structure", ok)
355
356     # Theorem 4.9: the five conditions agree, and exactly eight hold.
357     ok_equiv, ok_count, ok_ivp = True, True, True
358     for N in (1, 2, 3, 4):
359         classes = word_classes(N)
360         classset = set(classes)
361         n_true = 0
362         for T in perms_of_letters():
363             i = preserves_partition(T)
364             # (ii) T G T^{-1} = G' : conjugation preserves the partition group
365             G = [dict(zip(LETTERS, LETTERS)), RHO, TAU,
366                  {x: RHO[TAU[x]] for x in LETTERS}]
367             ii = all(preserves_each_class(conj(T, g)) for g in G)
368             # (iii) T c T^{-1} = c'
369             iii = conj(T, RHO) == RHO
370             # (v) descent
371             v = all(frozenset(tuple(T[x] for x in w) for w in 0) in classset
372                     for 0 in classes)
373             # (iv) intertwining of the two frame-group actions
374             S = lambda w: tuple(T[x] for x in w)
375             iv = (all(S(flip_w(w)) == flip_w(S(w))
376                      for w in itertools.product(LETTERS, repeat=N))
377                  and (all(S(rev_w(w)) == rev_w(S(w))

```

```

378         for w in itertools.product(LETTERS, repeat=N)
379         or all(S(rev_w(w)) == flip_w(rev_w(S(w)))
380             for w in itertools.product(LETTERS, repeat=N))))
381     ok_equiv &= (i == ii == iii == iv == v)
382     n_true += bool(v)
383     # (iv') literal equivariance for the fixed labelling
384     ivp = (all(S(flip_w(w)) == flip_w(S(w))
385         for w in itertools.product(LETTERS, repeat=N))
386         and all(S(rev_w(w)) == rev_w(S(w))
387             for w in itertools.product(LETTERS, repeat=N)))
388     if N >= 2:
389         ok_ivp &= (ivp == (v and preserves_each_class(T)))
390     ok_count &= n_true == 8
391     check("Thm 4.9", "(i)-(v) are equivalent for all 24 bijections, N=1..4", ok_equiv)
392     check("Prop 4.14", "exactly eight of twenty-four descend", ok_count)
393     check("Prop 4.11", "(iv') holds exactly for the four class-preserving maps, N>=2", ok_ivp)
394
395     # Proposition 4.11 witness: the class-exchanging map (0 2)(1 3).
396     T = {"FF": "FB", "FB": "FF", "BB": "BF", "BF": "BB"}
397     check("Prop 4.11", "witness T=(0 2)(1 3): admissible, and T tau T^-1 = tau rho",
398         preserves_partition(T) and conj(T, RHO) == RHO
399         and conj(T, TAU) == {x: RHO[TAU[x]] for x in LETTERS})
400
401     # Equation (2) and Remark 4.13: size preserved, stabiliser type not.
402     N = 3
403     classes = word_classes(N)
404
405     def wtype(0):
406         w = sorted(0)[0]
407         if len(0) == 4:
408             return "1"
409         return "r" if rev_w(w) == w else "fr"
410
411     ok_size, ok_type = True, True
412     for T in isos:
413         S = lambda w: tuple(T[x] for x in w)
414         rho2, tau2 = conj(T, RHO), conj(T, TAU)
415         for 0 in classes:
416             img = frozenset(S(w) for w in 0)
417             img0 = word_orbit(sorted(img)[0], rho2, tau2)
418             ok_size &= len(img) == len(0) == len(img0)
419             if preserves_each_class(T):
420                 for 0 in classes:
421                     if len(0) == 2:
422                         img = word_orbit(S(sorted(0)[0]), rho2, tau2)
423                         ok_type &= (rev_w(sorted(img)[0], tau2) == sorted(img)[0]) == \
424                             (wtype(0) == "r")
425     check("Thm 4.9", "the induced bijection preserves class size, all eight maps", ok_size)
426     check("Rem 4.13", "class-preserving maps preserve stabiliser type", ok_type)
427
428     # Example 4.15: the counterexample.
429     T = {"FF": "FF", "FB": "FB", "BB": "BF", "BF": "BB"}
430     w0 = ("FF", "FB", "BF")
431     src = [w0, ("BB", "BF", "FB"), ("BF", "FB", "BB"), ("FB", "BF", "FF")]
432     ok = (not preserves_partition(T)) and set(word_orbit(w0)) == set(src)
433     imgs = [tuple(T[x] for x in w) for w in src]
434     sizes = [len(word_orbit(im)) for im in imgs]
435     distinct = len({word_orbit(im) for im in imgs})

```

```

436     check("Ex 4.15", "counterexample: images in four distinct classes, sizes 2,2,4,4",
437           ok and sizes == [2, 2, 4, 4] and distinct == 4,
438           f"images {imgs}")
439
440
441 # -----
442 # 5. Section 5: the induced action
443 # -----
444
445 def kappa_D(d, N):
446     out = list(d)
447     for i in range(1, N):
448         out[2 * i - 1] = (d[2 * i - 2] + d[2 * i - 1] + d[2 * i]) % 2
449     return tuple(out)
450
451
452 def lambda_D(d):
453     return tuple((x + 1) % 2 for x in d)
454
455
456 def section5():
457     print("Section 5 - the induced action on classes")
458
459     # Proposition 5.3(i): the kernel of the action of N on the classes.
460     ok = True
461     detail = []
462     for N in (1, 2, 3, 4):
463         classes = set(word_classes(N))
464         trivial = []
465         for T in [T for T in perms_of_letters() if preserves_partition(T)]:
466             acts_trivially = all(
467                 frozenset(tuple(T[x] for x in w) for w in 0) == 0 for 0 in classes)
468             if acts_trivially:
469                 trivial.append(T)
470             ident = dict(zip(LETTERS, LETTERS))
471             if N == 1:
472                 ok &= len(trivial) == 4
473                 detail.append("N=1: kernel of order 4")
474             else:
475                 ok &= len(trivial) == 2 and RHO in trivial and ident in trivial
476                 detail.append(f"N={N}: kernel {{id, rho}}")
477     check("Prop 5.3", "kernel is {id, rho} for N>=2 and is G for N=1", ok,
478           "; ".join(detail))
479
480     # Proposition 5.3(ii)-(iii): K has order four, acts faithfully, Psi is 2-to-1.
481     N = 3
482     classes = set(word_classes(N))
483     isos = [T for T in perms_of_letters() if preserves_partition(T)]
484     induced = {}
485     for T in isos:
486         key = tuple(sorted(
487             (tuple(sorted(map(str, 0))),
488              tuple(sorted(map(str, frozenset(tuple(T[x] for x in w) for w in 0)))))
489             for 0 in classes))
490         induced.setdefault(key, []).append(T)
491     check("Prop 5.3", "Psi is two-to-one onto four induced maps, fibres of size two",
492           len(induced) == 4 and all(len(v) == 2 for v in induced.values()))
493

```

```

494 # Proposition 5.5: the action in registry coordinates, and freeness.
495 ok_wd = ok_free = ok_par = True
496 detail = []
497 for N in (2, 3, 4, 5):
498     seqs = list(itertools.product((0, 1), repeat=2 * N - 1))
499     X = sorted({Dbar(d) for d in seqs})
500     ok_wd &= all(Dbar(kappa_D(reverse(d), N)) == Dbar(kappa_D(d, N)) for d in seqs)
501     ok_wd &= all(Dbar(kappa_D(c, N)) in X and Dbar(lambda_D(c)) in X for c in X)
502     fix_l = sum(1 for c in X if Dbar(lambda_D(c)) == c)
503     fix_kl = sum(1 for c in X if Dbar(lambda_D(kappa_D(c, N))) == c)
504     fix_k = sum(1 for c in X if Dbar(kappa_D(c, N)) == c)
505     ok_free &= fix_l == 0
506     # kappa fixes the central entry on reversal-symmetric classes always
507     ok_par &= all(kappa_D(c, N)[N - 1] == c[N - 1] for c in X if c == reverse(c))
508     # and on every class exactly when N is odd
509     allfix = all(kappa_D(c, N)[N - 1] == c[N - 1] for c in X)
510     ok_par &= (allfix == (N % 2 == 1))
511     detail.append(f"N={N}: |X|={len(X)}, fix(k)={fix_k}, fix(l)={fix_l}, fix(kl)={fix_kl}")
512 check("Prop 5.5", "kappa and lambda are well defined modulo reversal", ok_wd)
513 check("Prop 5.5", "lambda acts freely for every N", ok_free, "; ".join(detail))
514 check("Prop 5.5", "kappa fixes D_N on symmetric classes always, on all classes iff N odd",
515       ok_par)
516
517 # Proposition 5.5(i) against the letterwise action of tau.
518 N = 3
519 ok = True
520 for m in itertools.product((0, 1), repeat=2 * N):
521     w = word_of(m)
522     w2 = tuple(TAU[x] for x in w)
523     m2 = None
524     for mm in itertools.product((0, 1), repeat=2 * N):
525         if word_of(mm) == w2:
526             m2 = mm
527             break
528     ok &= Dseq(m2) == kappa_D(Dseq(m), N)
529 check("Prop 5.5", "the kappa formula is the letterwise action of tau on D", ok)
530
531 # Theorem 5.8(i): T rho = c' T, and the fibre.
532 ok = True
533 for T in [T for T in perms_of_letters() if preserves_partition(T)]:
534     lhs = {x: T[RHO[x]] for x in LETTERS}
535     rhs = {x: conj(T, RHO)[T[x]] for x in LETTERS}
536     ok &= lhs == rhs and conj(T, RHO) == RHO
537 check("Thm 5.8", "T o rho = c' o T for every admissible T", ok)
538
539
540 # -----
541 # 6. Section 6: the functional
542 # -----
543
544 def section6():
545     print("Section 6 - the variational criterion")
546
547     N = 3
548     seqs = list(itertools.product((0, 1), repeat=2 * N - 1))
549     X = sorted({Dbar(d) for d in seqs})
550     K = {"1": lambda c: c,
551         "k": lambda c: Dbar(kappa_D(c, N)),

```

```

552     "l": lambda c: Dbar(lambda_D(c)),
553     "kl": lambda c: Dbar(lambda_D(kappa_D(c, N)))}
554
555 def Phi(mu, w):
556     return {name: sum(mu.get(c, 0) * w.get(f(c), 0) for c in X)
557            for name, f in K.items()}
558
559 O1, O2 = (0, 0, 0, 0, 0), (0, 0, 1, 0, 1)
560 mu = {O1: 0.5, O2: 0.5}
561
562 # Example 6.14.
563 w = {c: 0 for c in X}
564 w[(0, 0, 1, 0, 1)] = 1
565 w[(0, 1, 1, 0, 1)] = 2
566 w[(0, 1, 0, 1, 1)] = 4
567 w[(0, 1, 0, 0, 1)] = 8
568 vals = Phi(mu, w)
569 check("Ex 6.14", "four values 1/2, 1, 2, 4 with unique maximiser kappa lambda",
570       [vals[k] for k in ("l", "k", "1", "kl")] == [0.5, 1.0, 2.0, 4.0],
571       str(vals))
572
573 # Example 6.15: a constructed tie on the degeneracy locus.
574 w2 = {c: 0 for c in X}
575 w2[(0, 0, 1, 0, 1)] = 1
576 w2[(0, 1, 0, 1, 1)] = 1
577 inv = all(w2[Dbar(lambda_D(c))] == w2[c] for c in X)
578 vals2 = Phi(mu, w2)
579 check("Ex 6.15", "lambda-invariant weight ties the four values in two pairs",
580       inv and [vals2[k] for k in ("l", "k", "1", "kl")] == [0.5, 0.0, 0.5, 0.0],
581       str(vals2))
582
583 # Remark 6.16: ten orbits of <lambda>, hence a ten-dimensional space of ties.
584 orbits = {frozenset({c, Dbar(lambda_D(c))}) for c in X}
585 check("Rem 6.18", "<lambda> has ten orbits on the twenty classes",
586       len(orbits) == 10 and all(len(o) == 2 for o in orbits))
587
588 # A point of Z with no symmetry in either factor, and a unique maximiser.
589 w3 = {c: 0 for c in X}
590 w3[(0, 0, 1, 0, 1)] = 1
591 w3[(0, 1, 1, 0, 1)] = 1
592 w3[(0, 1, 0, 1, 1)] = 5
593 w3[(0, 0, 0, 0, 1)] = 3
594 vals3 = Phi(mu, w3)
595 mu_sym = any(all(mu.get(c, 0) == mu.get(f(c), 0) for c in X)
596              for name, f in K.items() if name != "1")
597 w_sym = any(all(w3[c] == w3[f(c)] for c in X)
598             for name, f in K.items() if name != "1")
599 tied = len({round(v, 12) for v in vals3.values()}) < 4
600 best = max(vals3.values())
601 unique_max = sum(1 for v in vals3.values() if v == best) == 1
602 check("Ex 6.16", "a point of Z with no symmetry of mu or w, and a unique maximiser",
603       tied and unique_max and not mu_sym and not w_sym, str(vals3))
604
605 # Proposition 6.11: each of the six bilinear forms is nonzero.
606 ok = True
607 names = list(K)
608 for a, b in itertools.combinations(names, 2):
609     found = False

```

```

610     for c in X:
611         ka, lb = K[a](c), K[b](c)
612         if ka != lb:
613             mu1 = {c: 1.0}
614             w1 = {x: (1.0 if x == ka else 0.0) for x in X}
615             if abs(Phi(mu1, w1)[a] - Phi(mu1, w1)[b]) > 1e-12:
616                 found = True
617                 break
618     ok &= found
619     check("Prop 6.11", "each of the six pair conditions is a nonzero bilinear form", ok)
620
621     # Proposition 6.1 and Corollary 6.2: relabelling invariants are constant.
622     def entropy(dist):
623         import math
624         return -sum(p * math.log2(p) for p in dist.values() if p > 0)
625
626     isos = [T for T in perms_of_letters() if preserves_partition(T)]
627     words = list(itertools.product(LETTERS, repeat=N))
628     random.seed(20260913)
629     nu = {w: random.random() for w in words}
630     tot = sum(nu.values())
631     nu = {w: p / tot for w, p in nu.items()}
632     ents = set()
633     for T in isos:
634         push = {}
635         for w, p in nu.items():
636             im = tuple(T[x] for x in w)
637             push[im] = push.get(im, 0) + p
638             ents.add(round(entropy(push), 12))
639     check("Cor 6.2", "entropy of the pushforward is constant on the eight", len(ents) == 1)
640
641     # Proposition 6.4(ii): the word-level gap identity.
642     v = {w: random.random() for w in words}
643     ok = True
644     for T in isos:
645         S = lambda w: tuple(T[x] for x in w)
646         Trho = {x: T[RHO[x]] for x in LETTERS}
647         lam1 = sum(nu[w] * v[S(w)] for w in words)
648         lam2 = sum(nu[w] * v[tuple(Trho[x] for x in w)] for w in words)
649         gap = sum((nu[w] - nu[flip_w(w)]) * v[S(w)] for w in words)
650         ok &= abs((lam1 - lam2) - gap) < 1e-12
651     check("Prop 6.4", "word-level gap identity Lambda(T)-Lambda(T rho) = <nu - flip nu, v o T>", ok)
652
653     # Proposition 6.4(i): an Fr-invariant weight collapses the word level.
654     vinv = {}
655     for w in words:
656         O = word_orbit(w)
657         vinv[w] = sum(hash(x) % 7 for x in sorted(map(str, O))) % 5
658     ok = all(vinv[w] == vinv[u] for w in words for u in word_orbit(w))
659     vals = {round(sum(nu[w] * vinv[tuple(T[x] for x in w)]) for w in words), 12)
660             for T in isos}
661     check("Prop 6.4", "an Fr-invariant weight gives at most four word-level values",
662           ok and len(vals) <= 4)
663
664
665     # -----
666     # 7. Appendix A: the printed table
667     # -----

```

```

668
669 TABLE_A = """
670 FFFFFF 00000 2 r 0 00000 11111
671 FFFFFB 00001 4 1 0 00011 01111
672 FFFFBB 00010 4 1 0 00010 10111
673 FFFFBF 00011 4 1 0 00001 00111
674 FFFBBB 00100 2 fr 1 01110 11011
675 FFFBBF 00101 4 1 1 01101 01011
676 FFFBFF 00110 4 1 1 00110 10011
677 FFFBFB 00111 4 1 1 01111 00011
678 FFBBBF 01001 4 1 0 01011 01101
679 FFBBFF 01010 2 r 0 01010 10101
680 FFBBFB 01011 4 1 0 01001 00101
681 FFBFFB 01101 4 1 1 00101 01001
682 FFBFBB 01110 2 fr 1 00100 10001
683 FFBFBF 01111 4 1 1 00111 00001
684 FB BBBF 10001 2 r 0 11011 01110
685 FBFFF B 10011 4 1 0 10011 00110
686 FBFBFB 10101 2 fr 1 10101 01010
687 FBFBFB 10111 4 1 1 10111 00010
688 FBFBFB 11011 2 r 0 10001 00100
689 FBFBFB 11111 2 fr 1 11111 00000
690 """
691
692
693 def appendixA():
694     print("Appendix A - the printed table")
695     N = 3
696     rows = []
697     for 0 in marker_classes(6):
698         rep = min(0)
699         d = Dseq(rep)
700         c = Dbar(d)
701         rows.append((
702             "".join("F" if x == 0 else "B" for x in rep),
703             "".join(map(str, c)),
704             len(0),
705             stabiliser_type(rep),
706             c[N - 1],
707             "".join(map(str, Dbar(kappa_D(c, N)))),
708             "".join(map(str, Dbar(lambda_D(c)))),
709         ))
710     rows.sort(key=lambda r: r[1])
711     printed = []
712     for line in TABLE_A.strip().splitlines():
713         a, b, s, t, dn, k, l = line.split()
714         printed.append((a, b, int(s), t, int(dn), k, l))
715     check("Table A", "all twenty printed rows agree with recomputation",
716         rows == printed,
717         """ if rows == printed else
718         "\n".join(f"printed {p} vs computed {r}"
719                 for p, r in zip(printed, rows) if p != r))
720     starred = sum(1 for r in rows if r[5] == r[1])
721     check("Table A", "eight classes are fixed by kappa", starred == 8)
722
723
724 # -----
725

```

```

726 def main():
727     print(__doc__.split("Usage")[0].strip().splitlines()[0])
728     print("=" * 66)
729     section2()
730     section3()
731     section4()
732     section5()
733     section6()
734     appendixA()
735     print("=" * 66)
736     failed = [r for r in RESULTS if not r[2]]
737     print(f"{len(RESULTS) - len(failed)} of {len(RESULTS)} checks passed.")
738     if failed:
739         print("FAILURES:")
740         for tag, st, _, detail in failed:
741             print(f"  {tag}: {st}")
742             if detail:
743                 print(f"    {detail}")
744         return 1
745     print("No discrepancy found.")
746     return 0
747
748
749 if __name__ == "__main__":
750     sys.exit(main())

```

Listing 1: Complete standalone verification script `verify_twenty_classes.py` for Article 45: 750 lines, standard library only.

4 Console Transcript

Summary run. \$ python3 verify_twenty_classes.py

```

verify_twenty_classes.py
=====
Section 2 - the offset convention and the two frame groups
Section 3 - the orbit classification
Section 4 - descent
Section 5 - the induced action on classes
Section 6 - the variational criterion
Appendix A - the printed table
=====
45 of 45 checks passed.
No discrepancy found.

```

Exit code 0.

The verbose run below is the auditable record: one line for each of the 45 checks, naming the numbered statement of the article that it tests, with the computed data appended wherever a reader would otherwise have to take the verdict on trust.

```

verify_twenty_classes.py
=====
Section 2 - the offset convention and the two frame groups
[PASS] Lem 2.15      W is a bijection {0,1}^2N -> Sigma^N
[PASS] Lem 2.15      W intertwines marker flip with letterwise rho

```

```

[PASS] Lem 2.15      W intertwines marker reversal with reversal o tau
[PASS] Rem 2.17      witness FFFBBF: word (FF,FB,BF); images under flip and rev
Section 3 - the orbit classification
[PASS] Lem 3.1       orbit count  $2^{(2N-2)+2^{(N-1)}}$  for  $2N=2..14$ 
                    2N=2: 2; 2N=4: 6; 2N=6: 20; 2N=8: 72; 2N=10: 272; 2N=12: 1056; 2N=14: 4160
[PASS] Lem 3.1       fixed-point counts  $(2^{2N}, 0, 2^N, 2^N)$  of Proof 1
[PASS] Lem 3.1       Proof 2: palindromes/antipalindromes disjoint, flip-closed
[PASS] Lem 3.1       Proof 3: D onto, kernel the constants, flip/rev behaviour
[PASS] Thm 3.3       Dbar is constant on classes and separates them
[PASS] Prop 3.6       no class of size one; sizes 2 or 4 iff stabiliser type
[PASS] Prop 3.6       size two iff D reversal-symmetric; central entry criterion
[PASS] Prop 3.6       type counts  $2^{(N-1)}, 2^{(N-1)}, 2^{(2N-2)-2^{(N-1)}}$ 
[PASS] Cor 3.7       at  $2N=6$ : twenty classes, eight of size two (4 + 4), twelve of size four
[PASS] Prop 3.15      classes = multisets only at  $N=3$  (72 vs 35 at  $N=4$ )
                    agreement at N in [3]
[PASS] Prop 3.15      fibre profile of q is {2: 8, 4: 12}
[PASS] Prop 3.15      quoted code datum is internally consistent: 20 amino acids, 61+3=64 codons
[PASS] Prop 3.15      the two fibre profiles differ, so the surjections are inequivalent
Section 4 - descent
[PASS] Lem 4.4       flip is the unique fixed-point-free element of Fr  $(0, 2^N, 2^N)$ 
[PASS] Lem 4.4       the group Fr is unchanged by the substitution  $\tau \rightarrow \tau \rho$ 
[PASS] Lem 4.5       eight isomorphisms; four preserve each class, four exchange them
[PASS] Prop 4.6       the target classification has the same count for every structure
[PASS] Thm 4.9       (i)-(v) are equivalent for all 24 bijections,  $N=1..4$ 
[PASS] Prop 4.14      exactly eight of twenty-four descend
[PASS] Prop 4.11      (iv') holds exactly for the four class-preserving maps,  $N \geq 2$ 
[PASS] Prop 4.11      witness  $T=(0\ 2)(1\ 3)$ : admissible, and  $T \tau T^{-1} = \tau \rho$ 
[PASS] Thm 4.9       the induced bijection preserves class size, all eight maps
[PASS] Rem 4.13       class-preserving maps preserve stabiliser type
[PASS] Ex 4.15       counterexample: images in four distinct classes, sizes 2,2,4,4
                    images [('FF', 'FB', 'BB'), ('BF', 'BB', 'FB'), ('BB', 'FB', 'BF'), ('FB', 'BB', 'FF')]
Section 5 - the induced action on classes
[PASS] Prop 5.3       kernel is {id, rho} for  $N \geq 2$  and is G for  $N=1$ 
                    N=1: kernel of order 4; N=2: kernel {id, rho}; N=3: kernel {id, rho}; N=4: kernel {id, rho}
[PASS] Prop 5.3       Psi is two-to-one onto four induced maps, fibres of size two
[PASS] Prop 5.5       kappa and lambda are well defined modulo reversal
[PASS] Prop 5.5       lambda acts freely for every N
                    N=2:  $|X|=6$ ,  $\text{fix}(k)=4$ ,  $\text{fix}(l)=0$ ,  $\text{fix}(kl)=2$ ; N=3:  $|X|=20$ ,  $\text{fix}(k)=8$ ,  $\text{fix}(l)=0$ ,  $\text{fix}(kl)=0$ ; N=4:  $|X|=72$ ,  $\text{fix}(k)=16$ ,
                     $\text{fix}(l)=0$ ,  $\text{fix}(kl)=8$ ; N=5:  $|X|=272$ ,  $\text{fix}(k)=32$ ,  $\text{fix}(l)=0$ ,  $\text{fix}(kl)=0$ 
[PASS] Prop 5.5       kappa fixes  $D_N$  on symmetric classes always, on all classes iff N odd
[PASS] Prop 5.5       the kappa formula is the letterwise action of tau on D
[PASS] Thm 5.8        $T \circ \rho = c' \circ T$  for every admissible T
Section 6 - the variational criterion
[PASS] Ex 6.14       four values 1/2, 1, 2, 4 with unique maximiser kappa lambda
                    {'1': 0.5, 'k': 1.0, 'l': 2.0, 'kl': 4.0}
[PASS] Ex 6.15       lambda-invariant weight ties the four values in two pairs
                    {'1': 0.5, 'k': 0.0, 'l': 0.5, 'kl': 0.0}
[PASS] Rem 6.18       <lambda> has ten orbits on the twenty classes
[PASS] Ex 6.16       a point of Z with no symmetry of mu or w, and a unique maximiser
                    {'1': 0.5, 'k': 0.5, 'l': 2.5, 'kl': 0.0}
[PASS] Prop 6.11      each of the six pair conditions is a nonzero bilinear form
[PASS] Cor 6.2       entropy of the pushforward is constant on the eight
[PASS] Prop 6.4       word-level gap identity  $\text{Lambda}(T) - \text{Lambda}(T \rho) = \langle \text{nu} - \text{flip nu}, v \circ T \rangle$ 
[PASS] Prop 6.4       an Fr-invariant weight gives at most four word-level values
Appendix A - the printed table
[PASS] Table A       all twenty printed rows agree with recomputation
[PASS] Table A       eight classes are fixed by kappa
=====
45 of 45 checks passed.
No discrepancy found.

```

Listing 2: Per-check transcript, python3 verify_twenty_classes.py -v.